SS 2014

Efficient Algorithms and Data Structures II

Harald Räcke

Fakultät für Informatik
TU München

http://www14.in.tum.de/lehre/2014SS/ea/

Summer Term 2014

Part I

Organizational Matters

Part I

Organizational Matters

- Modul: IN2004
- Name: “Efficient Algorithms and Data Structures II”
  “Effiziente Algorithmen und Datenstrukturen II”
- ECTS: 8 Credit points
- Lectures:
  - 4 SWS
    Mon 12:15–14:45 (Room 00.13.009A)
    Fri 10:15–11:45 (Room 00.13.009A)
- Webpage: `http://www14.in.tum.de/lehre/2014SS/ea/`

Part I

Organizational Matters

- Modul: IN2004
- Name: "Efficient Algorithms and Data Structures II"
  "Effiziente Algorithmen und Datenstrukturen II"
- ECTS: 8 Credit points
- Lectures:
  - 4 SWS
    Mon 12:15–14:45 (Room 00.13.009A)
    Fri 10:15–11:45 (Room 00.13.009A)
- Webpage: `http://www14.in.tum.de/lehre/2014SS/ea/`

Part I

Organizational Matters

- Modul: IN2004
- Name: "Efficient Algorithms and Data Structures II"
  "Effiziente Algorithmen und Datenstrukturen II"
- ECTS: 8 Credit points
- Lectures:
  - 4 SWS
    Mon 12:15–14:45 (Room 00.13.009A)
    Fri 10:15–11:45 (Room 00.13.009A)
- Webpage: `http://www14.in.tum.de/lehre/2014SS/ea/`

Part I

Organizational Matters

- Modul: IN2004
- Name: "Efficient Algorithms and Data Structures II"
  "Effiziente Algorithmen und Datenstrukturen II"
- ECTS: 8 Credit points
- Lectures:
  - 4 SWS
    Mon 12:15–14:45 (Room 00.13.009A)
    Fri 10:15–11:45 (Room 00.13.009A)
- Webpage: http://www14.in.tum.de/lehre/2014SS/ea/

Part I

Organizational Matters

- Modul: IN2004
- Name: "Efficient Algorithms and Data Structures II"
  "Effiziente Algorithmen und Datenstrukturen II"
- ECTS: 8 Credit points
- Lectures:
  - 4 SWS
    Mon 12:15–14:45 (Room 00.13.009A)
    Fri 10:15–11:45 (Room 00.13.009A)
- Webpage: `http://www14.in.tum.de/lehre/2014SS/ea/`

The Lecturer

- Harald Räcke
- Email: raecke@in.tum.de
- Room: 03.09.044
- Office hours: (per appointment)

Tutorials

- Tutor:
  - Chintan Shah
  - chintan.shah@tum.de
  - Room: 03.09.059
  - per appointment
- Room: 03.11.018
- Time: Tue 14:15–15:45

Assessment

- In order to pass the module you need to pass an exam.

- Exam:
  - 3 hours
  - Date will be announced shortly.
  - There are no resources allowed, apart from a hand written piece of paper (A4).
  - Answers should be given in English, but German is also accepted.

Assessment

- In order to pass the module you need to pass an exam.
- Exam:
  - 3 hours
  - Date will be announced shortly.
  - There are no resources allowed, apart from a hand-written piece of paper (A4).
  - Answers should be given in English, but German is also accepted.

Assessment

- In order to pass the module you need to pass an exam.
- Exam:
  - 3 hours
  - Date will be announced shortly.
  - There are no resources allowed, apart from a hand-written piece of paper (A4).
  - Answers should be given in English, but German is also accepted.

Assessment

- In order to pass the module you need to pass an exam.
- Exam:
  - 3 hours
  - Date will be announced shortly.
  - There are no resources allowed, apart from a hand-written piece of paper (A4).
  - Answers should be given in English, but German is also accepted.

Assessment

- In order to pass the module you need to pass an exam.
- Exam:
  - 3 hours
  - Date will be announced shortly.
  - There are no resources allowed, apart from a hand-written piece of paper (A4).
  - Answers should be given in English, but German is also accepted.

Assessment

- In order to pass the module you need to pass an exam.
- Exam:
  - 3 hours
  - Date will be announced shortly.
  - There are no resources allowed, apart from a hand-written piece of paper (A4).
  - Answers should be given in English, but German is also accepted.

Assessment

- Assignment Sheets:
  - An assignment sheet is usually made available on Monday on the module webpage.
  - Solutions have to be handed in in the following week before the lecture on Monday.
  - You can hand in your solutions by putting them in the right folder in front of room 03.09.052.
  - Solutions have to be given in English.
  - Solutions will be discussed in the subsequent tutorial on Tuesday.
  - The first one will be out on Monday, 14 April.

Assessment

- Assignment Sheets:
  - An assignment sheet is usually made available on Monday on the module webpage.
  - Solutions have to be handed in in the following week before the lecture on Monday.
  - You can hand in your solutions by putting them in the right folder in front of room 03.09.052.
  - Solutions have to be given in English.
  - Solutions will be discussed in the subsequent tutorial on Tuesday.
  - The first one will be out on Monday, 14 April.

Assessment

- Assignment Sheets:
  - An assignment sheet is usually made available on Monday on the module webpage.
  - Solutions have to be handed in in the following week before the lecture on Monday.
  - You can hand in your solutions by putting them in the right folder in front of room 03.09.052.
  - Solutions have to be given in English.
  - Solutions will be discussed in the subsequent tutorial on Tuesday.
  - The first one will be out on Monday, 14 April.

Assessment

- Assignment Sheets:
  - An assignment sheet is usually made available on Monday on the module webpage.
  - Solutions have to be handed in in the following week before the lecture on Monday.
  - You can hand in your solutions by putting them in the right folder in front of room 03.09.052.
  - Solutions have to be given in English.
  - Solutions will be discussed in the subsequent tutorial on Tuesday.
  - The first one will be out on Monday, 14 April.

Assessment

- Assignment Sheets:
  - An assignment sheet is usually made available on Monday on the module webpage.
  - Solutions have to be handed in in the following week before the lecture on Monday.
  - You can hand in your solutions by putting them in the right folder in front of room 03.09.052.
  - Solutions have to be given in English.
  - Solutions will be discussed in the subsequent tutorial on Tuesday.
  - The first one will be out on Monday, 14 April.

Assessment

- Assignment Sheets:
  - An assignment sheet is usually made available on Monday on the module webpage.
  - Solutions have to be handed in in the following week before the lecture on Monday.
  - You can hand in your solutions by putting them in the right folder in front of room 03.09.052.
  - Solutions have to be given in English.
  - Solutions will be discussed in the subsequent tutorial on Tuesday.
  - The first one will be out on Monday, 14 April.

Assessment

- Assignment Sheets:
  - An assignment sheet is usually made available on Monday on the module webpage.
  - Solutions have to be handed in in the following week before the lecture on Monday.
  - You can hand in your solutions by putting them in the right folder in front of room 03.09.052.
  - Solutions have to be given in English.
  - Solutions will be discussed in the subsequent tutorial on Tuesday.
  - The first one will be out on Monday, 14 April.

1 Contents

Part 1: Linear Programming

Part 2: Approximation Algorithms

2 Literatur

- V. Chvatal:
  Linear Programming,
  Freeman, 1983
- R. Seidel:
  Skript Optimierung, 1996
- D. Bertsimas and J.N. Tsitsiklis:
  Introduction to Linear Optimization,
  Athena Scientific, 1997
- Vijay V. Vazirani:
  Approximation Algorithms,
  Springer 2001

- David P. Williamson and David B. Shmoys:
  The Design of Approximation Algorithms,
  Cambridge University Press 2011
- G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi:
  Complexity and Approximation,
  Springer, 1999

Part II

Linear Programming

Brewery Problem

$\bar{U}$ **Brewery brews ale and beer.**

- Production limited by supply of corn, hops and barley malt
- Recipes for ale and beer require different amounts of resources

	Corn (kg)	*Hops (kg)*	*Malt (kg)*	*Profit (€)*
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

Brewery Problem

$\bar{U}$ **Brewery brews ale and beer.**

- Production limited by supply of corn, hops and barley malt
- Recipes for ale and beer require different amounts of resources

	Corn (kg)	Hops (kg)	Malt (kg)	Profit (€)
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

Brewery Problem

$\bar{U}$ **Brewery brews ale and beer.**

- Production limited by supply of corn, hops and barley malt
- Recipes for ale and beer require different amounts of resources

	Corn (kg)	*Hops (kg)*	*Malt (kg)*	*Profit (€)*
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

Brewery Problem

	Corn (kg)	*Hops (kg)*	*Malt (kg)*	*Profit (€)*
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

How can brewer maximize profits?

- only brew ale: 34 barrels of ale
- only brew beer: 32 barrels of beer
- 7.5 barrels ale, 29.5 barrels beer
- 12 barrels ale, 28 barrels beer

Brewery Problem

	Corn (kg)	*Hops (kg)*	*Malt (kg)*	*Profit (€)*
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

How can brewer maximize profits?

- only brew ale: 34 barrels of ale $\Longrightarrow$ 442 €
- only brew beer: 32 barrels of beer $\Longrightarrow$ 736 €
- 7.5 barrels ale, 29.5 barrels beer $\Longrightarrow$ 776 €
- 12 barrels ale, 28 barrels beer $\Longrightarrow$ 800 €

Brewery Problem

	Corn (kg)	*Hops (kg)*	*Malt (kg)*	*Profit (€)*
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

How can brewer maximize profits?

- only brew ale: 34 barrels of ale $\Longrightarrow$ 442 €
- only brew beer: 32 barrels of beer $\Longrightarrow$ 736 €
- 7.5 barrels ale, 29.5 barrels beer $\Longrightarrow$ 776 €
- 12 barrels ale, 28 barrels beer $\Longrightarrow$ 800 €

Brewery Problem

	Corn (kg)	*Hops (kg)*	*Malt (kg)*	*Profit (€)*
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

How can brewer maximize profits?

- only brew ale: 34 barrels of ale $\Longrightarrow$ 442 €
- only brew beer: 32 barrels of beer $\Longrightarrow$ 736 €
- 7.5 barrels ale, 29.5 barrels beer $\Longrightarrow$ 776 €
- 12 barrels ale, 28 barrels beer $\Longrightarrow$ 800 €

Brewery Problem

	Corn (kg)	*Hops (kg)*	*Malt (kg)*	*Profit (€)*
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

How can brewer maximize profits?

- only brew ale: 34 barrels of ale $\Longrightarrow$ 442 €
- only brew beer: 32 barrels of beer $\Longrightarrow$ 736 €
- 7.5 barrels ale, 29.5 barrels beer $\Longrightarrow$ 776 €
- 12 barrels ale, 28 barrels beer $\Longrightarrow$ 800 €

Brewery Problem

	Corn (kg)	*Hops (kg)*	*Malt (kg)*	*Profit (€)*
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

How can brewer maximize profits?

- only brew ale: 34 barrels of ale $\Longrightarrow$ 442 €
- only brew beer: 32 barrels of beer $\Longrightarrow$ 736 €
- 7.5 barrels ale, 29.5 barrels beer $\Longrightarrow$ 776 €
- 12 barrels ale, 28 barrels beer $\Longrightarrow$ 800 €

Brewery Problem

	Corn (kg)	*Hops (kg)*	*Malt (kg)*	*Profit (€)*
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

How can brewer maximize profits?

- only brew ale: 34 barrels of ale $\Longrightarrow$ 442 €
- only brew beer: 32 barrels of beer $\Longrightarrow$ 736 €
- 7.5 barrels ale, 29.5 barrels beer $\Longrightarrow$ 776 €
- 12 barrels ale, 28 barrels beer $\Longrightarrow$ 800 €

Brewery Problem

	Corn (kg)	*Hops (kg)*	*Malt (kg)*	*Profit (€)*
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

How can brewer maximize profits?

- only brew ale: 34 barrels of ale $\Longrightarrow$ 442 €
- only brew beer: 32 barrels of beer $\Longrightarrow$ 736 €
- 7.5 barrels ale, 29.5 barrels beer $\Longrightarrow$ 776 €
- 12 barrels ale, 28 barrels beer $\Longrightarrow$ 800 €

Brewery Problem

	Corn (kg)	*Hops (kg)*	*Malt (kg)*	*Profit (€)*
ale (barrel)	5	4	35	13
beer (barrel)	15	4	20	23
supply	480	160	1190	

How can brewer maximize profits?

- only brew ale: 34 barrels of ale $\Longrightarrow$ 442 €
- only brew beer: 32 barrels of beer $\Longrightarrow$ 736 €
- 7.5 barrels ale, 29.5 barrels beer $\Longrightarrow$ 776 €
- 12 barrels ale, 28 barrels beer $\Longrightarrow$ 800 €

Brewery Problem

Linear Program

- Introduce variables a and b that define how much ale and beer to produce.
- Choose the variables in such a way that the objective function (profit) is maximized.
- Make sure that no constraints (due to limited supply) are violated.

$$
\begin{array}{rrcrl}
\text{max} & 13a & + & 23b & \\
\text{s.t.} & 5a & + & 15b & \le 480 \\
 & 4a & + & 4b & \le 160 \\
 & 35a & + & 20b & \le 1190 \\
 & & & a,b & \ge 0
\end{array}
$$

Brewery Problem

Linear Program

- Introduce variables a and b that define how much ale and beer to produce.
- Choose the variables in such a way that the objective function (profit) is maximized.
- Make sure that no constraints (due to limited supply) are violated.

$$\begin{array}{lrcrl} \max & 13a & + & 23b & \\ \text{s.t.} & 5a & + & 15b & \le 480 \\ & 4a & + & 4b & \le 160 \\ & 35a & + & 20b & \le 1190 \\ & & & a,b & \ge 0 \end{array}$$

Brewery Problem

Linear Program

- Introduce variables a and b that define how much ale and beer to produce.
- Choose the variables in such a way that the objective function (profit) is maximized.
- Make sure that no constraints (due to limited supply) are violated.

$$
\begin{array}{lrcrl}
\text{max} & 13a & + & 23b & \\
\text{s.t.} & 5a & + & 15b & \leq 480 \\
 & 4a & + & 4b & \leq 160 \\
 & 35a & + & 20b & \leq 1190 \\
 & & & a,b & \geq 0
\end{array}
$$

Brewery Problem

Linear Program

- Introduce variables a and b that define how much ale and beer to produce.
- Choose the variables in such a way that the objective function (profit) is maximized.
- Make sure that no constraints (due to limited supply) are violated.

$$
\begin{array}{lrcrl}
\max & 13a & + & 23b & \\
\text{s.t.} & 5a & + & 15b & \leq 480 \\
 & 4a & + & 4b & \leq 160 \\
 & 35a & + & 20b & \leq 1190 \\
 & & & a, b & \geq 0
\end{array}
$$

Brewery Problem

Linear Program

- Introduce variables a and b that define how much ale and beer to produce.
- Choose the variables in such a way that the objective function (profit) is maximized.
- Make sure that no constraints (due to limited supply) are violated.

$$
\begin{array}{lrcrl}
\text{max} & 13a & + & 23b & \\
\text{s.t.} & 5a & + & 15b & \leq 480 \\
 & 4a & + & 4b & \leq 160 \\
 & 35a & + & 20b & \leq 1190 \\
 & & & a,b & \geq 0
\end{array}
$$

Standard Form LPs

LP in standard form:

- input: numbers a_{ij}, c_j, b_i
- output: numbers x_j
- n = #decision variables, m = #constraints
- maximize linear objective function subject to linear inequalities

$$\begin{array}{llll} \max & \sum_{j=1}^{n} c_j x_j & & \\ \text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j & = & b_i \quad 1 \le i \le m \\ & x_j & \ge & 0 \quad 1 \le j \le n \end{array}$$

$$\begin{array}{llll} \max & c^T x & & \\ \text{s.t.} & Ax & = & b \\ & x & \ge & 0 \end{array}$$

Standard Form LPs

LP in standard form:

- input: numbers a_{ij}, c_j, b_i
- output: numbers x_j
- n = #decision variables, m = #constraints
- maximize linear objective function subject to linear inequalities

$$\begin{array}{lrcll} \max & \sum_{j=1}^{n} c_j x_j & & & \\ \text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j & \le & b_i & 1 \le i \le m \\ & x_j & \ge & 0 & 1 \le j \le n \end{array}$$

$$\begin{array}{lrcl} \max & c^T x & & \\ \text{s.t.} & Ax & \le & b \\ & x & \ge & 0 \end{array}$$

Standard Form LPs

LP in standard form:

- input: numbers a_{ij}, c_j, b_i
- output: numbers x_j
- n = #decision variables, m = #constraints
- maximize linear objective function subject to linear inequalities

$$
\begin{aligned}
\max \quad & \sum_{j=1}^{n} c_j x_j \\
\text{s.t.} \quad & \sum_{j=1}^{n} a_{ij} x_j = b_i \quad 1 \le i \le m \\
& x_j \ge 0 \quad 1 \le j \le n
\end{aligned}
$$

$$
\begin{aligned}
\max \quad & c^T x \\
\text{s.t.} \quad & Ax = b \\
& x \ge 0
\end{aligned}
$$

Standard Form LPs

LP in standard form:

- input: numbers a_{ij}, c_j, b_i
- output: numbers x_j
- n = #decision variables, m = #constraints
- maximize linear objective function subject to linear inequalities

$$\max \sum_{j=1}^{n} c_j x_j$$
$$\text{s.t.} \sum_{j=1}^{n} a_{ij} x_j = b_i \quad 1 \le i \le m$$
$$x_j \ge 0 \quad 1 \le j \le n$$

$$\max \; c^T x \quad \text{s.t.} \; Ax = b, \; x \ge 0$$

Standard Form LPs

LP in standard form:

- input: numbers a_{ij}, c_j, b_i
- output: numbers x_j
- n = #decision variables, m = #constraints
- maximize linear objective function subject to linear inequalities

$$\begin{array}{llcll} \max & \sum_{j=1}^{n} c_j x_j & & & \\ \text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j & = & b_i & 1 \le i \le m \\ & x_j & \ge & 0 & 1 \le j \le n \end{array}$$

$$\begin{array}{llcl} \max & c^t x & & \\ \text{s.t.} & Ax & = & b \\ & x & \ge & 0 \end{array}$$

Standard Form LPs

LP in standard form:

- input: numbers a_{ij}, c_j, b_i
- output: numbers x_j
- $n =$ #decision variables, $m =$ #constraints
- maximize linear objective function subject to linear inequalities

$$
\begin{array}{lrcll}
\text{max} & \sum_{j=1}^{n} c_j x_j & & & \\
\text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j & = & b_i & 1 \le i \le m \\
 & x_j & \ge & 0 & 1 \le j \le n
\end{array}
$$

$$
\begin{array}{lrcl}
\text{max} & c^t x & & \\
\text{s.t.} & Ax & = & b \\
 & x & \ge & 0
\end{array}
$$

Standard Form LPs

LP in standard form:

- input: numbers a_{ij}, c_j, b_i
- output: numbers x_j
- n = #decision variables, m = #constraints
- maximize linear objective function subject to linear inequalities

$$\begin{array}{llcll} \text{max} & \sum_{j=1}^{n} c_j x_j & & & \\ \text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j & = & b_i & 1 \le i \le m \\ & x_j & \ge & 0 & 1 \le j \le n \end{array}$$

$$\begin{array}{lrcl} \text{max} & c^t x & & \\ \text{s.t.} & Ax & = & b \\ & x & \ge & 0 \end{array}$$

Standard Form LPs

Original LP

$$
\begin{array}{lrcrl}
\text{max} & 13a & + & 23b & \\
\text{s.t.} & 5a & + & 15b & \le 480 \\
 & 4a & + & 4b & \le 160 \\
 & 35a & + & 20b & \le 1190 \\
 & & & a, b & \ge 0
\end{array}
$$

Standard Form

Add a slack variable to every constraint.

$$
\begin{array}{lrcrcccccccl}
\text{max} & 13a & + & 23b & & & & & & & & \\
\text{s.t.} & 5a & + & 15b & + & s_c & & & & & & = 480 \\
 & 4a & + & 4b & & & + & s_h & & & & = 160 \\
 & 35a & + & 20b & & & & & + & s_m & & = 1190 \\
 & a & , & b & , & s_c & , & s_h & , & s_m & & \ge 0
\end{array}
$$

Standard Form LPs

Original LP

$$\begin{array}{lrcrl} \max & 13a & + & 23b & \\ \text{s.t.} & 5a & + & 15b & \le 480 \\ & 4a & + & 4b & \le 160 \\ & 35a & + & 20b & \le 1190 \\ & & & a,b & \ge 0 \end{array}$$

Standard Form

Add a slack variable to every constraint.

$$\begin{array}{lrcrcrcrcrl} \max & 13a & + & 23b & & & & & & & \\ \text{s.t.} & 5a & + & 15b & + & s_c & & & & & = 480 \\ & 4a & + & 4b & & & + & s_h & & & = 160 \\ & 35a & + & 20b & & & & & + & s_m & = 1190 \\ & a & , & b & , & s_c & , & s_h & , & s_m & \ge 0 \end{array}$$

Standard Form LPs

There are different standard forms:

standard form

$$\begin{array}{lrcl} \max & c^t x & & \\ \text{s.t.} & Ax & = & b \\ & x & \geq & 0 \end{array}$$

$$\begin{array}{lrcl} \min & c^t x & & \\ \text{s.t.} & Ax & = & b \\ & x & \geq & 0 \end{array}$$

standard maximization form

$$\begin{array}{lrcl} \max & c^t x & & \\ \text{s.t.} & Ax & \leq & b \\ & x & \geq & 0 \end{array}$$

standard minimization form

$$\begin{array}{lrcl} \min & c^t x & & \\ \text{s.t.} & Ax & \geq & b \\ & x & \geq & 0 \end{array}$$

Standard Form LPs

There are different standard forms:

standard form

$$\begin{array}{lrcl} \max & c^t x & & \\ \text{s.t.} & Ax & = & b \\ & x & \geq & 0 \end{array}$$

$$\begin{array}{lrcl} \min & c^t x & & \\ \text{s.t.} & Ax & = & b \\ & x & \geq & 0 \end{array}$$

standard maximization form

$$\begin{array}{lrcl} \max & c^t x & & \\ \text{s.t.} & Ax & \leq & b \\ & x & \geq & 0 \end{array}$$

standard minimization form

$$\begin{array}{lrcl} \min & c^t x & & \\ \text{s.t.} & Ax & \geq & b \\ & x & \geq & 0 \end{array}$$

Standard Form LPs

There are different standard forms:

standard form

$$\begin{array}{llcl} \max & c^t x & & \\ \text{s.t.} & Ax & = & b \\ & x & \geq & 0 \end{array}$$

$$\begin{array}{llcl} \min & c^t x & & \\ \text{s.t.} & Ax & = & b \\ & x & \geq & 0 \end{array}$$

standard maximization form

$$\begin{array}{llcl} \max & c^t x & & \\ \text{s.t.} & Ax & \leq & b \\ & x & \geq & 0 \end{array}$$

standard minimization form

$$\begin{array}{llcl} \min & c^t x & & \\ \text{s.t.} & Ax & \geq & b \\ & x & \geq & 0 \end{array}$$

Standard Form LPs

There are different standard forms:

standard form

$$\begin{array}{lrcl} \max & c^t x & & \\ \text{s.t.} & Ax & = & b \\ & x & \geq & 0 \end{array}$$

$$\begin{array}{lrcl} \min & c^t x & & \\ \text{s.t.} & Ax & = & b \\ & x & \geq & 0 \end{array}$$

standard maximization form

$$\begin{array}{lrcl} \max & c^t x & & \\ \text{s.t.} & Ax & \leq & b \\ & x & \geq & 0 \end{array}$$

standard minimization form

$$\begin{array}{lrcl} \min & c^t x & & \\ \text{s.t.} & Ax & \geq & b \\ & x & \geq & 0 \end{array}$$

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

less or equal to equality:

greater or equal to equality:

min to max:

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

- **less or equal to equality:**

$$a - 3b + 5c \le 12 \implies \begin{aligned} a - 3b + 5c + s &= 12 \\ s &\ge 0 \end{aligned}$$

- **greater or equal to equality:**

$$a - 3b + 5c \ge 12 \implies \begin{aligned} a - 3b + 5c - s &= 12 \\ s &\ge 0 \end{aligned}$$

- **min to max:**

$$\min a - 3b + 5c \implies \max -a + 3b - 5c$$

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

- **less or equal to equality:**

$$a - 3b + 5c \leq 12 \implies \begin{aligned} a - 3b + 5c + s &= 12 \\ s &\geq 0 \end{aligned}$$

- **greater or equal to equality:**

- **min to max:**

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

- **less or equal to equality:**

$$a - 3b + 5c \leq 12 \implies \begin{aligned} a - 3b + 5c + s &= 12 \\ s &\geq 0 \end{aligned}$$

- **greater or equal to equality:**

$$a - 3b + 5c \geq 12 \implies \begin{aligned} a - 3b + 5c - s &= 12 \\ s &\geq 0 \end{aligned}$$

- **min to max:**

$$\min a - 3b + 5c \implies \max -a + 3b - 5c$$

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

- **less or equal to equality:**

$$a - 3b + 5c \leq 12 \implies \begin{aligned} a - 3b + 5c + s &= 12 \\ s &\geq 0 \end{aligned}$$

- **greater or equal to equality:**

$$a - 3b + 5c \geq 12 \implies \begin{aligned} a - 3b + 5c - s &= 12 \\ s &\geq 0 \end{aligned}$$

- **min to max:**

$$\min a - 3b + 5c \implies \max -a + 3b - 5c$$

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

- **less or equal to equality:**

$$a - 3b + 5c \leq 12 \implies \begin{aligned} a - 3b + 5c + s &= 12 \\ s &\geq 0 \end{aligned}$$

- **greater or equal to equality:**

$$a - 3b + 5c \geq 12 \implies \begin{aligned} a - 3b + 5c - s &= 12 \\ s &\geq 0 \end{aligned}$$

- **min to max:**

$$\min a - 3b + 5c \implies \max -a + 3b - 5c$$

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

- **less or equal to equality:**

$$a - 3b + 5c \leq 12 \implies \begin{aligned} a - 3b + 5c + s &= 12 \\ s &\geq 0 \end{aligned}$$

- **greater or equal to equality:**

$$a - 3b + 5c \geq 12 \implies \begin{aligned} a - 3b + 5c - s &= 12 \\ s &\geq 0 \end{aligned}$$

- **min to max:**

$$\min a - 3b + 5c \implies \max -a + 3b - 5c$$

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

- **equality to less or equal:**

$$a - 3b + 5c = 12 \implies \begin{array}{r} a - 3b + 5c \le 12 \\ -a + 3b - 5c \le -12 \end{array}$$

- **equality to greater or equal:**

$$a - 3b + 5c = 12 \implies \begin{array}{r} a - 3b + 5c \ge 12 \\ -a + 3b - 5c \ge -12 \end{array}$$

- **unrestricted to nonnegative:**

$$x \text{ unrestricted} \implies x = x^+ - x^-,\ x^+ \ge 0,\ x^- \ge 0$$

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

- **equality to less or equal:**

$$a - 3b + 5c = 12 \implies \begin{aligned} a - 3b + 5c &\le 12 \\ -a + 3b - 5c &\le -12 \end{aligned}$$

- **equality to greater or equal:**

- **unrestricted to nonnegative:**

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

- **equality to less or equal:**

$$a - 3b + 5c = 12 \implies \begin{array}{rcl} a - 3b + 5c & \leq & 12 \\ -a + 3b - 5c & \leq & -12 \end{array}$$

- **equality to greater or equal:**

$$a - 3b + 5c = 12 \implies \begin{array}{rcl} a - 3b + 5c & \geq & 12 \\ -a + 3b - 5c & \geq & -12 \end{array}$$

- **unrestricted to nonnegative:**

$$x \text{ unrestricted} \implies x = x^+ - x^-,\ x^+ \geq 0,\ x^- \geq 0$$

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

- **equality to less or equal:**

$$a - 3b + 5c = 12 \implies \begin{array}{r} a - 3b + 5c \le 12 \\ -a + 3b - 5c \le -12 \end{array}$$

- **equality to greater or equal:**

$$a - 3b + 5c = 12 \implies \begin{array}{r} a - 3b + 5c \ge 12 \\ -a + 3b - 5c \ge -12 \end{array}$$

- **unrestricted to nonnegative:**

$$x \text{ unrestricted} \implies x = x^+ - x^-, x^+ \ge 0, x^- \ge 0$$

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

- **equality to less or equal:**

$$a - 3b + 5c = 12 \implies \begin{aligned} a - 3b + 5c &\leq 12 \\ -a + 3b - 5c &\leq -12 \end{aligned}$$

- **equality to greater or equal:**

$$a - 3b + 5c = 12 \implies \begin{aligned} a - 3b + 5c &\geq 12 \\ -a + 3b - 5c &\geq -12 \end{aligned}$$

- **unrestricted to nonnegative:**

$$x \text{ unrestricted} \implies x = x^+ - x^-, x^+ \geq 0, x^- \geq 0$$

Standard Form LPs

It is easy to transform variants of LPs into (any) standard form:

- **equality to less or equal:**

$$a - 3b + 5c = 12 \implies \begin{array}{r} a - 3b + 5c \leq 12 \\ -a + 3b - 5c \leq -12 \end{array}$$

- **equality to greater or equal:**

$$a - 3b + 5c = 12 \implies \begin{array}{r} a - 3b + 5c \geq 12 \\ -a + 3b - 5c \geq -12 \end{array}$$

- **unrestricted to nonnegative:**

$$x \text{ unrestricted} \implies x = x^+ - x^-,\ x^+ \geq 0,\ x^- \geq 0$$

Standard Form LPs

Observations:

- a linear program does not contain x^2, $\cos(x)$, etc.
- transformations between standard forms can be done efficiently and only change the size of the LP by a small constant factor
- for the standard minimization or maximization LPs we could include the nonnegativity constraints into the set of ordinary constraints; this is of course not possible for the standard form

Standard Form LPs

Observations:

- a linear program does not contain x^2, $\cos(x)$, etc.
- transformations between standard forms can be done efficiently and only change the size of the LP by a small constant factor
- for the standard minimization or maximization LPs we could include the nonnegativity constraints into the set of ordinary constraints; this is of course not possible for the standard form

Standard Form LPs

Observations:

- a linear program does not contain x^2, $\cos(x)$, etc.
- transformations between standard forms can be done efficiently and only change the size of the LP by a small constant factor
- for the standard minimization or maximization LPs we could include the nonnegativity constraints into the set of ordinary constraints; this is of course not possible for the standard form

Fundamental Questions

Definition 1 (Linear Programming Problem (LP))

Let $A \in \mathbb{Q}^{m\times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$ s.t. $Ax = b$, $x \geq 0$, $c^t x \geq \alpha$?

Questions:

- Is LP in NP?
- Is LP in co-NP?
- Is LP in P?

Input size:

- n number of variables, m constraints, L number of bits to encode the input

Fundamental Questions

Definition 1 (Linear Programming Problem (LP))

Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$ s.t. $Ax = b$, $x \geq 0$, $c^t x \geq \alpha$?

Questions:

- Is LP in NP?
- Is LP in co-NP?
- Is LP in P?

Input size:

- n number of variables, m constraints, L number of bits to encode the input

Fundamental Questions

Definition 1 (Linear Programming Problem (LP))

Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$ s.t. $Ax = b$, $x \geq 0$, $c^t x \geq \alpha$?

Questions:

- Is LP in NP?
- Is LP in co-NP?
- Is LP in P?

Input size:

- n number of variables, m constraints, L number of bits to encode the input

Fundamental Questions

Definition 1 (Linear Programming Problem (LP))

Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$ s.t. $Ax = b$, $x \geq 0$, $c^t x \geq \alpha$?

Questions:

- Is LP in NP?
- Is LP in co-NP?
- Is LP in P?

Input size:

- n number of variables, m constraints, L number of bits to encode the input

Fundamental Questions

Definition 1 (Linear Programming Problem (LP))

Let $A \in \mathbb{Q}^{m\times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$ s.t. $Ax = b$, $x \geq 0$, $c^t x \geq \alpha$?

Questions:

- Is LP in NP?
- Is LP in co-NP?
- Is LP in P?

Input size:

- n number of variables, m constraints, L number of bits to encode the input

Fundamental Questions

Definition 1 (Linear Programming Problem (LP))

Let $A \in \mathbb{Q}^{m\times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$ s.t. $Ax = b$, $x \geq 0$, $c^t x \geq \alpha$?

Questions:

- Is LP in NP?
- Is LP in co-NP?
- Is LP in P?

Input size:

- n number of variables, m constraints, L number of bits to encode the input

Fundamental Questions

Definition 1 (Linear Programming Problem (LP))

Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$ s.t. $Ax = b$, $x \geq 0$, $c^t x \geq \alpha$?

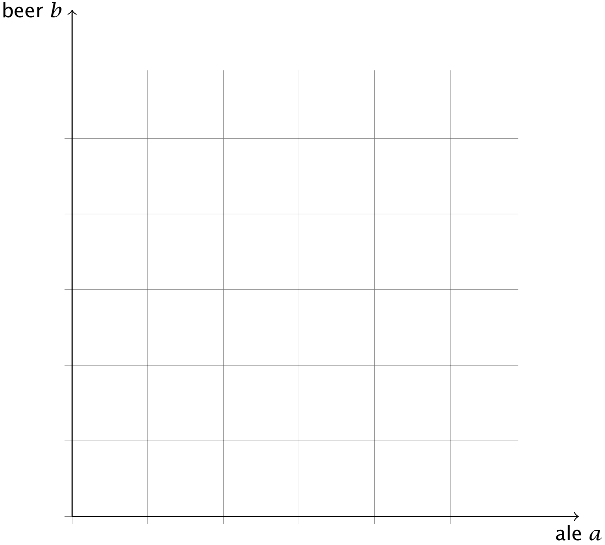
Questions:

- Is LP in NP?
- Is LP in co-NP?
- Is LP in P?

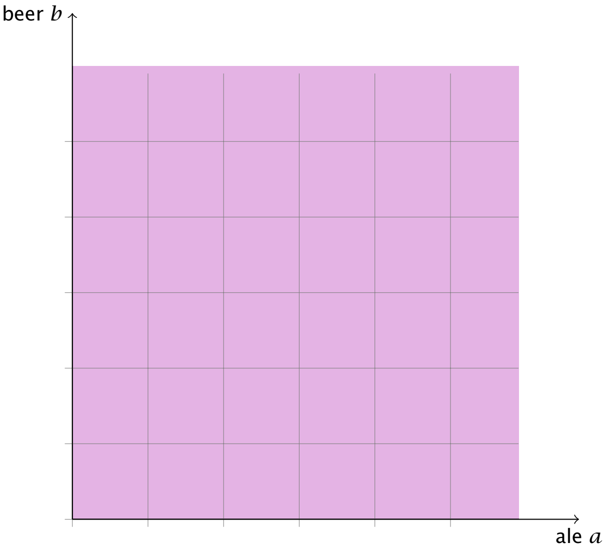
Input size:

- n number of variables, m constraints, L number of bits to encode the input

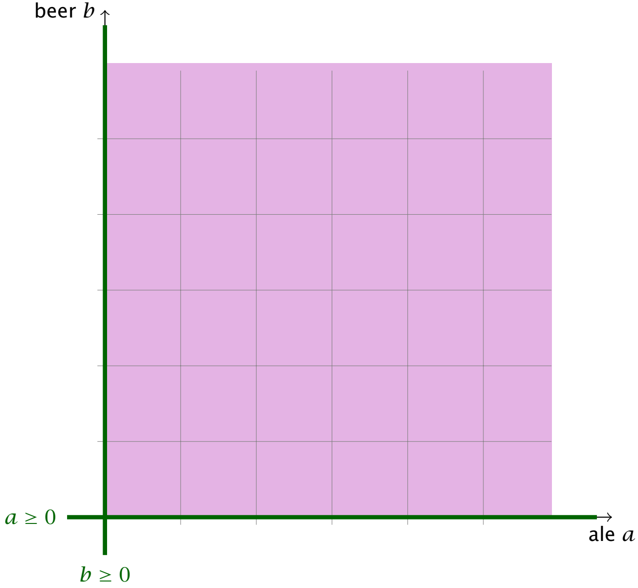
Geometry of Linear Programming



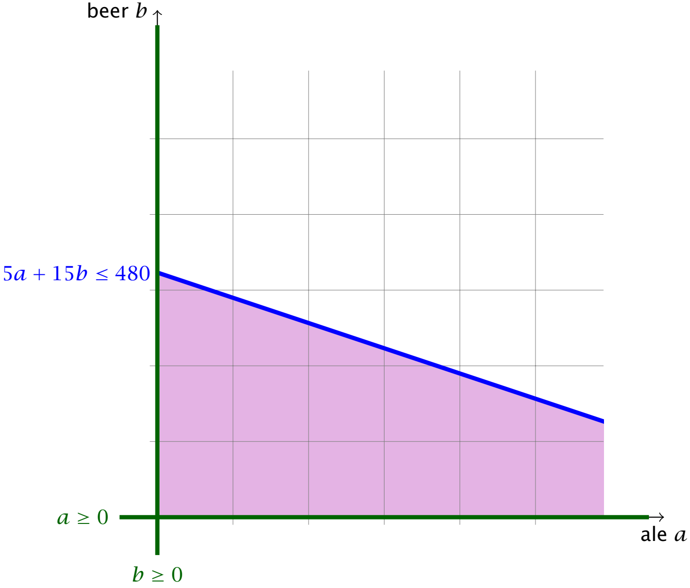
Geometry of Linear Programming



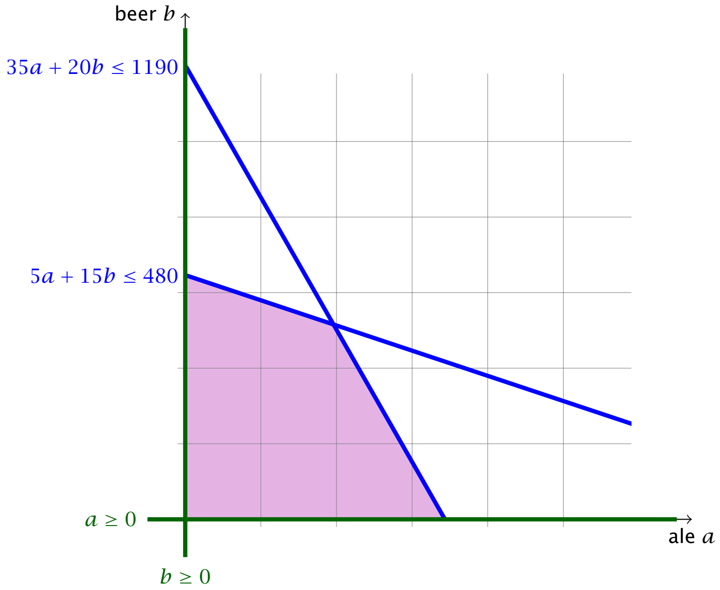
Geometry of Linear Programming



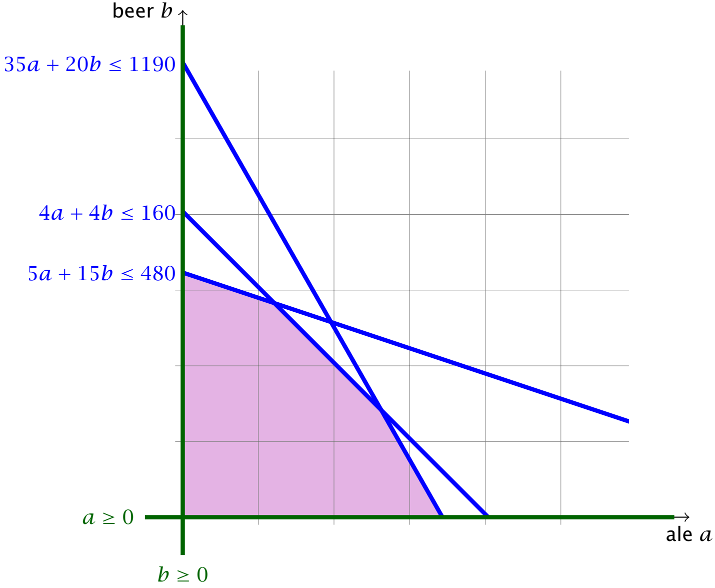
Geometry of Linear Programming



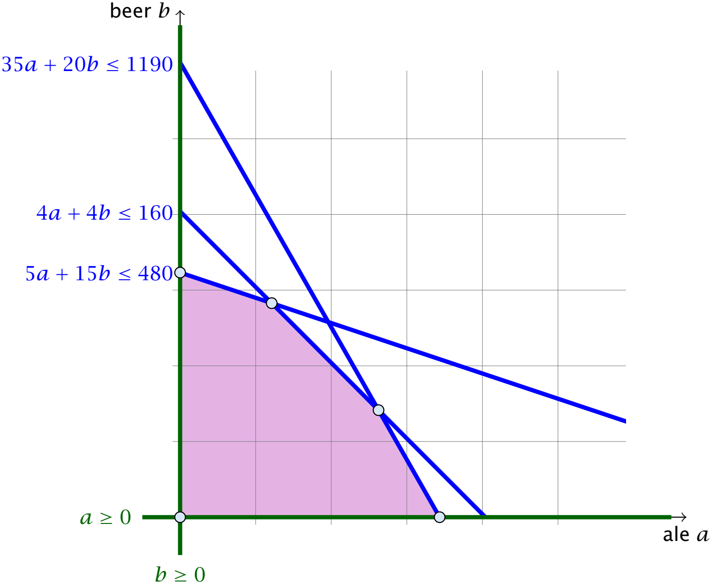
Geometry of Linear Programming



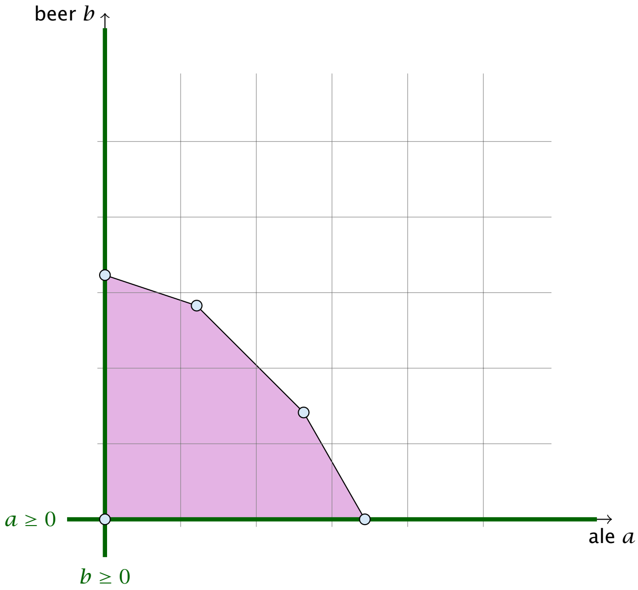
Geometry of Linear Programming



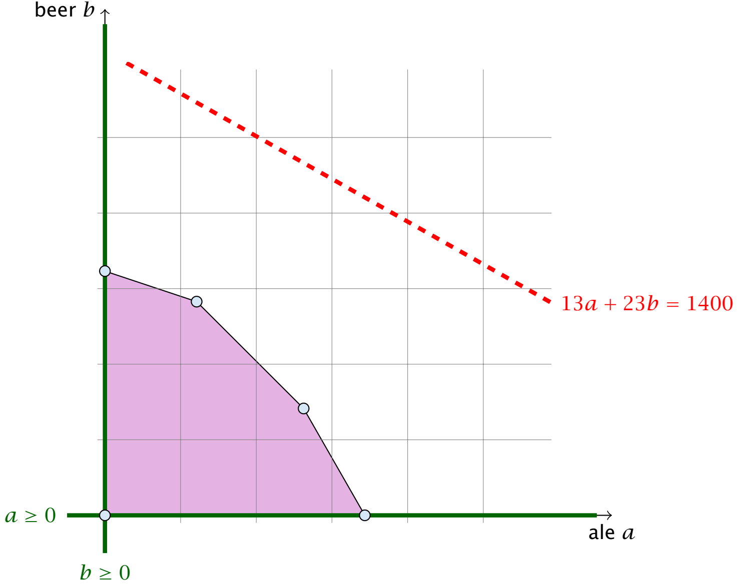
Geometry of Linear Programming



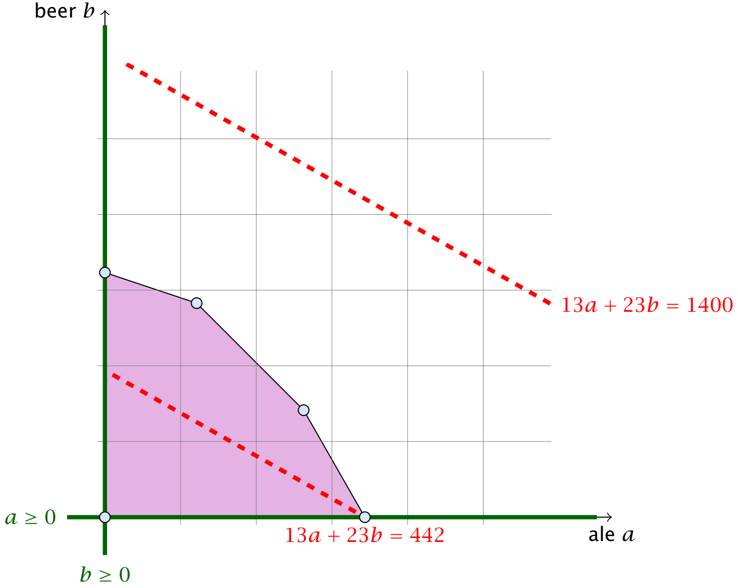
Geometry of Linear Programming



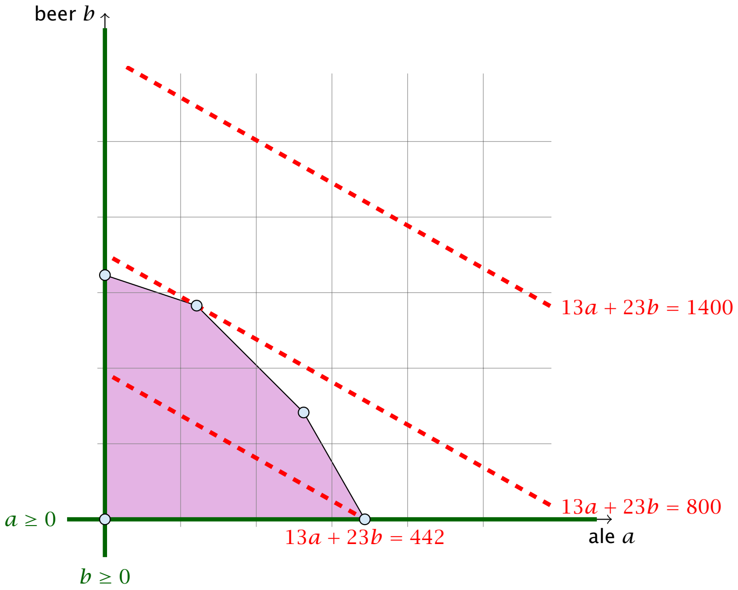
Geometry of Linear Programming



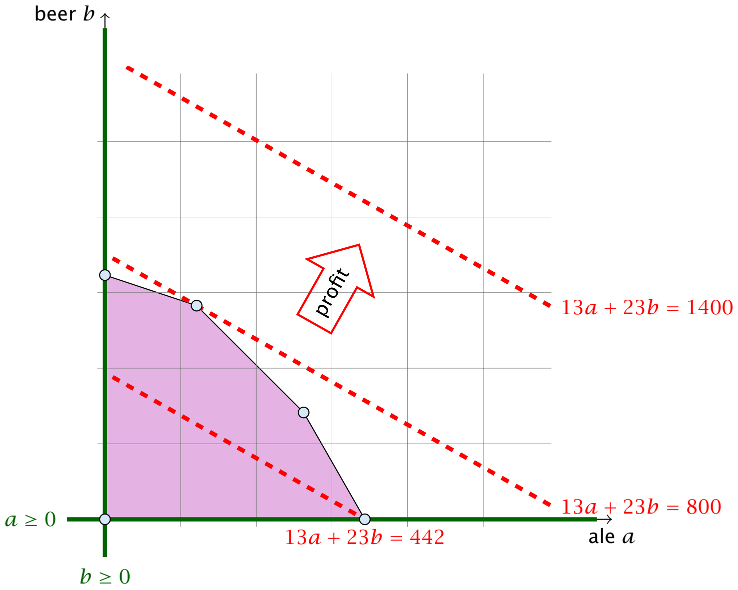
Geometry of Linear Programming



Geometry of Linear Programming

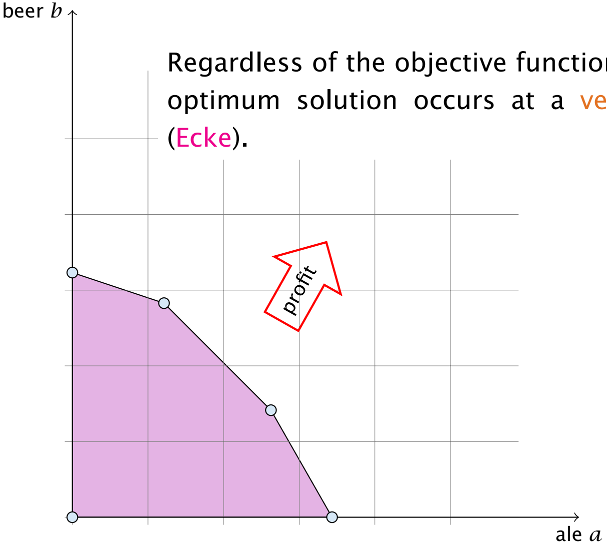


Geometry of Linear Programming



Geometry of Linear Programming

Regardless of the objective function an optimum solution occurs at a vertex (Ecke).



Convex Sets

A set $S \subseteq \mathbb{R}$ is convex if for all $x, y \in S$ also $\lambda x + (1-\lambda)y \in S$ for all $0 \le \lambda \le 1$.

A point in $x \in S$ that can't be written as a convex combination of two other points in the set is called a vertex.

Convex Sets

A set $S \subseteq \mathbb{R}$ is convex if for all $x, y \in S$ also $\lambda x + (1-\lambda)y \in S$ for all $0 \leq \lambda \leq 1$.

A point in $x \in S$ that can't be written as a convex combination of two other points in the set is called a vertex.

Definitions

Let for a Linear Program in standard form
$P = \{x \mid Ax = b, x \geq 0\}$.

Definitions

Let for a Linear Program in standard form
$P = \{x \mid Ax = b, x \geq 0\}$.

- P is called the feasible region (Lösungsraum) of the LP.

Definitions

Let for a Linear Program in standard form
$P = \{x \mid Ax = b, x \geq 0\}$.

- P is called the feasible region (Lösungsraum) of the LP.
- A point $x \in P$ is called a feasible point (gültige Lösung).
- If $P \neq \emptyset$ then the LP is called feasible (erfüllbar). Otherwise, it is called infeasible (unerfüllbar).
- An LP is bounded (beschränkt) if it is feasible and
  - $c^T x < \infty$ for all $x \in P$ (for maximization problems)
  - $c^T x > -\infty$ for all $x \in P$ (for minimization problems)

Definitions

Let for a Linear Program in standard form
$P = \{x \mid Ax = b, x \geq 0\}$.

- P is called the feasible region (Lösungsraum) of the LP.
- A point $x \in P$ is called a feasible point (gültige Lösung).
- If $P \neq \emptyset$ then the LP is called feasible (erfüllbar). Otherwise, it is called infeasible (unerfüllbar).
- An LP is bounded (beschränkt) if it is feasible and
  - $c^t x < \infty$ for all $x \in P$ (for maximization problems)
  - $c^t x > -\infty$ for all $x \in P$ (for minimization problems)

Definitions

Let for a Linear Program in standard form $P = \{x \mid Ax = b, x \geq 0\}$.

- P is called the feasible region (Lösungsraum) of the LP.
- A point $x \in P$ is called a feasible point (gültige Lösung).
- If $P \neq \emptyset$ then the LP is called feasible (erfüllbar). Otherwise, it is called infeasible (unerfüllbar).
- An LP is bounded (beschränkt) if it is feasible and
  - $c^T x < \infty$ for all $x \in P$ (for maximization problems)
  - $c^T x > -\infty$ for all $x \in P$ (for minimization problems)

Definitions

Let for a Linear Program in standard form
$P = \{x \mid Ax = b, x \geq 0\}$.

- P is called the feasible region (Lösungsraum) of the LP.
- A point $x \in P$ is called a feasible point (gültige Lösung).
- If $P \neq \emptyset$ then the LP is called feasible (erfüllbar). Otherwise, it is called infeasible (unerfüllbar).
- An LP is bounded (beschränkt) if it is feasible and
  - $c^t x < \infty$ for all $x \in P$ (for maximization problems)
  - $c^t x > -\infty$ for all $x \in P$ (for minimization problems)

Definitions

Let for a Linear Program in standard form
$P = \{x \mid Ax = b, x \geq 0\}$.

- P is called the feasible region (Lösungsraum) of the LP.
- A point $x \in P$ is called a feasible point (gültige Lösung).
- If $P \neq \emptyset$ then the LP is called feasible (erfüllbar). Otherwise, it is called infeasible (unerfüllbar).
- An LP is bounded (beschränkt) if it is feasible and
  - $c^t x < \infty$ for all $x \in P$ (for maximization problems)
  - $c^t x > -\infty$ for all $x \in P$ (for minimization problems)

Definitions

Let for a Linear Program in standard form
$P = \{x \mid Ax = b, x \geq 0\}$.

- P is called the feasible region (Lösungsraum) of the LP.
- A point $x \in P$ is called a feasible point (gültige Lösung).
- If $P \neq \emptyset$ then the LP is called feasible (erfüllbar). Otherwise, it is called infeasible (unerfüllbar).
- An LP is bounded (beschränkt) if it is feasible and
  - $c^t x < \infty$ for all $x \in P$ (for maximization problems)
  - $c^t x > -\infty$ for all $x \in P$ (for minimization problems)

Definitions

Definition 2

A polytop is a set $P \subseteq \mathbb{R}^n$ that is the convex hull of a finite set of points, i.e., $P = \mathrm{conv}(X)$ where

$$\mathrm{conv}(X) = \left\{ \sum_{i=1}^{\ell} \lambda_i x_i \mid \ell \in \mathbb{N}, x_1, \ldots, x_\ell \in X, \lambda_i \geq 0, \sum_i \lambda_i = 1 \right\}$$

and $|X| = c$.

Definitions

Definition 3

A polyhedron is a set $P \subseteq \mathbb{R}^n$ that can be represented as the intersection of finitely many half-spaces $\{H(a_1, b_1), \ldots, H(a_m, b_m)\}$, where

$$H(a_i, b_i) = \{x \in \mathbb{R}^n \mid a_i x \leq b_i\} \ .$$

Definitions

Theorem 4
P is a bounded polyhedron iff P is a polytop.

Definition 5

Let $P \subseteq \mathbb{R}^n$, $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$. The hyperplane

$$H(a,b) = \{x \in \mathbb{R}^n \mid ax = b\}$$

is a supporting hyperplane of P if $\max\{ax \mid x \in P\} = b$.

Definition 6

Let $P \subseteq \mathbb{R}^n$. F is a face of P if $F = P$ or $F = P \cap H$ for some supporting hyperplane H.

Definition 7

Let $P \subseteq \mathbb{R}^n$.

- v is a vertex of P if $\{v\}$ is a face of P.
- e is an edge of P if e is a face and $\dim(e) = 1$.
- F is a facet of P if F is a face and $\dim(e) = \dim(P) - 1$.

Definition 5

Let $P \subseteq \mathbb{R}^n$, $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$. The hyperplane

$$H(a,b) = \{x \in \mathbb{R}^n \mid ax = b\}$$

is a supporting hyperplane of P if $\max\{ax \mid x \in P\} = b$.

Definition 6

Let $P \subseteq \mathbb{R}^n$. F is a face of P if $F = P$ or $F = P \cap H$ for some supporting hyperplane H.

Definition 7

Let $P \subseteq \mathbb{R}^n$.

- v is a vertex of P if $\{v\}$ is a face of P.
- e is an edge of P if e is a face and $\dim(e) = 1$.
- F is a facet of P if F is a face and $\dim(e) = \dim(P) - 1$.

Definition 5

Let $P \subseteq \mathbb{R}^n$, $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$. The hyperplane

$$H(a,b) = \{x \in \mathbb{R}^n \mid ax = b\}$$

is a supporting hyperplane of P if $\max\{ax \mid x \in P\} = b$.

Definition 6

Let $P \subseteq \mathbb{R}^n$. F is a face of P if $F = P$ or $F = P \cap H$ for some supporting hyperplane H.

Definition 7

Let $P \subseteq \mathbb{R}^n$.

- v is a vertex of P if $\{v\}$ is a face of P.
- e is an edge of P if e is a face and $\dim(e) = 1$.
- F is a facet of P if F is a face and $\dim(e) = \dim(P) - 1$.

Observation

The feasible region of an LP is a Polyhedron.

Convex Sets

Theorem 8
If there exists an optimal solution to an LP then there exists an optimum solution that is a vertex.

Proof

- suppose x is optimal solution that is not a vertex
- there exists direction $d \neq 0$ such that $x \pm d \in P$
- $Ad = 0$ because $A(x \pm d) = b$
- Wlog. assume $c^t d \geq 0$ (by taking either d or $-d$)
- Consider $x + \lambda d, \lambda > 0$

Convex Sets

Theorem 8
If there exists an optimal solution to an LP then there exists an optimum solution that is a vertex.

Proof

- suppose x is optimal solution that is not a vertex
- there exists direction $d \neq 0$ such that $x \pm d \in P$
- $Ad = 0$ because $A(x \pm d) = b$
- Wlog. assume $c^t d \geq 0$ (by taking either d or $-d$)
- Consider $x + \lambda d$, $\lambda > 0$

Convex Sets

Theorem 8

If there exists an optimal solution to an LP then there exists an optimum solution that is a vertex.

Proof

- suppose x is optimal solution that is not a vertex
- there exists direction $d \neq 0$ such that $x \pm d \in P$
- $Ad = 0$ because $A(x \pm d) = b$
- Wlog. assume $c^t d \geq 0$ (by taking either d or $-d$)
- Consider $x + \lambda d$, $\lambda > 0$

Convex Sets

Theorem 8

If there exists an optimal solution to an LP then there exists an optimum solution that is a vertex.

Proof

- suppose x is optimal solution that is not a vertex
- there exists direction $d \neq 0$ such that $x \pm d \in P$
- $Ad = 0$ because $A(x \pm d) = b$
- Wlog. assume $c^t d \geq 0$ (by taking either d or $-d$)
- Consider $x + \lambda d$, $\lambda > 0$

Convex Sets

Theorem 8

If there exists an optimal solution to an LP then there exists an optimum solution that is a vertex.

Proof

- suppose x is optimal solution that is not a vertex
- there exists direction $d \neq 0$ such that $x \pm d \in P$
- $Ad = 0$ because $A(x \pm d) = b$
- Wlog. assume $c^t d \geq 0$ (by taking either d or $-d$)
- Consider $x + \lambda d$, $\lambda > 0$

Convex Sets

Theorem 8

If there exists an optimal solution to an LP then there exists an optimum solution that is a vertex.

Proof

- suppose x is optimal solution that is not a vertex
- there exists direction $d \neq 0$ such that $x \pm d \in P$
- $Ad = 0$ because $A(x \pm d) = b$
- Wlog. assume $c^t d \geq 0$ (by taking either d or $-d$)
- Consider $x + \lambda d$, $\lambda > 0$

Convex Sets

Case 1. [$\exists j$ s.t. $d_j < 0$]

- increase λ to λ' until first component of $x + \lambda d$ hits 0
- $x + \lambda' d$ is feasible. Since $A(x + \lambda' d) = b$ and $x + \lambda' d \geq 0$
- $x + \lambda' d$ has one more zero-component ($d_k = 0$ for $x_k = 0$ as $x \pm d \in P$)
- $c^t x' = c^t(x + \lambda' d) = c^t x + \lambda' c^t d \geq c^t x$

Case 2. [$d_j \geq 0$ for all j and $c^t d > 0$]

- $x + \lambda d$ is feasible for all $\lambda \geq 0$ since $A(x + \lambda d) = b$ and $x + \lambda d \geq x \geq 0$
- as $\lambda \to \infty$, $c^t(x + \lambda d) \to \infty$ as $c^t d > 0$

Convex Sets

Case 1. $[\exists j \text{ s.t. } d_j < 0]$

- increase λ to λ' until first component of $x + \lambda d$ hits 0
- $x + \lambda' d$ is feasible. Since $A(x + \lambda' d) = b$ and $x + \lambda' d \geq 0$
- $x + \lambda' d$ has one more zero-component ($d_k = 0$ for $x_k = 0$ as $x \pm d \in P$)
- $c^t x' = c^t(x + \lambda' d) = c^t x + \lambda' c^t d > c^t x$

Case 2. $[d_j \geq 0$ for all j and $c^t d > 0]$

- $x + \lambda d$ is feasible for all $\lambda \geq 0$ since $A(x + \lambda d) = b$ and $x + \lambda d \geq x \geq 0$
- as $\lambda \to \infty$, $c^t(x + \lambda d) \to \infty$ as $c^t d > 0$

Convex Sets

Case 1. $[\exists j \text{ s.t. } d_j < 0]$

- increase λ to λ' until first component of $x + \lambda d$ hits 0
- $x + \lambda' d$ is feasible. Since $A(x + \lambda' d) = b$ and $x + \lambda' d \geq 0$
- $x + \lambda' d$ has one more zero-component ($d_k = 0$ for $x_k = 0$ as $x \pm d \in P$)
- $c^t x' = c^t(x + \lambda' d) = c^t x + \lambda' c^t d \geq c^t x$

Case 2. $[d_j \geq 0 \text{ for all } j \text{ and } c^t d > 0]$

- $x + \lambda d$ is feasible for all $\lambda \geq 0$ since $A(x + \lambda d) = b$ and $x + \lambda d \geq x \geq 0$
- as $\lambda \to \infty$, $c^t(x + \lambda d) \to \infty$ as $c^t d > 0$

Convex Sets

Case 1. [$\exists j$ s.t. $d_j < 0$]

- increase λ to λ' until first component of $x + \lambda d$ hits 0
- $x + \lambda' d$ is feasible. Since $A(x + \lambda' d) = b$ and $x + \lambda' d \geq 0$
- $x + \lambda' d$ has one more zero-component ($d_k = 0$ for $x_k = 0$ as $x \pm d \in P$)
- $c^t x' = c^t(x + \lambda' d) = c^t x + \lambda' c^t d \geq c^t x$

Case 2. [$d_j \geq 0$ for all j and $c^t d > 0$]

- $x + \lambda d$ is feasible for all $\lambda \geq 0$ since $A(x + \lambda d) = b$ and $x + \lambda d \geq x \geq 0$
- as $\lambda \to \infty$, $c^t(x + \lambda d) \to \infty$ as $c^t d > 0$

Convex Sets

Case 1. $[\exists j \text{ s.t. } d_j < 0]$

- increase λ to λ' until first component of $x + \lambda d$ hits 0
- $x + \lambda' d$ is feasible. Since $A(x + \lambda' d) = b$ and $x + \lambda' d \geq 0$
- $x + \lambda' d$ has one more zero-component ($d_k = 0$ for $x_k = 0$ as $x \pm d \in P$)
- $c^t x' = c^t(x + \lambda' d) = c^t x + \lambda' c^t d \geq c^t x$

Case 2. $[d_j \geq 0 \text{ for all } j \text{ and } c^t d > 0]$

- $x + \lambda d$ is feasible for all $\lambda \geq 0$ since $A(x + \lambda d) = b$ and $x + \lambda d \geq x \geq 0$
- as $\lambda \to \infty$, $c^t(x + \lambda d) \to \infty$ as $c^t d > 0$

Convex Sets

Case 1. [$\exists j$ s.t. $d_j < 0$]

- increase λ to λ' until first component of $x + \lambda d$ hits 0
- $x + \lambda' d$ is feasible. Since $A(x + \lambda' d) = b$ and $x + \lambda' d \geq 0$
- $x + \lambda' d$ has one more zero-component ($d_k = 0$ for $x_k = 0$ as $x \pm d \in P$)
- $c^t x' = c^t(x + \lambda' d) = c^t x + \lambda' c^t d \geq c^t x$

Case 2. [$d_j \geq 0$ for all j and $c^t d > 0$]

- $x + \lambda d$ is feasible for all $\lambda \geq 0$ since $A(x + \lambda d) = b$ and $x + \lambda d \geq x \geq 0$
- as $\lambda \to \infty$, $c^t(x + \lambda d) \to \infty$ as $c^t d > 0$

Convex Sets

Case 1. $[\exists j$ s.t. $d_j < 0]$

- increase λ to λ' until first component of $x + \lambda d$ hits 0
- $x + \lambda' d$ is feasible. Since $A(x + \lambda' d) = b$ and $x + \lambda' d \geq 0$
- $x + \lambda' d$ has one more zero-component ($d_k = 0$ for $x_k = 0$ as $x \pm d \in P$)
- $c^t x' = c^t(x + \lambda' d) = c^t x + \lambda' c^t d \geq c^t x$

Case 2. $[d_j \geq 0$ for all j and $c^t d > 0]$

- $x + \lambda d$ is feasible for all $\lambda \geq 0$ since $A(x + \lambda d) = b$ and $x + \lambda d \geq x \geq 0$
- as $\lambda \to \infty$, $c^t(x + \lambda d) \to \infty$ as $c^t d > 0$

Convex Sets

Case 1. $[\exists j \text{ s.t. } d_j < 0]$

- increase λ to λ' until first component of $x + \lambda d$ hits 0
- $x + \lambda' d$ is feasible. Since $A(x + \lambda' d) = b$ and $x + \lambda' d \geq 0$
- $x + \lambda' d$ has one more zero-component ($d_k = 0$ for $x_k = 0$ as $x \pm d \in P$)
- $c^t x' = c^t (x + \lambda' d) = c^t x + \lambda' c^t d \geq c^t x$

Case 2. $[d_j \geq 0 \text{ for all } j \text{ and } c^t d > 0]$

- $x + \lambda d$ is feasible for all $\lambda \geq 0$ since $A(x + \lambda d) = b$ and $x + \lambda d \geq x \geq 0$
- as $\lambda \to \infty$, $c^t(x + \lambda d) \to \infty$ as $c^t d > 0$

Convex Sets

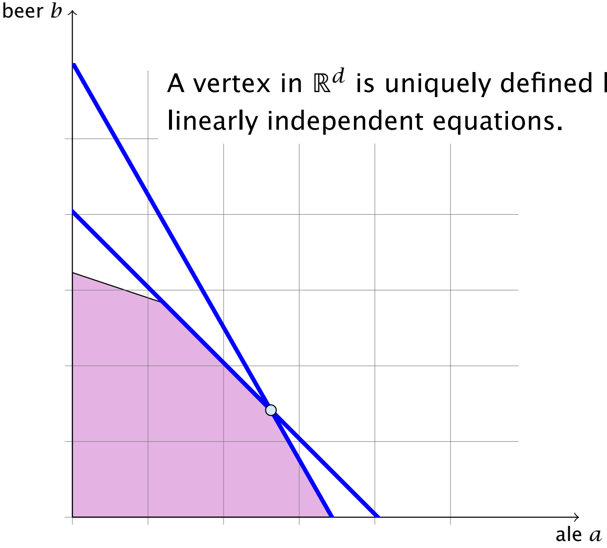
Case 1. $[\exists j \text{ s.t. } d_j < 0]$

- increase λ to λ' until first component of $x + \lambda d$ hits 0
- $x + \lambda' d$ is feasible. Since $A(x + \lambda' d) = b$ and $x + \lambda' d \geq 0$
- $x + \lambda' d$ has one more zero-component ($d_k = 0$ for $x_k = 0$ as $x \pm d \in P$)
- $c^t x' = c^t(x + \lambda' d) = c^t x + \lambda' c^t d \geq c^t x$

Case 2. $[d_j \geq 0 \text{ for all } j \text{ and } c^t d > 0]$

- $x + \lambda d$ is feasible for all $\lambda \geq 0$ since $A(x + \lambda d) = b$ and $x + \lambda d \geq x \geq 0$
- as $\lambda \to \infty$, $c^t(x + \lambda d) \to \infty$ as $c^t d > 0$

Algebraic View



A vertex in $\mathbb{R}^d$ is uniquely defined by d linearly independent equations.

Notation

Suppose $B \subseteq \{1 \dots n\}$ is a set of column-indices. Define A_B as the subset of columns of A indexed by B.

Theorem 9

*Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. Then x is a vertex **iff** A_B has linearly independent columns.*

Notation

Suppose $B \subseteq \{1 \dots n\}$ is a set of column-indices. Define A_B as the subset of columns of A indexed by B.

Theorem 9

*Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. Then x is a vertex **iff** A_B has linearly independent columns.*

Theorem 9

*Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. Then x is a vertex **iff** A_B has linearly independent columns.*

Proof ($\Leftarrow$)

- assume x is not a vertex
- there exists direction d s.t. $x \pm d \in P$
- $Ad = 0$ because $A(x \pm d) = b$
- define $B' = \{j \mid d_j \neq 0\}$
- $A_{B'}$ has linearly dependent columns as $Ad = 0$
- $d_j = 0$ for all j with $x_j = 0$ as $x \pm d \geq 0$
- Hence, $B' \subseteq B$, $A_{B'}$ is sub-matrix of A_B

Theorem 9

*Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. Then x is a vertex **iff** A_B has linearly independent columns.*

Proof ($\Leftarrow$)

- assume x is not a vertex
- there exists direction d s.t. $x \pm d \in P$
- $Ad = 0$ because $A(x \pm d) = b$
- define $B' = \{j \mid d_j \neq 0\}$
- $A_{B'}$ has linearly dependent columns as $Ad = 0$
- $d_j = 0$ for all j with $x_j = 0$ as $x \pm d \geq 0$
- Hence, $B' \subseteq B$, $A_{B'}$ is sub-matrix of A_B

Theorem 9

*Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. Then x is a vertex **iff** A_B has linearly independent columns.*

Proof (⇐)

- assume x is not a vertex
- there exists direction d s.t. $x \pm d \in P$
- $Ad = 0$ because $A(x \pm d) = b$
- define $B' = \{j \mid d_j \neq 0\}$
- $A_{B'}$ has linearly dependent columns as $Ad = 0$
- $d_j = 0$ for all j with $x_j = 0$ as $x \pm d \geq 0$
- Hence, $B' \subseteq B$, $A_{B'}$ is sub-matrix of A_B

Theorem 9

*Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. Then x is a vertex **iff** A_B has linearly independent columns.*

Proof (⇐)

- assume x is not a vertex
- there exists direction d s.t. $x \pm d \in P$
- $Ad = 0$ because $A(x \pm d) = b$
- define $B' = \{j \mid d_j \neq 0\}$
- $A_{B'}$ has linearly dependent columns as $Ad = 0$
- $d_j = 0$ for all j with $x_j = 0$ as $x \pm d \geq 0$
- Hence, $B' \subseteq B$, $A_{B'}$ is sub-matrix of A_B

Theorem 9

*Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. Then x is a vertex **iff** A_B has linearly independent columns.*

Proof (⇐)

- assume x is not a vertex
- there exists direction d s.t. $x \pm d \in P$
- $Ad = 0$ because $A(x \pm d) = b$
- define $B' = \{j \mid d_j \neq 0\}$
- $A_{B'}$ has linearly dependent columns as $Ad = 0$
- $d_j = 0$ for all j with $x_j = 0$ as $x \pm d \geq 0$
- Hence, $B' \subseteq B$, $A_{B'}$ is sub-matrix of A_B

Theorem 9

*Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. Then x is a vertex **iff** A_B has linearly independent columns.*

Proof (⇐)

- assume x is not a vertex
- there exists direction d s.t. $x \pm d \in P$
- $Ad = 0$ because $A(x \pm d) = b$
- define $B' = \{j \mid d_j \neq 0\}$
- $A_{B'}$ has linearly dependent columns as $Ad = 0$
- $d_j = 0$ for all j with $x_j = 0$ as $x \pm d \geq 0$
- Hence, $B' \subseteq B$, $A_{B'}$ is sub-matrix of A_B

Theorem 9

*Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. Then x is a vertex **iff** A_B has linearly independent columns.*

Proof (⇐)

- assume x is not a vertex
- there exists direction d s.t. $x \pm d \in P$
- $Ad = 0$ because $A(x \pm d) = b$
- define $B' = \{j \mid d_j \neq 0\}$
- $A_{B'}$ has linearly dependent columns as $Ad = 0$
- $d_j = 0$ for all j with $x_j = 0$ as $x \pm d \geq 0$
- Hence, $B' \subseteq B$, $A_{B'}$ is sub-matrix of A_B

Theorem 9

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. Then x is a vertex ***iff*** *A_B has linearly independent columns.*

Proof (⇐)

- assume x is not a vertex
- there exists direction d s.t. $x \pm d \in P$
- $Ad = 0$ because $A(x \pm d) = b$
- define $B' = \{j \mid d_j \neq 0\}$
- $A_{B'}$ has linearly dependent columns as $Ad = 0$
- $d_j = 0$ for all j with $x_j = 0$ as $x \pm d \geq 0$
- Hence, $B' \subseteq B$, $A_{B'}$ is sub-matrix of A_B

Theorem 9

*Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. Then x is a vertex **iff** A_B has linearly independent columns.*

Proof (⇒)

Theorem 9

*Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. Then x is a vertex **iff** A_B has linearly independent columns.*

Proof (⇒)

- assume A_B has linearly dependent columns
- there exists $d \neq 0$ such that $A_B d = 0$
- extend d to $\mathbb{R}^n$ by adding 0-components
- now, $Ad = 0$ and $d_j = 0$ whenever $x_j = 0$
- for sufficiently small λ we have $x \pm \lambda d \in P$
- hence, x is not a vertex

Theorem 9

Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. Then x is a vertex ***iff*** *A_B has linearly independent columns.*

Proof (⇒)

- assume A_B has linearly dependent columns
- there exists $d \neq 0$ such that $A_B d = 0$
- extend d to $\mathbb{R}^n$ by adding 0-components
- now, $Ad = 0$ and $d_j = 0$ whenever $x_j = 0$
- for sufficiently small λ we have $x \pm \lambda d \in P$
- hence, x is not a vertex

Theorem 9

*Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. Then x is a vertex **iff** A_B has linearly independent columns.*

Proof (⇒)

- assume A_B has linearly dependent columns
- there exists $d \neq 0$ such that $A_B d = 0$
- extend d to $\mathbb{R}^n$ by adding 0-components
- now, $Ad = 0$ and $d_j = 0$ whenever $x_j = 0$
- for sufficiently small λ we have $x \pm \lambda d \in P$
- hence, x is not a vertex

Theorem 9

*Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. Then x is a vertex **iff** A_B has linearly independent columns.*

Proof (⇒)

- assume A_B has linearly dependent columns
- there exists $d \neq 0$ such that $A_B d = 0$
- extend d to $\mathbb{R}^n$ by adding 0-components
- now, $Ad = 0$ and $d_j = 0$ whenever $x_j = 0$
- for sufficiently small λ we have $x \pm \lambda d \in P$
- hence, x is not a vertex

Theorem 9

*Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. Then x is a vertex **iff** A_B has linearly independent columns.*

Proof (⇒)

- assume A_B has linearly dependent columns
- there exists $d \neq 0$ such that $A_B d = 0$
- extend d to $\mathbb{R}^n$ by adding 0-components
- now, $Ad = 0$ and $d_j = 0$ whenever $x_j = 0$
- for sufficiently small λ we have $x \pm \lambda d \in P$
- hence, x is not a vertex

Theorem 9

*Let $P = \{x \mid Ax = b, x \geq 0\}$. For $x \in P$, define $B = \{j \mid x_j > 0\}$. Then x is a vertex **iff** A_B has linearly independent columns.*

Proof (⇒)

- assume A_B has linearly dependent columns
- there exists $d \neq 0$ such that $A_B d = 0$
- extend d to $\mathbb{R}^n$ by adding 0-components
- now, $Ad = 0$ and $d_j = 0$ whenever $x_j = 0$
- for sufficiently small λ we have $x \pm \lambda d \in P$
- hence, x is not a vertex

Observation

For an LP we can assume wlog. that the matrix A has full row-rank. This means $\text{rank}(A) = m$.

Observation

For an LP we can assume wlog. that the matrix A has full row-rank. This means $\text{rank}(A) = m$.

- assume that $\text{rank}(A) < m$
- assume wlog. that the first row A_1 lies in the span of the other rows $A_2, \dots, A_m$; this means

$$A_1 = \sum_{i=2}^{m} \lambda_i \cdot A_i \text{, for suitable } \lambda_i$$

C1 if now $b_1 = \sum_{i=2}^{m} \lambda_i \cdot b_i$ then for all x with $A_i x = b_i$ we also have $A_1 x = b_1$; hence the first constraint is superfluous

C2 if $b_1 \neq \sum_{i=2}^{m} \lambda_i \cdot b_i$ then the LP is infeasible, since for all x that fulfill constraints $A_2, \dots, A_m$ we have

$$A_1 x = \sum_{i=2}^{m} \lambda_i \cdot A_i x = \sum_{i=2}^{m} \lambda_i \cdot b_i \neq b_1$$

Observation

For an LP we can assume wlog. that the matrix A has full row-rank. This means $\operatorname{rank}(A) = m$.

- assume that $\operatorname{rank}(A) < m$
- assume wlog. that the first row A_1 lies in the span of the other rows $A_2, \dots, A_m$; this means

$$A_1 = \sum_{i=2}^{m} \lambda_i \cdot A_i \text{, for suitable } \lambda_i$$

C1 if now $b_1 = \sum_{i=2}^{m} \lambda_i \cdot b_i$ then for all x with $A_i x = b_i$ we also have $A_1 x = b_1$; hence the first constraint is superfluous

C2 if $b_1 \neq \sum_{i=2}^{m} \lambda_i \cdot b_i$ then the LP is infeasible, since for all x that fulfill constraints $A_2, \dots, A_m$ we have

$$A_1 x = \sum_{i=2}^{m} \lambda_i \cdot A_i x = \sum_{i=2}^{m} \lambda_i \cdot b_i \neq b_1$$

Observation

For an LP we can assume wlog. that the matrix A has full row-rank. This means $\operatorname{rank}(A) = m$.

- assume that $\operatorname{rank}(A) < m$
- assume wlog. that the first row A_1 lies in the span of the other rows $A_2, \dots, A_m$; this means

$$A_1 = \sum_{i=2}^{m} \lambda_i \cdot A_i, \text{ for suitable } \lambda_i$$

C1 if now $b_1 = \sum_{i=2}^{m} \lambda_i \cdot b_i$ then for all x with $A_i x = b_i$ we also have $A_1 x = b_1$; hence the first constraint is superfluous

C2 if $b_1 \neq \sum_{i=2}^{m} \lambda_i \cdot b_i$ then the LP is infeasible, since for all x that fulfill constraints $A_2, \dots, A_m$ we have

$$A_1 x = \sum_{i=2}^{m} \lambda_i \cdot A_i x = \sum_{i=2}^{m} \lambda_i \cdot b_i \neq b_1$$

Observation

For an LP we can assume wlog. that the matrix A has full row-rank. This means $\operatorname{rank}(A) = m$.

- assume that $\operatorname{rank}(A) < m$
- assume wlog. that the first row A_1 lies in the span of the other rows $A_2, \ldots, A_m$; this means

$$A_1 = \sum_{i=2}^{m} \lambda_i \cdot A_i, \text{ for suitable } \lambda_i$$

C1 if now $b_1 = \sum_{i=2}^{m} \lambda_i \cdot b_i$ then for all x with $A_i x = b_i$ we also have $A_1 x = b_1$; hence the first constraint is superfluous

C2 if $b_1 \neq \sum_{i=2}^{m} \lambda_i \cdot b_i$ then the LP is infeasible, since for all x that fulfill constraints $A_2, \ldots, A_m$ we have

$$A_1 x = \sum_{i=2}^{m} \lambda_i \cdot A_i x = \sum_{i=2}^{m} \lambda_i \cdot b_i \neq b_1$$

Observation

For an LP we can assume wlog. that the matrix A has full row-rank. This means $\operatorname{rank}(A) = m$.

- assume that $\operatorname{rank}(A) < m$
- assume wlog. that the first row A_1 lies in the span of the other rows $A_2, \dots, A_m$; this means

$$A_1 = \sum_{i=2}^{m} \lambda_i \cdot A_i, \text{ for suitable } \lambda_i$$

C1 if now $b_1 = \sum_{i=2}^{m} \lambda_i \cdot b_i$ then for all x with $A_i x = b_i$ we also have $A_1 x = b_1$; hence the first constraint is superfluous

C2 if $b_1 \neq \sum_{i=2}^{m} \lambda_i \cdot b_i$ then the LP is infeasible, since for all x that fulfill constraints $A_2, \dots, A_m$ we have

$$A_1 x = \sum_{i=2}^{m} \lambda_i \cdot A_i x = \sum_{i=2}^{m} \lambda_i \cdot b_i \neq b_1$$

Observation

For an LP we can assume wlog. that the matrix A has full row-rank. This means $\operatorname{rank}(A) = m$.

- assume that $\operatorname{rank}(A) < m$
- assume wlog. that the first row A_1 lies in the span of the other rows $A_2, \ldots, A_m$; this means

$$A_1 = \sum_{i=2}^{m} \lambda_i \cdot A_i, \text{ for suitable } \lambda_i$$

C1 if now $b_1 = \sum_{i=2}^{m} \lambda_i \cdot b_i$ then for all x with $A_i x = b_i$ we also have $A_1 x = b_1$; hence the first constraint is superfluous

C2 if $b_1 \neq \sum_{i=2}^{m} \lambda_i \cdot b_i$ then the LP is infeasible, since for all x that fulfill constraints $A_2, \ldots, A_m$ we have

$$A_1 x = \sum_{i=2}^{m} \lambda_i \cdot A_i x = \sum_{i=2}^{m} \lambda_i \cdot b_i \neq b_1$$

Observation

For an LP we can assume wlog. that the matrix A has full row-rank. This means $\mathrm{rank}(A) = m$.

- assume that $\mathrm{rank}(A) < m$
- assume wlog. that the first row A_1 lies in the span of the other rows $A_2, \ldots, A_m$; this means

$$A_1 = \sum_{i=2}^{m} \lambda_i \cdot A_i \text{, for suitable } \lambda_i$$

C1 if now $b_1 = \sum_{i=2}^{m} \lambda_i \cdot b_i$ then for all x with $A_i x = b_i$ we also have $A_1 x = b_1$; hence the first constraint is superfluous

C2 if $b_1 \neq \sum_{i=2}^{m} \lambda_i \cdot b_i$ then the LP is infeasible, since for all x that fulfill constraints $A_2, \ldots, A_m$ we have

$$A_1 x = \sum_{i=2}^{m} \lambda_i \cdot A_i x = \sum_{i=2}^{m} \lambda_i \cdot b_i \neq b_1$$

Observation

For an LP we can assume wlog. that the matrix A has full row-rank. This means $\text{rank}(A) = m$.

- assume that $\text{rank}(A) < m$
- assume wlog. that the first row A_1 lies in the span of the other rows $A_2, \dots, A_m$; this means

$$A_1 = \sum_{i=2}^{m} \lambda_i \cdot A_i, \text{ for suitable } \lambda_i$$

C1 if now $b_1 = \sum_{i=2}^{m} \lambda_i \cdot b_i$ then for all x with $A_i x = b_i$ we also have $A_1 x = b_1$; hence the first constraint is superfluous

C2 if $b_1 \neq \sum_{i=2}^{m} \lambda_i \cdot b_i$ then the LP is infeasible, since for all x that fulfill constraints $A_2, \dots, A_m$ we have

$$A_1 x = \sum_{i=2}^{m} \lambda_i \cdot A_i x = \sum_{i=2}^{m} \lambda_i \cdot b_i \neq b_1$$

Observation

For an LP we can assume wlog. that the matrix A has full row-rank. This means $\text{rank}(A) = m$.

- assume that $\text{rank}(A) < m$
- assume wlog. that the first row A_1 lies in the span of the other rows $A_2, \dots, A_m$; this means

$$A_1 = \sum_{i=2}^{m} \lambda_i \cdot A_i, \text{ for suitable } \lambda_i$$

C1 if now $b_1 = \sum_{i=2}^{m} \lambda_i \cdot b_i$ then for all x with $A_i x = b_i$ we also have $A_1 x = b_1$; hence the first constraint is superfluous

C2 if $b_1 \neq \sum_{i=2}^{m} \lambda_i \cdot b_i$ then the LP is infeasible, since for all x that fulfill constraints $A_2, \dots, A_m$ we have

$$A_1 x = \sum_{i=2}^{m} \lambda_i \cdot A_i x = \sum_{i=2}^{m} \lambda_i \cdot b_i \neq b_1$$

Observation

For an LP we can assume wlog. that the matrix A has full row-rank. This means $\operatorname{rank}(A) = m$.

- assume that $\operatorname{rank}(A) < m$
- assume wlog. that the first row A_1 lies in the span of the other rows $A_2, \dots, A_m$; this means

$$A_1 = \sum_{i=2}^{m} \lambda_i \cdot A_i, \text{ for suitable } \lambda_i$$

C1 if now $b_1 = \sum_{i=2}^{m} \lambda_i \cdot b_i$ then for all x with $A_i x = b_i$ we also have $A_1 x = b_1$; hence the first constraint is superfluous

C2 if $b_1 \neq \sum_{i=2}^{m} \lambda_i \cdot b_i$ then the LP is infeasible, since for all x that fulfill constraints $A_2, \dots, A_m$ we have

$$A_1 x = \sum_{i=2}^{m} \lambda_i \cdot A_i x = \sum_{i=2}^{m} \lambda_i \cdot b_i \neq b_1$$

From now on we will always assume that the constraint matrix of a standard form LP has full row rank.

Theorem 10

Given $P = \{x \mid Ax = b, x \geq 0\}$. x is a vertex iff there exists $B \subseteq \{1, \ldots, n\}$ with $|B| = m$ and

- A_B is non-singular
- $x_B = A_B^{-1} b \geq 0$
- $x_N = 0$

where $N = \{1, \ldots, n\} \setminus B$.

Proof

Take $B = \{j \mid x_j > 0\}$ and augment with linearly independent columns until $|B| = m$; always possible since $\mathrm{rank}(A) = m$.

Theorem 10

Given $P = \{x \mid Ax = b, x \geq 0\}$. x *is a vertex iff there exists* $B \subseteq \{1, \ldots, n\}$ *with* $|B| = m$ *and*

- A_B is non-singular
- $x_B = A_B^{-1} b \geq 0$
- $x_N = 0$

where $N = \{1, \ldots, n\} \setminus B$.

Proof

Take $B = \{j \mid x_j > 0\}$ and augment with linearly independent columns until $|B| = m$; always possible since $\operatorname{rank}(A) = m$.

Basic Feasible Solutions

$x \in \mathbb{R}^n$ is called basic solution (Basislösung) if $Ax = b$ and $\text{rank}(A_J) = |J|$ where $J = \{j \mid x_j \neq 0\}$;

x is a basic **feasible** solution (gültige Basislösung) if in addition $x \geq 0$.

A basis (Basis) is an index set $B \subseteq \{1, \dots, n\}$ with $\text{rank}(A_B) = m$ and $|B| = m$.

$x \in \mathbb{R}^n$ with $A_B x = b$ and $x_j = 0$ for all $j \notin B$ is the basic solution associated to basis B (die zu B assoziierte Basislösung)

Basic Feasible Solutions

$x \in \mathbb{R}^n$ is called basic solution (Basislösung) if $Ax = b$ and $\text{rank}(A_J) = |J|$ where $J = \{j \mid x_j \neq 0\}$;

x is a basic feasible solution (gültige Basislösung) if in addition $x \geq 0$.

A basis (Basis) is an index set $B \subseteq \{1, \ldots, n\}$ with $\text{rank}(A_B) = m$ and $|B| = m$.

$x \in \mathbb{R}^n$ with $A_B x = b$ and $x_j = 0$ for all $j \notin B$ is the basic solution associated to basis B (die zu B assoziierte Basislösung)

Basic Feasible Solutions

$x \in \mathbb{R}^n$ is called basic solution (Basislösung) if $Ax = b$ and $\text{rank}(A_J) = |J|$ where $J = \{j \mid x_j \neq 0\}$;

x is a basic **feasible** solution (gültige Basislösung) if in addition $x \geq 0$.

A basis (Basis) is an index set $B \subseteq \{1, \ldots, n\}$ with $\text{rank}(A_B) = m$ and $|B| = m$.

$x \in \mathbb{R}^n$ with $A_B x = b$ and $x_j = 0$ for all $j \notin B$ is the basic solution associated to basis B (die zu B assoziierte Basislösung)

Basic Feasible Solutions

$x \in \mathbb{R}^n$ is called basic solution (Basislösung) if $Ax = b$ and $\mathrm{rank}(A_J) = |J|$ where $J = \{j \mid x_j \neq 0\}$;

x is a basic **feasible** solution (gültige Basislösung) if in addition $x \geq 0$.

A basis (Basis) is an index set $B \subseteq \{1, \ldots, n\}$ with $\mathrm{rank}(A_B) = m$ and $|B| = m$.

$x \in \mathbb{R}^n$ with $A_B x = b$ and $x_j = 0$ for all $j \notin B$ is the basic solution associated to basis B (die zu B assoziierte Basislösung)

Basic Feasible Solutions

$x \in \mathbb{R}^n$ is called basic solution (Basislösung) if $Ax = b$ and $\text{rank}(A_J) = |J|$ where $J = \{j \mid x_j \neq 0\}$;

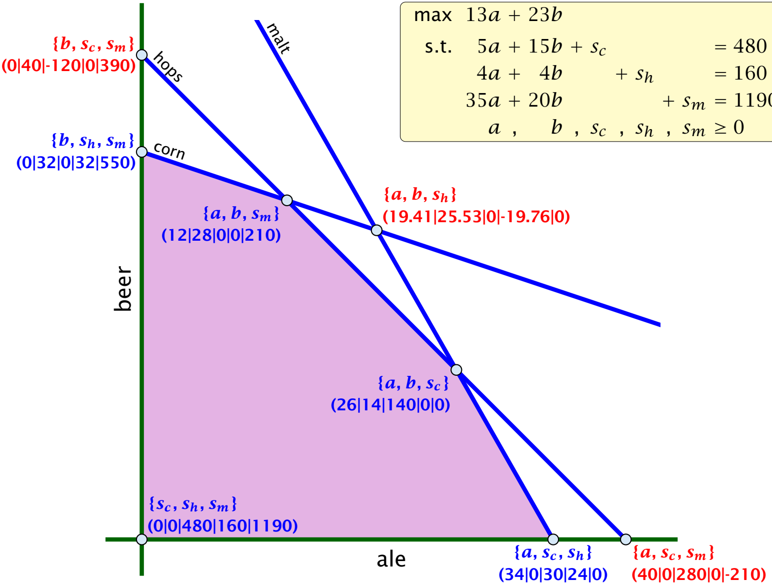
x is a basic **feasible** solution (gültige Basislösung) if in addition $x \geq 0$.

A basis (Basis) is an index set $B \subseteq \{1, \ldots, n\}$ with $\text{rank}(A_B) = m$ and $|B| = m$.

$x \in \mathbb{R}^n$ with $A_B x = b$ and $x_j = 0$ for all $j \notin B$ is the basic solution associated to basis B (die zu B assoziierte Basislösung)

Algebraic View

$$
\begin{aligned}
\max\quad & 13a + 23b \\
\text{s.t.}\quad & 5a + 15b + s_c = 480 \\
& 4a + 4b + s_h = 160 \\
& 35a + 20b + s_m = 1190 \\
& a\ ,\ b\ ,\ s_c\ ,\ s_h\ ,\ s_m \geq 0
\end{aligned}
$$



Fundamental Questions

Linear Programming Problem (LP)

Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$ s.t. $Ax = b$, $x \geq 0$, $c^t x \geq \alpha$?

Questions:

- Is LP in NP? yes!
- Is LP in co-NP?
- Is LP in P?

Proof:

- Given a basis B we can compute the associated basis solution by calculating $A_B^{-1} b$ in polynomial time; then we can also compute the profit.

Fundamental Questions

Linear Programming Problem (LP)

Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$ s.t. $Ax = b$, $x \geq 0$, $c^t x \geq \alpha$?

Questions:

- Is LP in NP? yes!
- Is LP in co-NP?
- Is LP in P?

Proof:

- Given a basis B we can compute the associated basis solution by calculating $A_B^{-1}b$ in polynomial time; then we can also compute the profit.

Observation

We can compute an optimal solution to a linear program in time $\mathcal{O}\left(\binom{n}{m} \cdot \text{poly}(n,m)\right)$.

- there are only $\binom{n}{m}$ different bases.
- compute the profit of each of them and take the maximum

4 Simplex Algorithm

Enumerating all basic feasible solutions (BFS), in order to find the optimum is slow.

Simplex Algorithm [George Dantzig 1947]
Move from BFS to adjacent BFS, without decreasing objective function.

Two BFSs are called adjacent if the bases just differ in one variable.

4 Simplex Algorithm

Enumerating all basic feasible solutions (BFS), in order to find the optimum is slow.

Simplex Algorithm [George Dantzig 1947]
Move from BFS to adjacent BFS, without decreasing objective function.

Two BFSs are called adjacent if the bases just differ in one variable.

4 Simplex Algorithm

$$\begin{array}{lrcrcccccl}
\max & 13a & + & 23b & & & & \\
\text{s.t.} & 5a & + & 15b & + s_c & & & = 480 \\
& 4a & + & 4b & & + s_h & & = 160 \\
& 35a & + & 20b & & & + s_m & = 1190 \\
& a & , & b & , s_c & , s_h & , s_m & \geq 0
\end{array}$$

$$\begin{array}{lrcrccccl}
\max Z & & & & & & & \\
& 13a & + & 23b & & & & - Z = 0 \\
& 5a & + & 15b & + s_c & & & = 480 \\
& 4a & + & 4b & & + s_h & & = 160 \\
& 35a & + & 20b & & & + s_m & = 1190 \\
& a & , & b & , s_c & , s_h & , s_m & \geq 0
\end{array}$$

basis = $\{s_c, s_h, s_m\}$

$A = B = 0$

$Z = 0$

$s_c = 480$

$s_h = 160$

$s_m = 1190$

4 Simplex Algorithm

$$
\begin{array}{llllllll}
\max & 13a + 23b & & & & & & \\
\text{s.t.} & 5a + 15b & + s_c & & & = & 480 \\
 & 4a + 4b & & + s_h & & = & 160 \\
 & 35a + 20b & & & + s_m & = & 1190 \\
 & a\ ,\ b & ,\ s_c & ,\ s_h & ,\ s_m & \geq & 0
\end{array}
$$

$$
\begin{array}{lllllll}
\max Z & & & & & & \\
13a + 23b & & & & - Z & = & 0 \\
5a + 15b & + s_c & & & & = & 480 \\
4a + 4b & & + s_h & & & = & 160 \\
35a + 20b & & & + s_m & & = & 1190 \\
a\ ,\ b & ,\ s_c & ,\ s_h & ,\ s_m & & \geq & 0
\end{array}
$$

basis $= \{s_c, s_h, s_m\}$
$A = B = 0$
$Z = 0$

$s_c = 480$
$s_h = 160$
$s_m = 1190$

Pivoting Step

$$
\begin{array}{l}
\max Z \\
\begin{array}{rcrcccccl}
13a & + & 23b & & & & & - Z & = 0 \\
5a & + & 15b & + s_c & & & & & = 480 \\
4a & + & 4b & & + s_h & & & & = 160 \\
35a & + & 20b & & & + s_m & & & = 1190 \\
a & , & b & , s_c & , s_h & , s_m & & & \geq 0
\end{array}
\end{array}
$$

$$
\begin{array}{l}
\text{basis} = \{s_c, s_h, s_m\} \\
a = b = 0 \\
Z = 0 \\
s_c = 480 \\
s_h = 160 \\
s_m = 1190
\end{array}
$$

- choose variable to bring into the basis
- chosen variable should have positive coefficient in objective function
- apply test to find out by how much the variable can be increased
- pivot on row found by min-ratio test
- the existing basis variable in this row leaves the basis

Pivoting Step

$$
\begin{array}{l}
\max Z \\
\quad 13a + 23b \qquad\qquad\qquad\qquad - Z = 0 \\
\quad 5a + 15b + s_c \qquad\qquad\qquad = 480 \\
\quad 4a + 4b \quad + s_h \qquad\qquad = 160 \\
\quad 35a + 20b \qquad\quad + s_m \quad = 1190 \\
\quad a, \ b, \ s_c, \ s_h, \ s_m \qquad \geq 0
\end{array}
$$

basis $= \{s_c, s_h, s_m\}$

$a = b = 0$

$Z = 0$

$s_c = 480$

$s_h = 160$

$s_m = 1190$

- choose variable to bring into the basis
- chosen variable should have positive coefficient in objective function
- apply min-ratio test to find out by how much the variable can be increased
- pivot on row found by min-ratio test
- the existing basis variable in this row leaves the basis

Pivoting Step

max Z

$$\begin{aligned} 13a + 23b \quad\quad\quad\quad\quad\quad - Z &= 0 \\ 5a + 15b + s_c \quad\quad\quad\quad &= 480 \\ 4a + \ 4b \quad + s_h \quad\quad &= 160 \\ 35a + 20b \quad\quad\quad + s_m &= 1190 \\ a\ ,\ b\ ,\ s_c\ ,\ s_h\ ,\ s_m &\geq 0 \end{aligned}$$

basis = $\{s_c, s_h, s_m\}$

$a = b = 0$

$Z = 0$

$s_c = 480$

$s_h = 160$

$s_m = 1190$

- choose variable to bring into the basis
- chosen variable should have positive coefficient in objective function
- apply min-ratio test to find out by how much the variable can be increased
- pivot on row found by min-ratio test
- the existing basis variable in this row leaves the basis

Pivoting Step

$$
\begin{array}{l}
\max Z \\
\quad 13a + 23b \qquad\qquad\qquad\qquad - Z = 0 \\
\quad 5a + 15b + s_c \qquad\qquad\qquad\quad = 480 \\
\quad 4a + 4b \qquad + s_h \qquad\qquad\quad = 160 \\
\quad 35a + 20b \qquad\qquad + s_m \qquad = 1190 \\
\quad a\ ,\quad b\ ,\ s_c\ ,\ s_h\ ,\ s_m \qquad\quad \geq 0
\end{array}
$$

$$
\begin{array}{l}
\text{basis} = \{s_c, s_h, s_m\} \\
a = b = 0 \\
Z = 0 \\
s_c = 480 \\
s_h = 160 \\
s_m = 1190
\end{array}
$$

- choose variable to bring into the basis
- chosen variable should have positive coefficient in objective function
- apply min-ratio test to find out by how much the variable can be increased
- pivot on row found by min-ratio test
- the existing basis variable in this row leaves the basis

Pivoting Step

$$
\begin{array}{l}
\max Z \\
\quad 13a + 23b \qquad\qquad\qquad - Z = 0 \\
\quad 5a + 15b + s_c \qquad\qquad\quad = 480 \\
\quad 4a + \;\; 4b \quad + s_h \qquad\quad = 160 \\
\quad 35a + 20b \qquad\quad + s_m \;\; = 1190 \\
\quad a \;,\;\; b \;,\; s_c \;,\; s_h \;,\; s_m \quad \geq 0
\end{array}
$$

$$
\begin{array}{l}
\text{basis} = \{s_c, s_h, s_m\} \\
a = b = 0 \\
Z = 0 \\
s_c = 480 \\
s_h = 160 \\
s_m = 1190
\end{array}
$$

- choose variable to bring into the basis
- chosen variable should have positive coefficient in objective function
- apply min-ratio test to find out by how much the variable can be increased
- pivot on row found by min-ratio test
- the existing basis variable in this row leaves the basis

Pivoting Step

max Z

$$
\begin{array}{rcrcccccl}
13a & + & 23b & & & & & -\, Z & = 0 \\
5a & + & 15b & + s_c & & & & & = 480 \\
4a & + & 4b & & + s_h & & & & = 160 \\
35a & + & 20b & & & + s_m & & & = 1190 \\
a & , & b & , s_c & , s_h & , s_m & & & \geq 0
\end{array}
$$

basis $= \{s_c, s_h, s_m\}$
$a = b = 0$
$Z = 0$

$s_c = 480$
$s_h = 160$
$s_m = 1190$

- choose variable to bring into the basis
- chosen variable should have positive coefficient in objective function
- apply min-ratio test to find out by how much the variable can be increased
- pivot on row found by min-ratio test
- the existing basis variable in this row leaves the basis

max Z

$$
\begin{aligned}
13a + 23b \quad & - Z = 0 \\
5a + 15b + s_c \quad & = 480 \\
4a + 4b + s_h \quad & = 160 \\
35a + 20b + s_m \quad & = 1190 \\
a, b, s_c, s_h, s_m \quad & \geq 0
\end{aligned}
$$

basis = $\{s_c, s_h, s_m\}$

$a = b = 0$

$Z = 0$

$s_c = 480$

$s_h = 160$

$s_m = 1190$

max Z

$$
\begin{aligned}
13a + 23b \quad & - Z = 0 \\
5a + 15b + s_c \quad & = 480 \\
4a + 4b \quad + s_h \quad & = 160 \\
35a + 20b \quad + s_m \quad & = 1190 \\
a, \; b, \; s_c, \; s_h, \; s_m \quad & \geq 0
\end{aligned}
$$

basis = $\{s_c, s_h, s_m\}$

$a = b = 0$

$Z = 0$

$s_c = 480$

$s_h = 160$

$s_m = 1190$

- Choose variable with coefficient ≥ 0 as entering variable.

$$\begin{array}{l} \max Z \\ \quad 13a + 23\mathbf{b} \qquad\qquad\qquad - Z = 0 \\ \quad 5a + 15\mathbf{b} + s_c \qquad\qquad\quad = 480 \\ \quad 4a + 4\mathbf{b} \qquad + s_h \qquad\quad = 160 \\ \quad 35a + 20\mathbf{b} \qquad\qquad + s_m \quad = 1190 \\ \quad a, \mathbf{b}, s_c, s_h, s_m \geq 0 \end{array}$$

$$\begin{array}{l} \text{basis} = \{s_c, s_h, s_m\} \\ a = b = 0 \\ Z = 0 \\ s_c = 480 \\ s_h = 160 \\ s_m = 1190 \end{array}$$

- Choose variable with coefficient ≥ 0 as entering variable.
- If we keep $a = 0$ and increase b from 0 to $\theta > 0$ s.t. all constraints ($Ax = b, x \geq 0$) are still fulfilled the objective value Z will strictly increase.

$$
\begin{array}{l}
\max Z \\
\quad 13a + 23\mathbf{b} \qquad\qquad\qquad\qquad - Z = 0 \\
\quad 5a + 15\mathbf{b} + s_c \qquad\qquad\qquad = 480 \\
\quad 4a + 4\mathbf{b} \qquad + s_h \qquad\qquad = 160 \\
\quad 35a + 20\mathbf{b} \qquad\qquad + s_m \qquad = 1190 \\
\quad a \;,\; \mathbf{b} \;,\; s_c \;,\; s_h \;,\; s_m \qquad \geq 0
\end{array}
$$

$$
\begin{array}{l}
\text{basis} = \{s_c, s_h, s_m\} \\
a = b = 0 \\
Z = 0 \\
s_c = 480 \\
s_h = 160 \\
s_m = 1190
\end{array}
$$

- Choose variable with coefficient ≥ 0 as entering variable.
- If we keep $a = 0$ and increase b from 0 to $\theta > 0$ s.t. all constraints ($Ax = b, x \geq 0$) are still fulfilled the objective value Z will strictly increase.
- For maintaining $Ax = b$ we need e.g. to set $s_c = 480 - 15\theta$.

$$\begin{array}{l}\max Z\\ 13a + 23b \qquad\qquad\qquad - Z = 0\\ 5a + 15\mathbf{b} + s_c \qquad\qquad = 480\\ 4a + 4b \quad + s_h \qquad = 160\\ 35a + 20b \qquad + s_m \quad = 1190\\ a,\ b,\ s_c,\ s_h,\ s_m \geq 0\end{array}$$

$$\begin{array}{l}\text{basis} = \{s_c, s_h, s_m\}\\ a = b = 0\\ Z = 0\\ s_c = 480\\ s_h = 160\\ s_m = 1190\end{array}$$

- Choose variable with coefficient ≥ 0 as entering variable.
- If we keep $a = 0$ and increase b from 0 to $\theta > 0$ s.t. all constraints ($Ax = b, x \geq 0$) are still fulfilled the objective value Z will strictly increase.
- For maintaining $Ax = b$ we need e.g. to set $s_c = 480 - 15\theta$.
- Choosing $\theta = \min\{480/15, 160/4, 1190/20\}$ ensures that in the new solution one current basic variable becomes 0, and no variable goes negative.

$$\begin{array}{l} \max Z \\ \quad 13a + 23b \qquad\qquad\qquad\qquad - Z = 0 \\ \quad 5a + 15\mathbf{b} + s_c \qquad\qquad\qquad = 480 \\ \quad 4a + 4b \qquad + s_h \qquad\qquad = 160 \\ \quad 35a + 20b \qquad\qquad + s_m \qquad = 1190 \\ \quad a, \quad b, \quad s_c, \quad s_h, \quad s_m \qquad \geq 0 \end{array}$$

$$\begin{array}{l} \text{basis} = \{s_c, s_h, s_m\} \\ a = b = 0 \\ Z = 0 \\ s_c = 480 \\ s_h = 160 \\ s_m = 1190 \end{array}$$

- Choose variable with coefficient ≥ 0 as entering variable.
- If we keep $a = 0$ and increase b from 0 to $\theta > 0$ s.t. all constraints ($Ax = b, x \geq 0$) are still fulfilled the objective value Z will strictly increase.
- For maintaining $Ax = b$ we need e.g. to set $s_c = 480 - 15\theta$.
- Choosing $\theta = \min\{480/15, 160/4, 1190/20\}$ ensures that in the new solution one current basic variable becomes 0, and no variable goes negative.
- The basic variable in the row that gives $\min\{480/15, 160/4, 1190/20\}$ becomes the leaving variable.

max Z

$$\begin{array}{rcrcrcrcrcrcl}
13a & + & 23b & & & & & & & - & Z & = & 0 \\
5a & + & 15\mathbf{b} & + & s_c & & & & & & & = & 480 \\
4a & + & 4b & & & + & s_h & & & & & = & 160 \\
35a & + & 20b & & & & & + & s_m & & & = & 1190 \\
a & , & b & , & s_c & , & s_h & , & s_m & & & \geq & 0
\end{array}$$

basis = $\{s_c, s_h, s_m\}$

$a = b = 0$

$Z = 0$

$s_c = 480$

$s_h = 160$

$s_m = 1190$

Substitute $b = \frac{1}{15}(480 - 5a - s_c)$.

max Z

$$\begin{array}{rcrcrcrcrcrcl}
\frac{16}{3}a & & & - & \frac{23}{15}s_c & & & & & - & Z & = & -736 \\
\frac{1}{3}a & + & b & + & \frac{1}{15}s_c & & & & & & & = & 32 \\
\frac{8}{3}a & & & - & \frac{4}{15}s_c & + & s_h & & & & & = & 32 \\
\frac{85}{3}a & & & - & \frac{4}{3}s_c & & & + & s_m & & & = & 550 \\
a & , & b & , & s_c & , & s_h & , & s_m & & & \geq & 0
\end{array}$$

basis = $\{b, s_h, s_m\}$

$a = s_c = 0$

$Z = 736$

$b = 32$

$s_h = 32$

$s_m = 550$

$$\begin{array}{l} \max Z \\ 13a + 23b \qquad\qquad - Z = 0 \\ 5a + 15\mathbf{b} + s_c = 480 \\ 4a + 4b + s_h = 160 \\ 35a + 20b + s_m = 1190 \\ a,\ b,\ s_c,\ s_h,\ s_m \geq 0 \end{array}$$

basis = $\{s_c, s_h, s_m\}$
$a = b = 0$
$Z = 0$

$s_c = 480$
$s_h = 160$
$s_m = 1190$

Substitute $b = \frac{1}{15}(480 - 5a - s_c)$.

$$\begin{array}{l} \max Z \\ \frac{16}{3}a - \frac{23}{15}s_c \qquad - Z = -736 \\ \frac{1}{3}a + b + \frac{1}{15}s_c = 32 \\ \frac{8}{3}a - \frac{4}{15}s_c + s_h = 32 \\ \frac{85}{3}a - \frac{4}{3}s_c + s_m = 550 \\ a,\ b,\ s_c,\ s_h,\ s_m \geq 0 \end{array}$$

basis = $\{b, s_h, s_m\}$
$a = s_c = 0$
$Z = 736$

$b = 32$
$s_h = 32$
$s_m = 550$

max Z

$$
\begin{array}{rcrcrcrcrcrcl}
13a & + & 23b & & & & & & & - & Z & = & 0 \\
5a & + & 15\mathbf{b} & + & s_c & & & & & & & = & 480 \\
4a & + & 4b & & & + & s_h & & & & & = & 160 \\
35a & + & 20b & & & & & + & s_m & & & = & 1190 \\
a & , & b & , & s_c & , & s_h & , & s_m & & & \geq & 0
\end{array}
$$

basis $= \{s_c, s_h, s_m\}$
$a = b = 0$
$Z = 0$

$s_c = 480$
$s_h = 160$
$s_m = 1190$

Substitute $b = \frac{1}{15}(480 - 5a - s_c)$.

max Z

$$
\begin{array}{rcrcrcrcrcrcl}
\frac{16}{3}a & & & - & \frac{23}{15}s_c & & & & & - & Z & = & -736 \\
\frac{1}{3}a & + & b & + & \frac{1}{15}s_c & & & & & & & = & 32 \\
\frac{8}{3}a & & & - & \frac{4}{15}s_c & + & s_h & & & & & = & 32 \\
\frac{85}{3}a & & & - & \frac{4}{3}s_c & & & + & s_m & & & = & 550 \\
a & , & b & , & s_c & , & s_h & , & s_m & & & \geq & 0
\end{array}
$$

basis $= \{b, s_h, s_m\}$
$a = s_c = 0$
$Z = 736$

$b = 32$
$s_h = 32$
$s_m = 550$

max Z

$$
\begin{aligned}
\tfrac{16}{3}a \quad &- \tfrac{23}{15}s_c \qquad\qquad\qquad - Z = -736 \\
\tfrac{1}{3}a + b &+ \tfrac{1}{15}s_c \qquad\qquad\qquad\qquad = 32 \\
\tfrac{8}{3}a \quad &- \tfrac{4}{15}s_c + s_h \qquad\qquad\quad = 32 \\
\tfrac{85}{3}a \quad &- \tfrac{4}{3}s_c \qquad + s_m \qquad = 550 \\
a \;,\; b \;,\; &\quad s_c \;,\; s_h \;,\; s_m \qquad \geq 0
\end{aligned}
$$

basis = $\{b, s_h, s_m\}$

$a = s_c = 0$

$Z = 736$

$b = 32$

$s_h = 32$

$s_m = 550$

max Z

$$
\begin{aligned}
\tfrac{16}{3}a \qquad - \tfrac{23}{15}s_c \qquad\qquad - Z &= -736 \\
\tfrac{1}{3}a + b + \tfrac{1}{15}s_c \qquad\qquad &= 32 \\
\tfrac{8}{3}a \qquad - \tfrac{4}{15}s_c + s_h \qquad &= 32 \\
\tfrac{85}{3}a \qquad - \tfrac{4}{3}s_c \qquad + s_m &= 550 \\
a\ ,\ b\ ,\ s_c\ ,\ s_h\ ,\ s_m &\geq 0
\end{aligned}
$$

basis = $\{b, s_h, s_m\}$

$a = s_c = 0$

$Z = 736$

$b = 32$

$s_h = 32$

$s_m = 550$

Choose variable a to bring into basis.

max Z

$$\begin{array}{rcl}
\frac{16}{3}a - \frac{23}{15}s_c - Z &=& -736 \\
\frac{1}{3}a + b + \frac{1}{15}s_c &=& 32 \\
\frac{8}{3}a - \frac{4}{15}s_c + s_h &=& 32 \\
\frac{85}{3}a - \frac{4}{3}s_c + s_m &=& 550 \\
a\ ,\ b\ ,\ s_c\ ,\ s_h\ ,\ s_m &\geq& 0
\end{array}$$

basis = $\{b, s_h, s_m\}$
$a = s_c = 0$
$Z = 736$

$b = 32$
$s_h = 32$
$s_m = 550$

Choose variable a to bring into basis.

Computing $\min\{3 \cdot 32, 3\cdot 32/8, 3\cdot 550/85\}$ means pivot on line 2.

$$\max Z$$

$$\begin{aligned}
\tfrac{16}{3}a \quad\quad - \tfrac{23}{15}s_c \quad\quad\quad\quad - Z &= -736\\
\tfrac{1}{3}a + b + \tfrac{1}{15}s_c \quad\quad\quad\quad &= 32\\
\tfrac{8}{3}\mathbf{a} \quad\quad - \tfrac{4}{15}s_c + s_h \quad\quad &= 32\\
\tfrac{85}{3}a \quad\quad - \tfrac{4}{3}s_c \quad\quad + s_m &= 550\\
a\ ,\ b\ ,\ s_c\ ,\ s_h\ ,\ s_m &\geq 0
\end{aligned}$$

basis $= \{b, s_h, s_m\}$

$a = s_c = 0$

$Z = 736$

$b = 32$

$s_h = 32$

$s_m = 550$

Choose variable a to bring into basis.

Computing $\min\{3 \cdot 32, {}^{3\cdot 32}/_8, {}^{3\cdot 550}/_{85}\}$ means pivot on line 2.

Substitute $a = \frac{3}{8}(32 + \frac{4}{15}s_c - s_h)$.

max Z

$$
\begin{array}{rcl}
\frac{16}{3}a - \frac{23}{15}s_c - Z &=& -736 \\
\frac{1}{3}a + b + \frac{1}{15}s_c &=& 32 \\
\frac{8}{3}a - \frac{4}{15}s_c + s_h &=& 32 \\
\frac{85}{3}a - \frac{4}{3}s_c + s_m &=& 550 \\
a, b, s_c, s_h, s_m &\geq& 0
\end{array}
$$

basis = $\{b, s_h, s_m\}$
$a = s_c = 0$
$Z = 736$
$b = 32$
$s_h = 32$
$s_m = 550$

Choose variable a to bring into basis.

Computing $\min\{3 \cdot 32, {}^{3\cdot 32}/_{8}, {}^{3\cdot 550}/_{85}\}$ means pivot on line 2.

Substitute $a = \frac{3}{8}(32 + \frac{4}{15}s_c - s_h)$.

max Z

$$
\begin{array}{rcl}
- s_c - 2s_h - Z &=& -800 \\
b + \frac{1}{10}s_c - \frac{1}{8}s_h &=& 28 \\
a - \frac{1}{10}s_c + \frac{3}{8}s_h &=& 12 \\
\frac{3}{2}s_c - \frac{85}{8}s_h + s_m &=& 210 \\
a, b, s_c, s_h, s_m &\geq& 0
\end{array}
$$

basis = $\{a, b, s_m\}$
$s_c = s_h = 0$
$Z = 800$
$b = 28$
$a = 12$
$s_m = 210$

4 Simplex Algorithm

Pivoting stops when all coefficients in the objective function are non-positive.

Solution is optimal:

- any feasible solution satisfies all equations in the tableaux
- in particular: $Z = 800 - s_c - 2s_h$, $s_c \geq 0$, $s_h \geq 0$
- hence optimum solution value is at most 800
- the current solution has value 800

4 Simplex Algorithm

Pivoting stops when all coefficients in the objective function are non-positive.

Solution is optimal:

- any feasible solution satisfies all equations in the tableaux
- in particular: $Z = 800 - s_c - 2s_h$, $s_c \geq 0$, $s_h \geq 0$
- hence optimum solution value is at most 800
- the current solution has value 800

4 Simplex Algorithm

Pivoting stops when all coefficients in the objective function are non-positive.

Solution is optimal:

- any feasible solution satisfies all equations in the tableaux
- in particular: $Z = 800 - s_c - 2s_h$, $s_c \geq 0, s_h \geq 0$
- hence optimum solution value is at most 800
- the current solution has value 800

4 Simplex Algorithm

Pivoting stops when all coefficients in the objective function are non-positive.

Solution is optimal:

- any feasible solution satisfies all equations in the tableaux
- in particular: $Z = 800 - s_c - 2s_h$, $s_c \geq 0, s_h \geq 0$
- hence optimum solution value is at most 800
- the current solution has value 800

4 Simplex Algorithm

Pivoting stops when all coefficients in the objective function are non-positive.

Solution is optimal:

- any feasible solution satisfies all equations in the tableaux
- in particular: $Z = 800 - s_c - 2s_h$, $s_c \geq 0, s_h \geq 0$
- hence optimum solution value is at most 800
- the current solution has value 800

4 Simplex Algorithm

Pivoting stops when all coefficients in the objective function are non-positive.

Solution is optimal:

- any feasible solution satisfies all equations in the tableaux
- in particular: $Z = 800 - s_c - 2s_h$, $s_c \geq 0, s_h \geq 0$
- hence optimum solution value is at most 800
- the current solution has value 800

Matrix View

Let our linear program be

$$\begin{array}{rcrcl} c_B^t x_B & + & c_N^t x_N & = & Z \\ A_B x_B & + & A_N x_N & = & b \\ x_B & , & x_N & \geq & 0 \end{array}$$

The simplex tableaux for basis B is

$$\begin{array}{rcrcl} & & (c_N^t - c_B^t A_B^{-1} A_N) x_N & = & Z - c_B^t A_B^{-1} b \\ I x_B & + & A_B^{-1} A_N x_N & = & A_B^{-1} b \\ x_B & , & x_N & \geq & 0 \end{array}$$

The BFS is given by $x_N = 0, x_B = A_B^{-1} b$.

If $(c_N^t - c_B^t A_B^{-1} A_N) \leq 0$ we know that we have an optimum solution.

Matrix View

Let our linear program be

$$\begin{array}{rcrcl} c_B^t x_B & + & c_N^t x_N & = & Z \\ A_B x_B & + & A_N x_N & = & b \\ x_B & , & x_N & \geq & 0 \end{array}$$

The simplex tableaux for basis B is

$$\begin{array}{rcrcl} & & (c_N^t - c_B^t A_B^{-1} A_N) x_N & = & Z - c_B^t A_B^{-1} b \\ I x_B & + & A_B^{-1} A_N x_N & = & A_B^{-1} b \\ x_B & , & x_N & \geq & 0 \end{array}$$

The BFS is given by $x_N = 0, x_B = A_B^{-1} b$.

If $(c_N^t - c_B^t A_B^{-1} A_N) \leq 0$ we know that we have an optimum solution.

Matrix View

Let our linear program be

$$\begin{array}{ccccc} c_B^t x_B & + & c_N^t x_N & = & Z \\ A_B x_B & + & A_N x_N & = & b \\ x_B & , & x_N & \geq & 0 \end{array}$$

The simplex tableaux for basis B is

$$\begin{array}{ccccc} & & (c_N^t - c_B^t A_B^{-1} A_N) x_N & = & Z - c_B^t A_B^{-1} b \\ I x_B & + & A_B^{-1} A_N x_N & = & A_B^{-1} b \\ x_B & , & x_N & \geq & 0 \end{array}$$

The BFS is given by $x_N = 0, x_B = A_B^{-1} b$.

If $(c_N^t - c_B^t A_B^{-1} A_N) \leq 0$ we know that we have an optimum solution.

Matrix View

Let our linear program be

$$\begin{array}{rcrcl} c_B^t x_B & + & c_N^t x_N & = & Z \\ A_B x_B & + & A_N x_N & = & b \\ x_B & , & x_N & \geq & 0 \end{array}$$

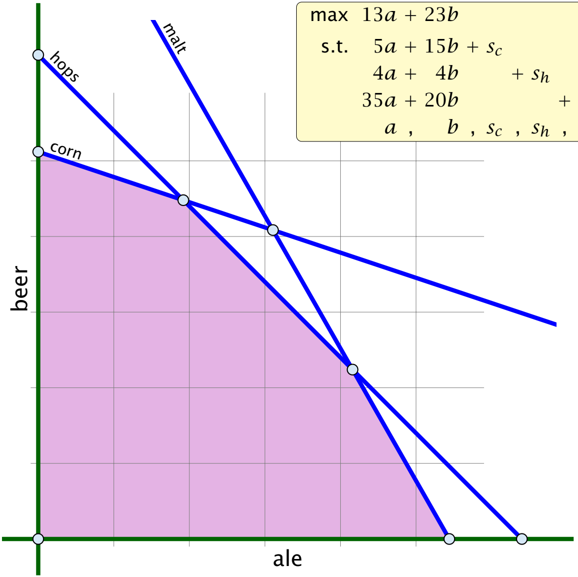
The simplex tableaux for basis B is

$$\begin{array}{rcrcl} & & (c_N^t - c_B^t A_B^{-1} A_N) x_N & = & Z - c_B^t A_B^{-1} b \\ I x_B & + & A_B^{-1} A_N x_N & = & A_B^{-1} b \\ x_B & , & x_N & \geq & 0 \end{array}$$

The BFS is given by $x_N = 0, x_B = A_B^{-1} b$.

If $(c_N^t - c_B^t A_B^{-1} A_N) \leq 0$ we know that we have an optimum solution.

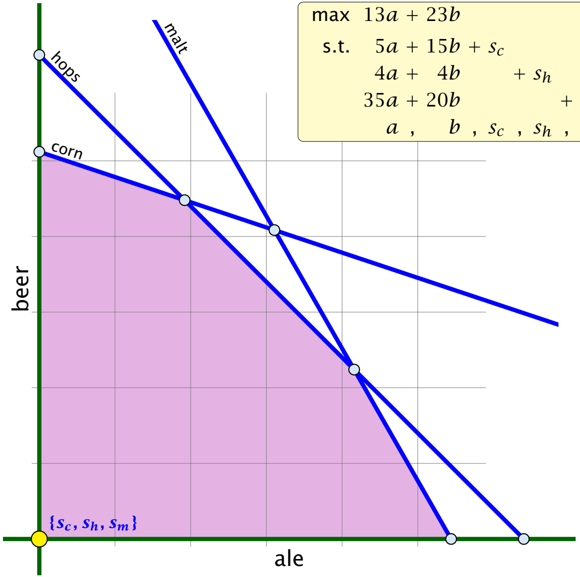
Geometric View of Pivoting



$$\begin{array}{lrcrcccccl}
\text{max} & 13a & + & 23b & & & & & & \\
\text{s.t.} & 5a & + & 15b & + & s_c & & & = & 480 \\
& 4a & + & 4b & & & + s_h & & = & 160 \\
& 35a & + & 20b & & & & + s_m & = & 1190 \\
& a & , & b & , & s_c & , s_h & , s_m & \geq & 0
\end{array}$$

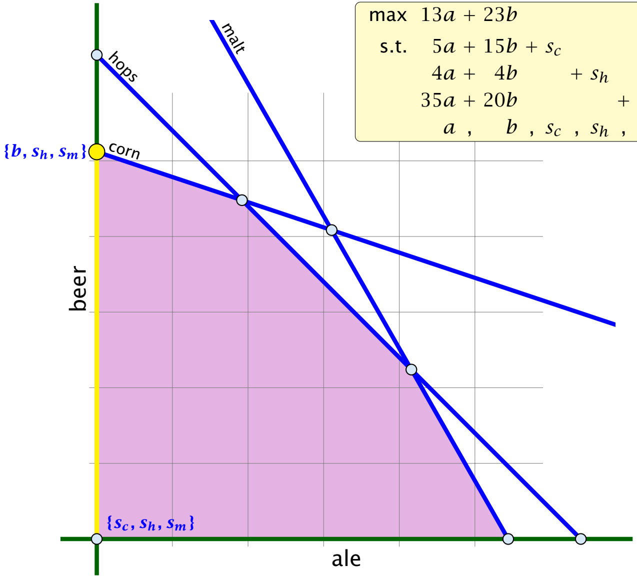
Geometric View of Pivoting

$$\begin{array}{lrcrcccccl}
\max & 13a & + & 23b & & & & & & \\
\text{s.t.} & 5a & + & 15b & + & s_c & & & = & 480 \\
& 4a & + & 4b & & & + & s_h & = & 160 \\
& 35a & + & 20b & & & & + s_m & = & 1190 \\
& a & , & b & , & s_c & , & s_h \;,\; s_m & \geq & 0
\end{array}$$



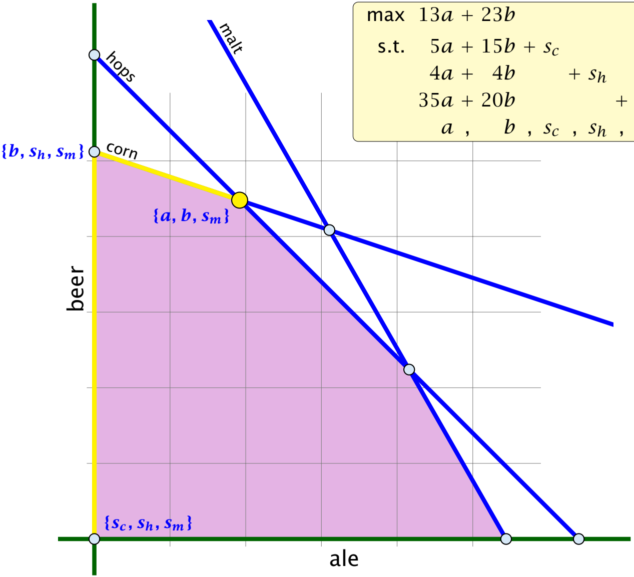
Geometric View of Pivoting

$$
\begin{array}{lrcrcrcrcrcl}
\max & 13a & + & 23b & & & & & & & & \\
\text{s.t.} & 5a & + & 15b & + & s_c & & & & & = & 480 \\
 & 4a & + & 4b & & & + & s_h & & & = & 160 \\
 & 35a & + & 20b & & & & & + & s_m & = & 1190 \\
 & a & , & b & , & s_c & , & s_h & , & s_m & \geq & 0
\end{array}
$$



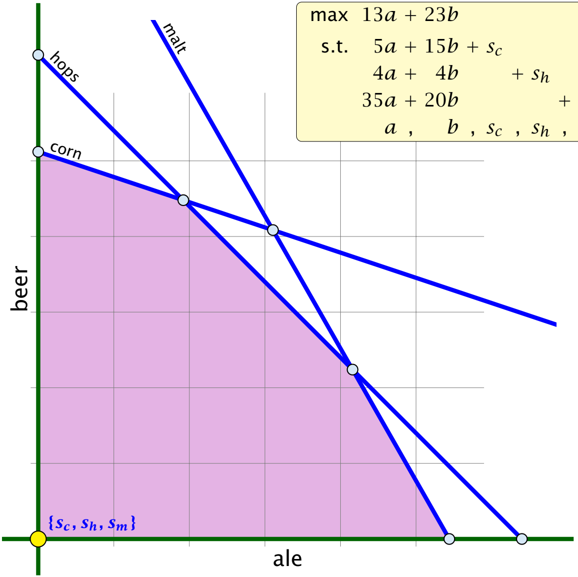
Geometric View of Pivoting

$$
\begin{array}{lrcrcrcrcrcl}
\max & 13a & + & 23b & & & & & & & & \\
\text{s.t.} & 5a & + & 15b & + & s_c & & & & & = & 480 \\
 & 4a & + & 4b & & & + & s_h & & & = & 160 \\
 & 35a & + & 20b & & & & & + & s_m & = & 1190 \\
 & a & , & b & , & s_c & , & s_h & , & s_m & \geq & 0
\end{array}
$$

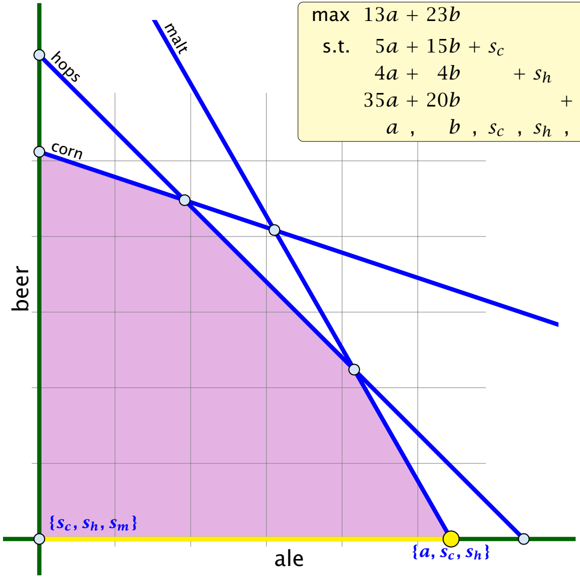


Geometric View of Pivoting

$$\begin{array}{llllllll} \max & 13a & + 23b & & & & & \\ \text{s.t.} & 5a & + 15b & + s_c & & & = & 480 \\ & 4a & + 4b & & + s_h & & = & 160 \\ & 35a & + 20b & & & + s_m & = & 1190 \\ & a & , b & , s_c & , s_h & , s_m & \geq & 0 \end{array}$$



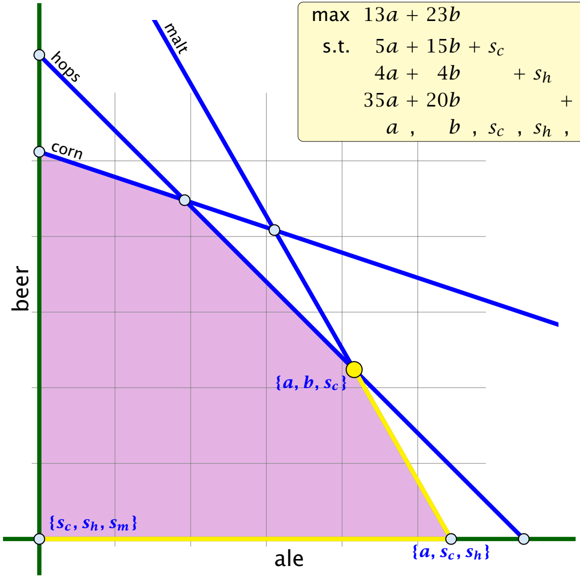
Geometric View of Pivoting



$$\begin{array}{lllllll}
\max & 13a + 23b & & & & & \\
\text{s.t.} & 5a + 15b & + s_c & & & = & 480 \\
& 4a + 4b & & + s_h & & = & 160 \\
& 35a + 20b & & & + s_m & = & 1190 \\
& a \;, \; b & , s_c & , s_h & , s_m & \geq & 0
\end{array}$$

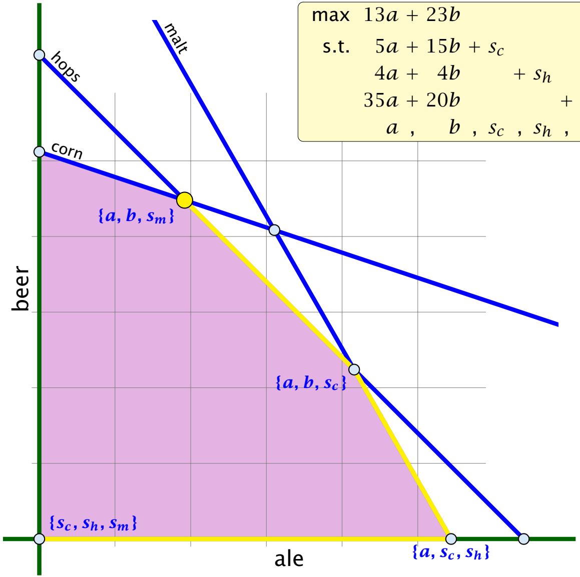
Geometric View of Pivoting

$$\begin{array}{ll} \text{max} & 13a + 23b \\ \text{s.t.} & 5a + 15b + s_c = 480 \\ & 4a + 4b + s_h = 160 \\ & 35a + 20b + s_m = 1190 \\ & a,\ b,\ s_c,\ s_h,\ s_m \geq 0 \end{array}$$



Geometric View of Pivoting

$$
\begin{array}{lrcrcrcrcrcl}
\max & 13a & + & 23b & & & & & & & & \\
\text{s.t.} & 5a & + & 15b & + & s_c & & & & & = & 480 \\
 & 4a & + & 4b & & & + & s_h & & & = & 160 \\
 & 35a & + & 20b & & & & & + & s_m & = & 1190 \\
 & a & , & b & , & s_c & , & s_h & , & s_m & \geq & 0
\end{array}
$$



Algebraic Definition of Pivoting

- Given basis B with BFS x^*.
- Choose index $j \notin B$ in order to increase x_j^* from 0 to $\theta > 0$.
  - Other non-basis variables should stay at 0.
  - Basis variables change to maintain feasibility.
- Go from x^* to $x^* + \theta \cdot d$.

Requirements for d:

- $d_j = 1$ (normalization)
- $d_\ell = 0$, $\ell \notin B$, $\ell \neq j$
- $A(x^* + \theta d) = b$ must hold. Hence $Ad = 0$.
- Altogether: $A_B d_B + A_{*j} = Ad = 0$, which gives $d_B = -A_B^{-1} A_{*j}$.

Algebraic Definition of Pivoting

- Given basis B with BFS x^*.
- Choose index $j \notin B$ in order to increase x_j^* from 0 to $\theta > 0$.
  - Other non-basis variables should stay at 0.
  - Basis variables change to maintain feasibility.
- Go from x^* to $x^* + \theta \cdot d$.

Requirements for d:

- $d_j = 1$ (normalization)
- $d_\ell = 0$, $\ell \notin B$, $\ell \neq j$
- $A(x^* + \theta d) = b$ must hold. Hence $Ad = 0$.
- Altogether: $A_B d_B + A_{*j} = Ad = 0$, which gives $d_B = -A_B^{-1} A_{*j}$.

Algebraic Definition of Pivoting

- Given basis B with BFS x^*.
- Choose index $j \notin B$ in order to increase x_j^* from 0 to $\theta > 0$.
  - Other non-basis variables should stay at 0.
  - Basis variables change to maintain feasibility.
- Go from x^* to $x^* + \theta \cdot d$.

Requirements for d:

Algebraic Definition of Pivoting

- Given basis B with BFS x^*.
- Choose index $j \notin B$ in order to increase x_j^* from 0 to $\theta > 0$.
  - Other non-basis variables should stay at 0.
  - Basis variables change to maintain feasibility.
- Go from x^* to $x^* + \theta \cdot d$.

Requirements for d:

Algebraic Definition of Pivoting

- Given basis B with BFS x^*.
- Choose index $j \notin B$ in order to increase x_j^* from 0 to $\theta > 0$.
  - Other non-basis variables should stay at 0.
  - Basis variables change to maintain feasibility.
- Go from x^* to $x^* + \theta \cdot d$.

Algebraic Definition of Pivoting

- Given basis B with BFS x^*.
- Choose index $j \notin B$ in order to increase x_j^* from 0 to $\theta > 0$.
  - Other non-basis variables should stay at 0.
  - Basis variables change to maintain feasibility.
- Go from x^* to $x^* + \theta \cdot d$.

Requirements for d:

- $d_j = 1$ (normalization)
- $d_\ell = 0$, $\ell \notin B$, $\ell \neq j$
- $A(x^* + \theta d) = b$ must hold. Hence $Ad = 0$.
- Altogether: $A_B d_B + A_{*j} = Ad = 0$, which gives $d_B = -A_B^{-1} A_{*j}$.

Algebraic Definition of Pivoting

- Given basis B with BFS x^*.
- Choose index $j \notin B$ in order to increase x_j^* from 0 to $\theta > 0$.
  - Other non-basis variables should stay at 0.
  - Basis variables change to maintain feasibility.
- Go from x^* to $x^* + \theta \cdot d$.

Requirements for d:

- $d_j = 1$ (normalization)
- $d_\ell = 0$, $\ell \notin B$, $\ell \neq j$
- $A(x^* + \theta d) = b$ must hold. Hence $Ad = 0$.
- Altogether: $A_B d_B + A_{*j} = Ad = 0$, which gives $d_B = -A_B^{-1} A_{*j}$.

Algebraic Definition of Pivoting

- Given basis B with BFS x^*.
- Choose index $j \notin B$ in order to increase x_j^* from 0 to $\theta > 0$.
  - Other non-basis variables should stay at 0.
  - Basis variables change to maintain feasibility.
- Go from x^* to $x^* + \theta \cdot d$.

Requirements for d:

- $d_j = 1$ (normalization)
- $d_\ell = 0$, $\ell \notin B$, $\ell \neq j$
- $A(x^* + \theta d) = b$ must hold. Hence $Ad = 0$.
- Altogether: $A_B d_B + A_{*j} = Ad = 0$, which gives $d_B = -A_B^{-1} A_{*j}$.

Algebraic Definition of Pivoting

- Given basis B with BFS x^*.
- Choose index $j \notin B$ in order to increase x_j^* from 0 to $\theta > 0$.
  - Other non-basis variables should stay at 0.
  - Basis variables change to maintain feasibility.
- Go from x^* to $x^* + \theta \cdot d$.

Requirements for d:

- $d_j = 1$ (normalization)
- $d_\ell = 0$, $\ell \notin B$, $\ell \neq j$
- $A(x^* + \theta d) = b$ must hold. Hence $Ad = 0$.
- Altogether: $A_B d_B + A_{*j} = Ad = 0$, which gives $d_B = -A_B^{-1} A_{*j}$.

Algebraic Definition of Pivoting

Definition 11 (j-th basis direction)

Let B be a basis, and let $j \notin B$. The vector d with $d_j = 1$ and $d_\ell = 0, \ell \notin B, \ell \neq j$ and $d_B = -A_B^{-1}A_{*j}$ is called the j-th basis direction for B.

Going from x^* to $x^* + \theta \cdot d$ the objective function changes by

$$\theta \cdot c^t d = \theta(c_j - c_B^t A_B^{-1} A_{*j})$$

Algebraic Definition of Pivoting

Definition 11 (j-th basis direction)

Let B be a basis, and let $j \notin B$. The vector d with $d_j = 1$ and $d_\ell = 0, \ell \notin B, \ell \neq j$ and $d_B = -A_B^{-1}A_{*j}$ is called the j-th basis direction for B.

Going from x^* to $x^* + \theta \cdot d$ the objective function changes by

$$\theta \cdot c^t d = \theta(c_j - c_B^t A_B^{-1} A_{*j})$$

Algebraic Definition of Pivoting

Definition 12 (Reduced Cost)

For a basis B the value

$$\tilde{c}_j = c_j - c_B^t A_B^{-1} A_{*j}$$

is called the reduced cost for variable x_j.

Note that this is defined for every j. If $j \in B$ then the above term is 0.

Algebraic Definition of Pivoting

Let our linear program be

$$\begin{array}{rcrcl} c_B^t x_B & + & c_N^t x_N & = & Z \\ A_B x_B & + & A_N x_N & = & b \\ x_B & , & x_N & \geq & 0 \end{array}$$

The simplex tableaux for basis B is

$$\begin{array}{rcrcl} & & (c_N^t - c_B^t A_B^{-1} A_N) x_N & = & Z - c_B^t A_B^{-1} b \\ I x_B & + & A_B^{-1} A_N x_N & = & A_B^{-1} b \\ x_B & , & x_N & \geq & 0 \end{array}$$

The BFS is given by $x_N = 0, x_B = A_B^{-1} b$.

If $(c_N^t - c_B^t A_B^{-1} A_N) \leq 0$ we know that we have an optimum solution.

Algebraic Definition of Pivoting

Let our linear program be

$$\begin{array}{ccccc} c_B^t x_B & + & c_N^t x_N & = & Z \\ A_B x_B & + & A_N x_N & = & b \\ x_B & , & x_N & \geq & 0 \end{array}$$

The simplex tableaux for basis B is

$$\begin{array}{ccccc} & & (c_N^t - c_B^t A_B^{-1} A_N) x_N & = & Z - c_B^t A_B^{-1} b \\ I x_B & + & A_B^{-1} A_N x_N & = & A_B^{-1} b \\ x_B & , & x_N & \geq & 0 \end{array}$$

The BFS is given by $x_N = 0, x_B = A_B^{-1} b$.

If $(c_N^t - c_B^t A_B^{-1} A_N) \leq 0$ we know that we have an optimum solution.

Algebraic Definition of Pivoting

Let our linear program be

$$\begin{array}{rcrcl} c_B^t x_B & + & c_N^t x_N & = & Z \\ A_B x_B & + & A_N x_N & = & b \\ x_B & , & x_N & \geq & 0 \end{array}$$

The simplex tableaux for basis B is

$$\begin{array}{rcrcl} & & (c_N^t - c_B^t A_B^{-1} A_N) x_N & = & Z - c_B^t A_B^{-1} b \\ I x_B & + & A_B^{-1} A_N x_N & = & A_B^{-1} b \\ x_B & , & x_N & \geq & 0 \end{array}$$

The BFS is given by $x_N = 0, x_B = A_B^{-1} b$.

If $(c_N^t - c_B^t A_B^{-1} A_N) \leq 0$ we know that we have an optimum solution.

Algebraic Definition of Pivoting

Let our linear program be

$$\begin{array}{rcrcl} c_B^t x_B & + & c_N^t x_N & = & Z \\ A_B x_B & + & A_N x_N & = & b \\ x_B & , & x_N & \geq & 0 \end{array}$$

The simplex tableaux for basis B is

$$\begin{array}{rcrcl} & & (c_N^t - c_B^t A_B^{-1} A_N) x_N & = & Z - c_B^t A_B^{-1} b \\ I x_B & + & A_B^{-1} A_N x_N & = & A_B^{-1} b \\ x_B & , & x_N & \geq & 0 \end{array}$$

The BFS is given by $x_N = 0, x_B = A_B^{-1} b$.

If $(c_N^t - c_B^t A_B^{-1} A_N) \leq 0$ we know that we have an optimum solution.

4 Simplex Algorithm

Questions:

Questions:

- What happens if the min ratio test fails to give us a value θ by which we can safely increase the entering variable?
- How do we find the initial basic feasible solution?
- Is there always a basis B such that

$$(c_N^t - c_B^t A_B^{-1} A_N) \leq 0 \ ?$$

  Then we can terminate because we know that the solution is optimal.
- If yes how do we make sure that we reach such a basis?

4 Simplex Algorithm

Questions:

- What happens if the min ratio test fails to give us a value θ by which we can safely increase the entering variable?
- How do we find the initial basic feasible solution?
- Is there always a basis B such that

$$(c_N^t - c_B^t A_B^{-1} A_N) \leq 0 \ ?$$

  Then we can terminate because we know that the solution is optimal.
- If yes how do we make sure that we reach such a basis?

4 Simplex Algorithm

Questions:

- What happens if the min ratio test fails to give us a value θ by which we can safely increase the entering variable?
- How do we find the initial basic feasible solution?
- Is there always a basis B such that

$$(c_N^t - c_B^t A_B^{-1} A_N) \leq 0 \quad ?$$

  Then we can terminate because we know that the solution is optimal.
- If yes how do we make sure that we reach such a basis?

4 Simplex Algorithm

Questions:

- What happens if the min ratio test fails to give us a value θ by which we can safely increase the entering variable?
- How do we find the initial basic feasible solution?
- Is there always a basis B such that

$$(c_N^t - c_B^t A_B^{-1} A_N) \leq 0 \ \ ?$$

  Then we can terminate because we know that the solution is optimal.
- If yes how do we make sure that we reach such a basis?

Min Ratio Test

The min ratio test computes a value $\theta \geq 0$ such that after setting the entering variable to θ the leaving variable becomes 0 and all other variables stay non-negative.

For this one computes b_i/A_{ie} for all constraints i and calculates the minimum positive value.

What does it mean that the ratio b_i/A_{ie} (and hence A_{ie}) is negative for a constraint?

This means that the corresponding basic variable will increase if we increase b. Hence, there is no danger of this basic variable becoming negative

What happens if **all** b_i/A_{ie} are negative? Then we do not have a leaving variable. Then the LP is unbounded!

Min Ratio Test

The min ratio test computes a value $\theta \geq 0$ such that after setting the entering variable to θ the leaving variable becomes 0 and all other variables stay non-negative.

For this one computes b_i/A_{ie} for all constraints i and calculates the minimum positive value.

What does it mean that the ratio b_i/A_{ie} (and hence A_{ie}) is negative for a constraint?

This means that the corresponding basic variable will increase if we increase b. Hence, there is no danger of this basic variable becoming negative

What happens if **all** b_i/A_{ie} are negative? Then we do not have a leaving variable. Then the LP is unbounded!

Min Ratio Test

The min ratio test computes a value $\theta \geq 0$ such that after setting the entering variable to θ the leaving variable becomes 0 and all other variables stay non-negative.

For this one computes b_i/A_{ie} for all constraints i and calculates the minimum positive value.

What does it mean that the ratio b_i/A_{ie} (and hence A_{ie}) is negative for a constraint?

This means that the corresponding basic variable will increase if we increase b. Hence, there is no danger of this basic variable becoming negative

What happens if **all** b_i/A_{ie} are negative? Then we do not have a leaving variable. Then the LP is unbounded!

Min Ratio Test

The min ratio test computes a value $\theta \geq 0$ such that after setting the entering variable to θ the leaving variable becomes 0 and all other variables stay non-negative.

For this one computes b_i/A_{ie} for all constraints i and calculates the minimum positive value.

What does it mean that the ratio b_i/A_{ie} (and hence A_{ie}) is negative for a constraint?

This means that the corresponding basic variable will increase if we increase b. Hence, there is no danger of this basic variable becoming negative

What happens if **all** b_i/A_{ie} are negative? Then we do not have a leaving variable. Then the LP is unbounded!

Min Ratio Test

The min ratio test computes a value $\theta \geq 0$ such that after setting the entering variable to θ the leaving variable becomes 0 and all other variables stay non-negative.

For this one computes b_i/A_{ie} for all constraints i and calculates the minimum positive value.

What does it mean that the ratio b_i/A_{ie} (and hence A_{ie}) is negative for a constraint?

This means that the corresponding basic variable will increase if we increase b. Hence, there is no danger of this basic variable becoming negative

What happens if **all** b_i/A_{ie} are negative? Then we do not have a leaving variable. Then the LP is unbounded!

Min Ratio Test

The min ratio test computes a value $\theta \geq 0$ such that after setting the entering variable to θ the leaving variable becomes 0 and all other variables stay non-negative.

For this one computes b_i/A_{ie} for all constraints i and calculates the minimum positive value.

What does it mean that the ratio b_i/A_{ie} (and hence A_{ie}) is negative for a constraint?

This means that the corresponding basic variable will increase if we increase b. Hence, there is no danger of this basic variable becoming negative

What happens if **all** b_i/A_{ie} are negative? Then we do not have a leaving variable. Then the LP is unbounded!

Termination

The objective function does not decrease during one iteration of the simplex-algorithm.

Does it always increase?

Termination

The objective function does not decrease during one iteration of the simplex-algorithm.

Does it always increase?

Termination

The objective function does not decrease during one iteration of the simplex-algorithm.

Does it always increase?

Termination

The objective function may not increase!

Because a variable x_ℓ with $\ell \in B$ is already 0.

The set of inequalities is degenerate (also the basis is degenerate).

Definition 13 (Degeneracy)

A BFS x^* is called degenerate if the set $J = \{j \mid x_j^* > 0\}$ fulfills $|J| < m$.

It is possible that the algorithm cycles, i.e., it cycles through a sequence of different bases without ever terminating. Happens, very rarely in practise.

Termination

The objective function may not increase!

Because a variable x_ℓ with $\ell \in B$ is already 0.

The set of inequalities is degenerate (also the basis is degenerate).

Definition 13 (Degeneracy)

A BFS x^* is called degenerate if the set $J = \{j \mid x_j^* > 0\}$ fulfills $|J| < m$.

It is possible that the algorithm cycles, i.e., it cycles through a sequence of different bases without ever terminating. Happens, very rarely in practise.

Termination

The objective function may not increase!

Because a variable x_ℓ with $\ell \in B$ is already 0.

The set of inequalities is degenerate (also the basis is degenerate).

Definition 13 (Degeneracy)

A BFS x^* is called degenerate if the set $J = \{j \mid x_j^* > 0\}$ fulfills $|J| < m$.

It is possible that the algorithm cycles, i.e., it cycles through a sequence of different bases without ever terminating. Happens, very rarely in practise.

Termination

The objective function may not increase!

Because a variable x_ℓ with $\ell \in B$ is already 0.

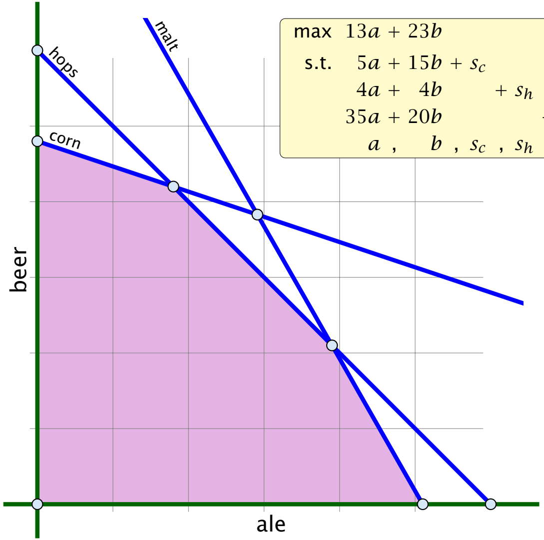
The set of inequalities is degenerate (also the basis is degenerate).

Definition 13 (Degeneracy)

A BFS x^* is called degenerate if the set $J = \{j \mid x_j^* > 0\}$ fulfills $|J| < m$.

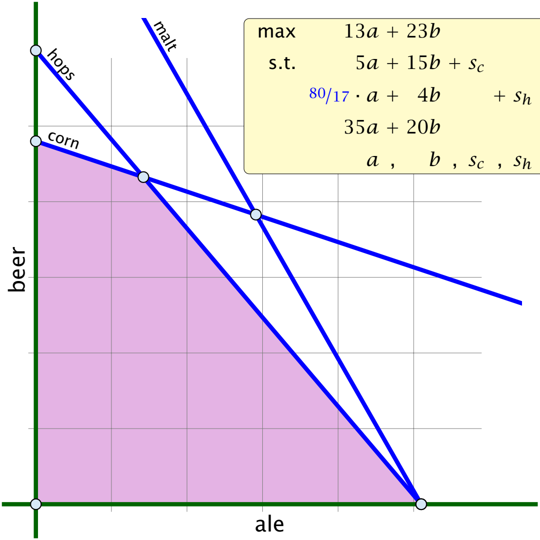
It is possible that the algorithm cycles, i.e., it cycles through a sequence of different bases without ever terminating. Happens, very rarely in practise.

Non Degenerate Example



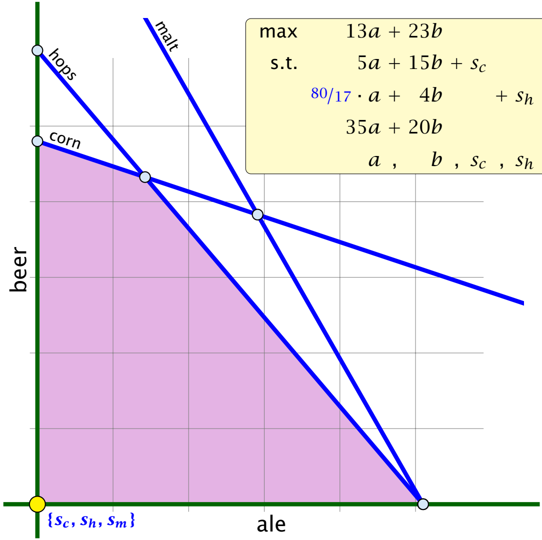
$$
\begin{array}{lrcl}
\max & 13a + 23b & & \\
\text{s.t.} & 5a + 15b + s_c & = & 480 \\
& 4a + 4b + s_h & = & 160 \\
& 35a + 20b + s_m & = & 1190 \\
& a, b, s_c, s_h, s_m & \geq & 0
\end{array}
$$

Degenerate Example



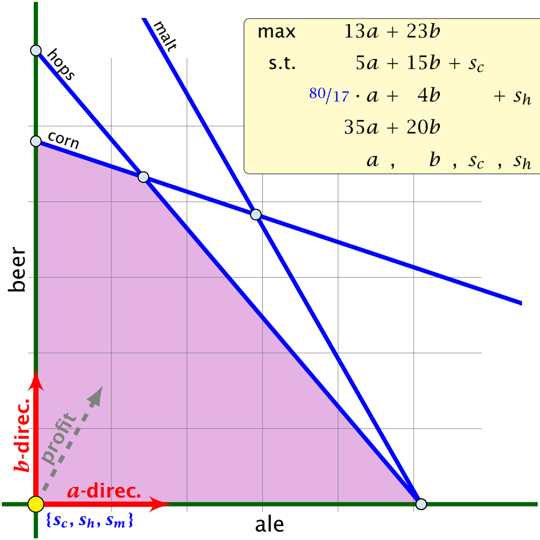
$$
\begin{array}{llcl}
\max & 13a + 23b & & \\
\text{s.t.} & 5a + 15b + s_c & = & 480 \\
 & {}^{80}/_{17} \cdot a + 4b + s_h & = & 160 \\
 & 35a + 20b + s_m & = & 1190 \\
 & a\ ,\ b\ ,\ s_c\ ,\ s_h\ ,\ s_m & \geq & 0
\end{array}
$$

Degenerate Example



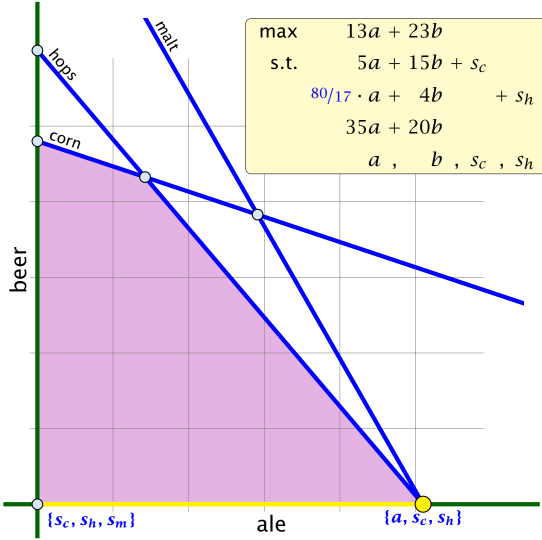
$$
\begin{array}{lrl}
\max & 13a + 23b & \\
\text{s.t.} & 5a + 15b + s_c & = 480 \\
 & \tfrac{80}{17} \cdot a + 4b + s_h & = 160 \\
 & 35a + 20b + s_m & = 1190 \\
 & a, b, s_c, s_h, s_m & \geq 0
\end{array}
$$

Degenerate Example



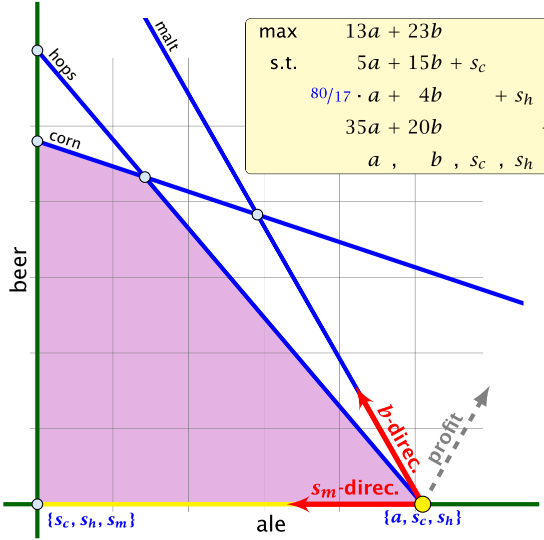
$$
\begin{array}{llllllll}
\text{max} & 13a & + 23b & & & & & \\
\text{s.t.} & 5a & + 15b & + s_c & & & = & 480 \\
 & 80/17 \cdot a & + \ 4b & & + s_h & & = & 160 \\
 & 35a & + 20b & & & + s_m & = & 1190 \\
 & a\ , & b\ , & s_c\ , & s_h\ , & s_m & \geq & 0
\end{array}
$$

Degenerate Example



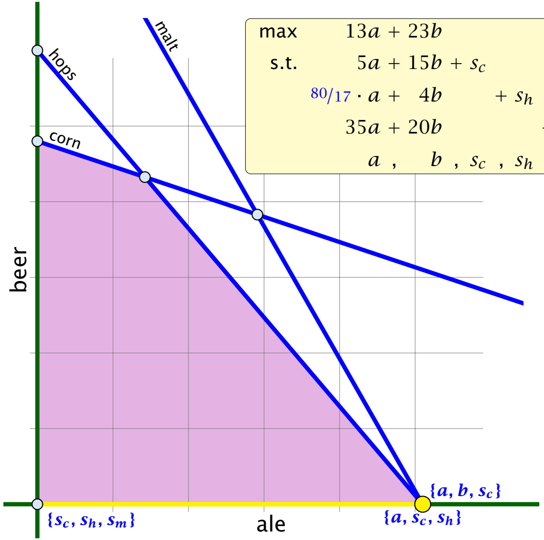
$$
\begin{array}{lrcl}
\max & 13a + 23b & & \\
\text{s.t.} & 5a + 15b + s_c & = & 480 \\
 & 80/17 \cdot a + 4b + s_h & = & 160 \\
 & 35a + 20b + s_m & = & 1190 \\
 & a\ ,\ b\ ,\ s_c\ ,\ s_h\ ,\ s_m & \geq & 0
\end{array}
$$

Degenerate Example



$$
\begin{array}{llcl}
\text{max} & 13a + 23b & & \\
\text{s.t.} & 5a + 15b + s_c & = & 480 \\
 & {}^{80}/_{17} \cdot a + 4b + s_h & = & 160 \\
 & 35a + 20b + s_m & = & 1190 \\
 & a \ ,\ b \ ,\ s_c \ ,\ s_h \ ,\ s_m & \geq & 0
\end{array}
$$

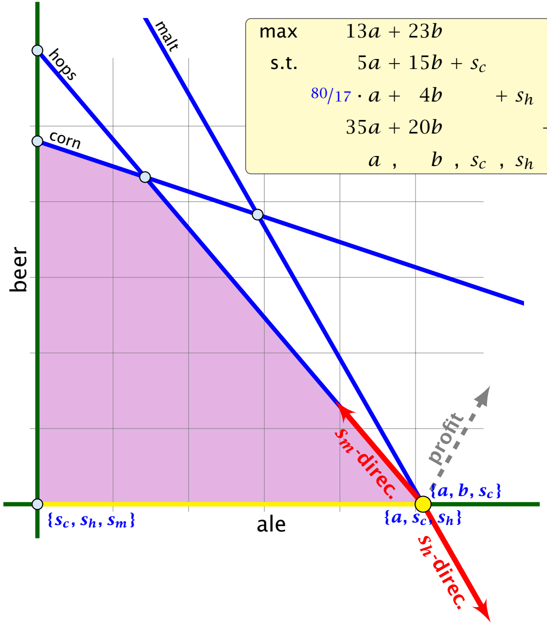
Degenerate Example



$$
\begin{array}{lrcrcrcrcrcl}
\text{max} & 13a & + & 23b & & & & & & & & \\
\text{s.t.} & 5a & + & 15b & + & s_c & & & & & = & 480 \\
 & 80/17 \cdot a & + & 4b & & & + & s_h & & & = & 160 \\
 & 35a & + & 20b & & & & & + & s_m & = & 1190 \\
 & a & , & b & , & s_c & , & s_h & , & s_m & \geq & 0
\end{array}
$$

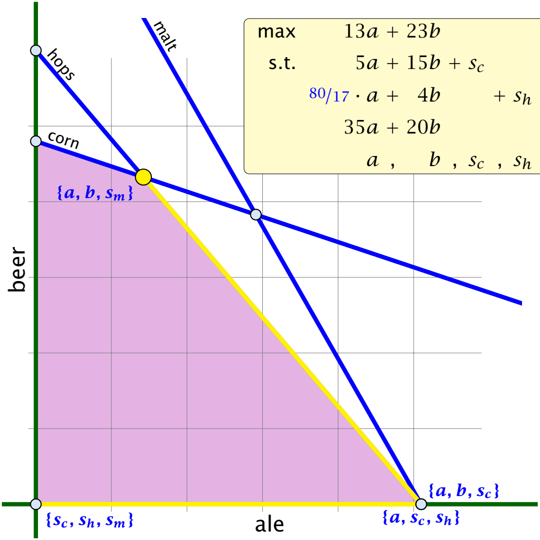
Degenerate Example

$$
\begin{array}{llllllll}
\max & 13a & + & 23b & & & & \\
\text{s.t.} & 5a & + & 15b & + s_c & & & = 480 \\
 & 80/17 \cdot a & + & 4b & & + s_h & & = 160 \\
 & 35a & + & 20b & & & + s_m & = 1190 \\
 & a & , & b & , s_c & , s_h & , s_m & \geq 0
\end{array}
$$



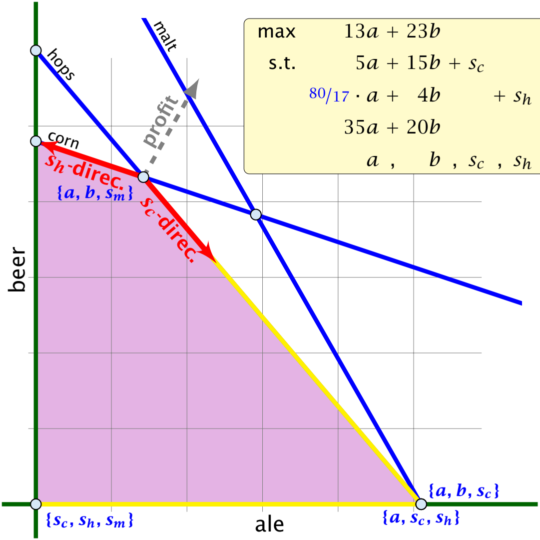
Degenerate Example

$$
\begin{array}{llllllll}
\text{max} & 13a & + 23b & & & & & \\
\text{s.t.} & 5a & + 15b & + s_c & & & = 480 \\
& \tfrac{80}{17} \cdot a & + \ \ 4b & & + s_h & & = 160 \\
& 35a & + 20b & & & + s_m & = 1190 \\
& a, & b, & s_c, & s_h, & s_m & \geq 0
\end{array}
$$



Degenerate Example

$$
\begin{array}{lrcrcrcrcrcl}
\max & 13a & + & 23b & & & & & & & & \\
\text{s.t.} & 5a & + & 15b & + & s_c & & & & & = & 480 \\
& \frac{80}{17} \cdot a & + & 4b & & & + & s_h & & & = & 160 \\
& 35a & + & 20b & & & & & + & s_m & = & 1190 \\
& a & , & b & , & s_c & , & s_h & , & s_m & \geq & 0
\end{array}
$$



Summary: How to choose pivot-elements

- We can choose a column e as an entering variable if $\tilde{c}_e > 0$ ($\tilde{c}_e$ is reduced cost for x_e).
- The standard choice is the column that maximizes $\tilde{c}_e$.
- If $A_{ie} \leq 0$ for all $i \in \{1, \ldots, m\}$ then the maximum is not bounded.
- Otw. choose a leaving variable ℓ such that $b_\ell / A_{\ell e}$ is minimal among all variables i with $A_{ie} > 0$.
- If several variables have minimum $b_\ell / A_{\ell e}$ you reach a degenerate basis.
- Depending on the choice of ℓ it may happen that the algorithm runs into a cycle where it does not escape from a degenerate vertex.

Summary: How to choose pivot-elements

- We can choose a column e as an entering variable if $\tilde{c}_e > 0$ ($\tilde{c}_e$ is reduced cost for x_e).
- The standard choice is the column that maximizes $\tilde{c}_e$.
- If $A_{ie} \leq 0$ for all $i \in \{1, \ldots, m\}$ then the maximum is not bounded.
- Otw. choose a leaving variable ℓ such that $b_\ell / A_{\ell e}$ is minimal among all variables i with $A_{ie} > 0$.
- If several variables have minimum $b_\ell / A_{\ell e}$ you reach a degenerate basis.
- Depending on the choice of ℓ it may happen that the algorithm runs into a cycle where it does not escape from a degenerate vertex.

Summary: How to choose pivot-elements

- We can choose a column e as an entering variable if $\tilde{c}_e > 0$ ($\tilde{c}_e$ is reduced cost for x_e).
- The standard choice is the column that maximizes $\tilde{c}_e$.
- If $A_{ie} \leq 0$ for all $i \in \{1, \ldots, m\}$ then the maximum is not bounded.
- Otw. choose a leaving variable ℓ such that $b_\ell / A_{\ell e}$ is minimal among all variables i with $A_{ie} > 0$.
- If several variables have minimum $b_\ell / A_{\ell e}$ you reach a degenerate basis.
- Depending on the choice of ℓ it may happen that the algorithm runs into a cycle where it does not escape from a degenerate vertex.

Summary: How to choose pivot-elements

- We can choose a column e as an entering variable if $\tilde{c}_e > 0$ ($\tilde{c}_e$ is reduced cost for x_e).
- The standard choice is the column that maximizes $\tilde{c}_e$.
- If $A_{ie} \leq 0$ for all $i \in \{1, \ldots, m\}$ then the maximum is not bounded.
- Otw. choose a leaving variable ℓ such that $b_\ell / A_{\ell e}$ is minimal among all variables i with $A_{ie} > 0$.
- If several variables have minimum $b_\ell / A_{\ell e}$ you reach a degenerate basis.
- Depending on the choice of ℓ it may happen that the algorithm runs into a cycle where it does not escape from a degenerate vertex.

Summary: How to choose pivot-elements

- We can choose a column e as an entering variable if $\tilde{c}_e > 0$ ($\tilde{c}_e$ is reduced cost for x_e).
- The standard choice is the column that maximizes $\tilde{c}_e$.
- If $A_{ie} \leq 0$ for all $i \in \{1, \ldots, m\}$ then the maximum is not bounded.
- Otw. choose a leaving variable ℓ such that $b_\ell / A_{\ell e}$ is minimal among all variables i with $A_{ie} > 0$.
- If several variables have minimum $b_\ell / A_{\ell e}$ you reach a degenerate basis.
- Depending on the choice of ℓ it may happen that the algorithm runs into a cycle where it does not escape from a degenerate vertex.

Summary: How to choose pivot-elements

- We can choose a column e as an entering variable if $\tilde{c}_e > 0$ ($\tilde{c}_e$ is reduced cost for x_e).
- The standard choice is the column that maximizes $\tilde{c}_e$.
- If $A_{ie} \leq 0$ for all $i \in \{1, \ldots, m\}$ then the maximum is not bounded.
- Otw. choose a leaving variable ℓ such that $b_\ell / A_{\ell e}$ is minimal among all variables i with $A_{ie} > 0$.
- If several variables have minimum $b_\ell / A_{\ell e}$ you reach a degenerate basis.
- Depending on the choice of ℓ it may happen that the algorithm runs into a cycle where it does not escape from a degenerate vertex.

Termination

What do we have so far?

Suppose we are given an initial feasible solution to an LP. If the LP is non-degenerate then Simplex will terminate.

Note that we either terminate because the min-ratio test fails and we can conclude that the LP is unbounded, or we terminate because the vector of reduced cost is non-positive. In the latter case we have an optimum solution.

How do we come up with an initial solution?

- $Ax \leq b, x \geq 0$, and $\boldsymbol{b \geq 0}$.
- The standard slack from for this problem is $Ax + Is = b, x \geq 0, s \geq 0$, where s denotes the vector of slack variables.
- Then $s = b$, $x = 0$ is a basic feasible solution (how?).
- We directly can start the simplex algorithm.

How do we find an initial basic feasible solution for an arbitrary problem?

How do we come up with an initial solution?

- $Ax \le b, x \ge 0$, and $\boldsymbol{b \ge 0}$.
- The standard slack from for this problem is $Ax + Is = b, x \ge 0, s \ge 0$, where s denotes the vector of slack variables.
- Then $s = b$, $x = 0$ is a basic feasible solution (how?).
- We directly can start the simplex algorithm.

How do we find an initial basic feasible solution for an arbitrary problem?

How do we come up with an initial solution?

- $Ax \leq b, x \geq 0$, and $\mathbf{b \geq 0}$.
- The standard slack from for this problem is $Ax + Is = b, x \geq 0, s \geq 0$, where s denotes the vector of slack variables.
- Then $s = b$, $x = 0$ is a basic feasible solution (how?).
- We directly can start the simplex algorithm.

How do we find an initial basic feasible solution for an arbitrary problem?

How do we come up with an initial solution?

- $Ax \leq b, x \geq 0$, and $\mathbf{b} \geq \mathbf{0}$.
- The standard slack from for this problem is $Ax + Is = b, x \geq 0, s \geq 0$, where s denotes the vector of slack variables.
- Then $s = b$, $x = 0$ is a basic feasible solution (how?).
- We directly can start the simplex algorithm.

How do we find an initial basic feasible solution for an arbitrary problem?

How do we come up with an initial solution?

- $Ax \leq b, x \geq 0$, and $\boldsymbol{b \geq 0}$.
- The standard slack from for this problem is $Ax + Is = b, x \geq 0, s \geq 0$, where s denotes the vector of slack variables.
- Then $s = b$, $x = 0$ is a basic feasible solution (how?).
- We directly can start the simplex algorithm.

How do we find an initial basic feasible solution for an arbitrary problem?

Two phase algorithm

Suppose we want to maximize $c^t x$ s.t. $Ax = b, x \geq 0$.

1. Multiply all rows with $b_i < 0$ by -1.
2. maximize $-\sum_i v_i$ s.t. $Ax + Iv = b, x \geq 0, v \geq 0$ using Simplex. $x = 0, v = b$ is initial feasible.
3. If $\sum_i v_i > 0$ then the original problem is infeasible.
4. Otw. you have $x \geq 0$ with $Ax = b$.
5. From this you can get basic feasible solution.
6. Now you can start the Simplex for the original problem.

Two phase algorithm

Suppose we want to maximize $c^t x$ s.t. $Ax = b, x \geq 0$.

1. Multiply all rows with $b_i < 0$ by -1.
2. maximize $-\sum_i v_i$ s.t. $Ax + Iv = b$, $x \geq 0$, $v \geq 0$ using Simplex. $x = 0$, $v = b$ is initial feasible.
3. If $\sum_i v_i > 0$ then the original problem is infeasible.
4. Otw. you have $x \geq 0$ with $Ax = b$.
5. From this you can get basic feasible solution.
6. Now you can start the Simplex for the original problem.

Two phase algorithm

Suppose we want to maximize $c^t x$ s.t. $Ax = b, x \geq 0$.

1. Multiply all rows with $b_i < 0$ by -1.
2. maximize $-\sum_i v_i$ s.t. $Ax + Iv = b$, $x \geq 0$, $v \geq 0$ using Simplex. $x = 0$, $v = b$ is initial feasible.
3. If $\sum_i v_i > 0$ then the original problem is infeasible.
4. Otw. you have $x \geq 0$ with $Ax = b$.
5. From this you can get basic feasible solution.
6. Now you can start the Simplex for the original problem.

Two phase algorithm

Suppose we want to maximize $c^t x$ s.t. $Ax = b, x \geq 0$.

1. Multiply all rows with $b_i < 0$ by -1.
2. maximize $-\sum_i v_i$ s.t. $Ax + Iv = b$, $x \geq 0$, $v \geq 0$ using Simplex. $x = 0$, $v = b$ is initial feasible.
3. If $\sum_i v_i > 0$ then the original problem is infeasible.
4. Otw. you have $x \geq 0$ with $Ax = b$.
5. From this you can get basic feasible solution.
6. Now you can start the Simplex for the original problem.

Two phase algorithm

Suppose we want to maximize $c^t x$ s.t. $Ax = b, x \geq 0$.

1. Multiply all rows with $b_i < 0$ by -1.
2. maximize $-\sum_i v_i$ s.t. $Ax + Iv = b$, $x \geq 0$, $v \geq 0$ using Simplex. $x = 0$, $v = b$ is initial feasible.
3. If $\sum_i v_i > 0$ then the original problem is infeasible.
4. Otw. you have $x \geq 0$ with $Ax = b$.
5. From this you can get basic feasible solution.
6. Now you can start the Simplex for the original problem.

Two phase algorithm

Suppose we want to maximize $c^t x$ s.t. $Ax = b, x \geq 0$.

1. Multiply all rows with $b_i < 0$ by -1.
2. maximize $-\sum_i v_i$ s.t. $Ax + Iv = b$, $x \geq 0$, $v \geq 0$ using Simplex. $x = 0$, $v = b$ is initial feasible.
3. If $\sum_i v_i > 0$ then the original problem is infeasible.
4. Otw. you have $x \geq 0$ with $Ax = b$.
5. From this you can get basic feasible solution.
6. Now you can start the Simplex for the original problem.

Two phase algorithm

Suppose we want to maximize $c^t x$ s.t. $Ax = b, x \geq 0$.

1. Multiply all rows with $b_i < 0$ by -1.
2. maximize $-\sum_i v_i$ s.t. $Ax + Iv = b$, $x \geq 0$, $v \geq 0$ using Simplex. $x = 0$, $v = b$ is initial feasible.
3. If $\sum_i v_i > 0$ then the original problem is infeasible.
4. Otw. you have $x \geq 0$ with $Ax = b$.
5. From this you can get basic feasible solution.
6. Now you can start the Simplex for the original problem.

Optimality

Lemma 14
Let B be a basis and x^* a BFS corresponding to basis B. $\tilde{c} \leq 0$ implies that x^* is an optimum solution to the LP.

Duality

How do we get an upper bound to a maximization LP?

$$\begin{array}{rrcrl} \text{max} & 13a & + & 23b & \\ \text{s.t.} & 5a & + & 15b & \le 480 \\ & 4a & + & 4b & \le 160 \\ & 35a & + & 20b & \le 1190 \\ & & & a, b & \ge 0 \end{array}$$

Note that a lower bound is easy to derive. Every choice of $a, b \ge 0$ gives us a lower bound (e.g. $a = 12, b = 28$ gives us a lower bound of 800).

If you take a conic combination of the rows (multiply the i-th row with $y_i \ge 0$) such that $\sum_i y_i a_{ij} \ge c_j$ then $\sum_i y_i b_i$ will be an upper bound.

Duality

How do we get an upper bound to a maximization LP?

$$\begin{array}{rrcrl} \text{max} & 13a & + & 23b & \\ \text{s.t.} & 5a & + & 15b & \leq 480 \\ & 4a & + & 4b & \leq 160 \\ & 35a & + & 20b & \leq 1190 \\ & & & a, b & \geq 0 \end{array}$$

Note that a lower bound is easy to derive. Every choice of $a, b \geq 0$ gives us a lower bound (e.g. $a = 12, b = 28$ gives us a lower bound of 800).

If you take a conic combination of the rows (multiply the i-th row with $y_i \geq 0$) such that $\sum_i y_i a_{ij} \geq c_j$ then $\sum_i y_i b_i$ will be an upper bound.

Duality

How do we get an upper bound to a maximization LP?

$$\begin{array}{rrcrl} \text{max} & 13a & + & 23b & \\ \text{s.t.} & 5a & + & 15b & \leq 480 \\ & 4a & + & 4b & \leq 160 \\ & 35a & + & 20b & \leq 1190 \\ & & & a, b & \geq 0 \end{array}$$

Note that a lower bound is easy to derive. Every choice of $a, b \geq 0$ gives us a lower bound (e.g. $a = 12, b = 28$ gives us a lower bound of 800).

If you take a conic combination of the rows (multiply the i-th row with $y_i \geq 0$) such that $\sum_i y_i a_{ij} \geq c_j$ then $\sum_i y_i b_i$ will be an upper bound.

Duality

Definition 15
Let $z = \max\{c^t x \mid Ax \le b, x \ge 0\}$ be a linear program P (called the primal linear program).

The linear program D defined by

$$w = \min\{b^t y \mid A^t y \ge c, y \ge 0\}$$

is called the dual problem.

Duality

Lemma 16
The dual of the dual problem is the primal problem.

Proof:

- $w = \min\{b^t y \mid A^t y \geq c, y \geq 0\}$
- $w = -\max\{-b^t y \mid -A^t y \leq -c, y \geq 0\}$

The dual problem is

- $z = -\min\{-c^t x \mid -Ax \geq -b, x \geq 0\}$
- $z = \max\{c^t x \mid Ax \leq b, x \geq 0\}$

Duality

Lemma 16

The dual of the dual problem is the primal problem.

Proof:

- $w = \min\{b^t y \mid A^t y \geq c, y \geq 0\}$
- $w = -\max\{-b^t y \mid -A^t y \leq -c, y \geq 0\}$

The dual problem is

- $z = -\min\{-c^t x \mid -Ax \geq -b, x \geq 0\}$
- $z = \max\{c^t x \mid Ax \leq b, x \geq 0\}$

Duality

Lemma 16

The dual of the dual problem is the primal problem.

Proof:

- $w = \min\{b^t y \mid A^t y \geq c, y \geq 0\}$
- $w = -\max\{-b^t y \mid -A^t y \leq -c, y \geq 0\}$

The dual problem is

- $z = -\min\{-c^t x \mid -Ax \geq -b, x \geq 0\}$
- $z = \max\{c^t x \mid Ax \leq b, x \geq 0\}$

Duality

Lemma 16

The dual of the dual problem is the primal problem.

Proof:

- $w = \min\{b^t y \mid A^t y \geq c, y \geq 0\}$
- $w = -\max\{-b^t y \mid -A^t y \leq -c, y \geq 0\}$

The dual problem is

- $z = -\min\{-c^t x \mid -Ax \geq -b, x \geq 0\}$
- $z = \max\{c^t x \mid Ax \leq b, x \geq 0\}$

Duality

Lemma 16

The dual of the dual problem is the primal problem.

Proof:

- $w = \min\{b^t y \mid A^t y \geq c, y \geq 0\}$
- $w = -\max\{-b^t y \mid -A^t y \leq -c, y \geq 0\}$

The dual problem is

- $z = -\min\{-c^t x \mid -Ax \geq -b, x \geq 0\}$
- $z = \max\{c^t x \mid Ax \leq b, x \geq 0\}$

Weak Duality

Let $z = \max\{c^t x \mid Ax \le b, x \ge 0\}$ and
$w = \min\{b^t y \mid A^t y \ge c, y \ge 0\}$ be a primal dual pair.

x is primal feasible iff $x \in \{x \mid Ax \le b, x \ge 0\}$

y is dual feasible, iff $y \in \{y \mid A^t y \ge c, y \ge 0\}$.

Theorem 17 (Weak Duality)

Let $\hat{x}$ be primal feasible and let $\hat{y}$ be dual feasible. Then

$$c^t \hat{x} \le z \le w \le b^t \hat{y} \ .$$

Weak Duality

Let $z = \max\{c^t x \mid Ax \le b, x \ge 0\}$ and
$w = \min\{b^t y \mid A^t y \ge c, y \ge 0\}$ be a primal dual pair.

x is primal feasible iff $x \in \{x \mid Ax \le b, x \ge 0\}$

y is dual feasible, iff $y \in \{y \mid A^t y \ge c, y \ge 0\}$.

Theorem 17 (Weak Duality)

Let $\hat{x}$ be primal feasible and let $\hat{y}$ be dual feasible. Then

$$c^t \hat{x} \le z \le w \le b^t \hat{y} \ .$$

Weak Duality

$A^t\hat{y} \geq c \Rightarrow \hat{x}^tA^t\hat{y} \geq \hat{x}^tc \; (\hat{x} \geq 0)$

$A\hat{x} \leq b \Rightarrow y^tA\hat{x} \leq \hat{y}^tb \; (\hat{y} \geq 0)$

This gives

$$c^t\hat{x} \leq \hat{y}^tA\hat{x} \leq b^t\hat{y} \; .$$

Since, there exists primal feasible $\hat{x}$ with $c^t\hat{x} = z$, and dual feasible $\hat{y}$ with $b^ty = w$ we get $z \leq w$.

If P is unbounded then D is infeasible.

Weak Duality

$$A^t\hat{y} \geq c \Rightarrow \hat{x}^t A^t \hat{y} \geq \hat{x}^t c \;(\hat{x} \geq 0)$$

$$A\hat{x} \leq b \Rightarrow y^t A\hat{x} \leq \hat{y}^t b \;(\hat{y} \geq 0)$$

This gives

$$c^t\hat{x} \leq \hat{y}^t A\hat{x} \leq b^t\hat{y} \quad .$$

Since, there exists primal feasible $\hat{x}$ with $c^t\hat{x} = z$, and dual feasible $\hat{y}$ with $b^t y = w$ we get $z \leq w$.

If P is unbounded then D is infeasible.

Weak Duality

$$A^t\hat{y} \geq c \Rightarrow \hat{x}^t A^t \hat{y} \geq \hat{x}^t c \; (\hat{x} \geq 0)$$

$$A\hat{x} \leq b \Rightarrow y^t A\hat{x} \leq \hat{y}^t b \; (\hat{y} \geq 0)$$

This gives

$$c^t\hat{x} \leq \hat{y}^t A \hat{x} \leq b^t\hat{y} \; .$$

Since, there exists primal feasible $\hat{x}$ with $c^t\hat{x} = z$, and dual feasible $\hat{y}$ with $b^t y = w$ we get $z \leq w$.

If P is unbounded then D is infeasible.

Weak Duality

$A^t\hat{y} \geq c \Rightarrow \hat{x}^t A^t \hat{y} \geq \hat{x}^t c$ $(\hat{x} \geq 0)$

$A\hat{x} \leq b \Rightarrow y^t A\hat{x} \leq \hat{y}^t b$ $(\hat{y} \geq 0)$

This gives

$$c^t\hat{x} \leq \hat{y}^t A\hat{x} \leq b^t\hat{y} \quad .$$

Since, there exists primal feasible $\hat{x}$ with $c^t\hat{x} = z$, and dual feasible $\hat{y}$ with $b^t y = w$ we get $z \leq w$.

If P is unbounded then D is infeasible.

Weak Duality

$A^t\hat{y} \geq c \Rightarrow \hat{x}^t A^t \hat{y} \geq \hat{x}^t c$ $(\hat{x} \geq 0)$

$A\hat{x} \leq b \Rightarrow y^t A\hat{x} \leq \hat{y}^t b$ $(\hat{y} \geq 0)$

This gives

$$c^t\hat{x} \leq \hat{y}^t A\hat{x} \leq b^t\hat{y} \ .$$

Since, there exists primal feasible $\hat{x}$ with $c^t\hat{x} = z$, and dual feasible $\hat{y}$ with $b^t y = w$ we get $z \leq w$.

If P is unbounded then D is infeasible.

Weak Duality

$$A^t\hat{y} \geq c \Rightarrow \hat{x}^t A^t \hat{y} \geq \hat{x}^t c \;(\hat{x} \geq 0)$$

$$A\hat{x} \leq b \Rightarrow y^t A\hat{x} \leq \hat{y}^t b \;(\hat{y} \geq 0)$$

This gives

$$c^t\hat{x} \leq \hat{y}^t A\hat{x} \leq b^t\hat{y} \;.$$

Since, there exists primal feasible $\hat{x}$ with $c^t\hat{x} = z$, and dual feasible $\hat{y}$ with $b^t y = w$ we get $z \leq w$.

If P is unbounded then D is infeasible.

Weak Duality

$$A^t\hat{y} \geq c \Rightarrow \hat{x}^t A^t \hat{y} \geq \hat{x}^t c \ (\hat{x} \geq 0)$$

$$A\hat{x} \leq b \Rightarrow y^t A\hat{x} \leq \hat{y}^t b \ (\hat{y} \geq 0)$$

This gives

$$c^t\hat{x} \leq \hat{y}^t A\hat{x} \leq b^t\hat{y} \ .$$

Since, there exists primal feasible $\hat{x}$ with $c^t\hat{x} = z$, and dual feasible $\hat{y}$ with $b^t y = w$ we get $z \leq w$.

If P is unbounded then D is infeasible.

Weak Duality

$A^t\hat{y} \geq c \Rightarrow \hat{x}^t A^t \hat{y} \geq \hat{x}^t c$ $(\hat{x} \geq 0)$

$A\hat{x} \leq b \Rightarrow y^t A\hat{x} \leq \hat{y}^t b$ $(\hat{y} \geq 0)$

This gives

$$c^t\hat{x} \leq \hat{y}^t A\hat{x} \leq b^t\hat{y} \ .$$

Since, there exists primal feasible $\hat{x}$ with $c^t\hat{x} = z$, and dual feasible $\hat{y}$ with $b^t y = w$ we get $z \leq w$.

If P is unbounded then D is infeasible.

Weak Duality

$A^t\hat{y} \geq c \Rightarrow \hat{x}^tA^t\hat{y} \geq \hat{x}^tc$ $(\hat{x} \geq 0)$

$A\hat{x} \leq b \Rightarrow y^tA\hat{x} \leq \hat{y}^tb$ $(\hat{y} \geq 0)$

This gives

$$c^t\hat{x} \leq \hat{y}^tA\hat{x} \leq b^t\hat{y} \ .$$

Since, there exists primal feasible $\hat{x}$ with $c^t\hat{x} = z$, and dual feasible $\hat{y}$ with $b^ty = w$ we get $z \leq w$.

If P is unbounded then D is infeasible.

The following linear programs form a primal dual pair:

$$
\begin{aligned}
z &= \max\{c^t x \mid Ax = b, x \geq 0\} \\
w &= \min\{b^t y \mid A^t y \geq c\}
\end{aligned}
$$

This means for computing the dual of a standard form LP, we do not have non-negativity constraints for the dual variables.

Proof

Primal:

$$\max\{c^t x \mid Ax = b, x \geq 0\}$$

Proof

Primal:

$$\begin{aligned}&\max\{c^t x \mid Ax = b, x \geq 0\}\\&\qquad = \max\{c^t x \mid Ax \leq b, -Ax \leq -b, x \geq 0\}\end{aligned}$$

Proof

Primal:

$$
\begin{aligned}
\max\{c^t x \mid Ax = b, x \geq 0\} \\
&= \max\{c^t x \mid Ax \leq b, -Ax \leq -b, x \geq 0\} \\
&= \max\{c^t x \mid \begin{bmatrix} A \\ -A \end{bmatrix} x \leq \begin{bmatrix} b \\ -b \end{bmatrix}, x \geq 0\}
\end{aligned}
$$

Proof

Primal:

$$\max\{c^t x \mid Ax = b, x \ge 0\}$$
$$= \max\{c^t x \mid Ax \le b, -Ax \le -b, x \ge 0\}$$
$$= \max\{c^t x \mid \begin{bmatrix} A \\ -A \end{bmatrix} x \le \begin{bmatrix} b \\ -b \end{bmatrix}, x \ge 0\}$$

Dual:

$$\min\{[b^t \; -b^t]y \mid [A^t \; -A^t]y \ge c, y \ge 0\}$$

Proof

Primal:

$$\begin{aligned}\max\{c^tx \mid Ax = b, x \geq 0\}\\ &= \max\{c^tx \mid Ax \leq b, -Ax \leq -b, x \geq 0\}\\ &= \max\{c^tx \mid \begin{bmatrix} A \\ -A \end{bmatrix} x \leq \begin{bmatrix} b \\ -b \end{bmatrix}, x \geq 0\}\end{aligned}$$

Dual:

$$\begin{aligned}\min\{[b^t \; -b^t]y \mid [A^t \; -A^t]y \geq c, y \geq 0\}\\ &= \min\left\{[b^t \; -b^t] \cdot \begin{bmatrix} y^+ \\ y^- \end{bmatrix} \,\middle|\, [A^t \; -A^t] \cdot \begin{bmatrix} y^+ \\ y^- \end{bmatrix} \geq c, y^- \geq 0, y^+ \geq 0\right\}\end{aligned}$$

Proof

Primal:

$$\begin{aligned}
&\max\{c^t x \mid Ax = b, x \geq 0\}\\
&\quad = \max\{c^t x \mid Ax \leq b, -Ax \leq -b, x \geq 0\}\\
&\quad = \max\{c^t x \mid \begin{bmatrix} A \\ -A \end{bmatrix} x \leq \begin{bmatrix} b \\ -b \end{bmatrix}, x \geq 0\}
\end{aligned}$$

Dual:

$$\begin{aligned}
&\min\{[b^t \; -b^t] y \mid [A^t \; -A^t] y \geq c, y \geq 0\}\\
&\quad = \min\left\{[b^t \; -b^t] \cdot \begin{bmatrix} y^+ \\ y^- \end{bmatrix} \;\middle|\; [A^t \; -A^t] \cdot \begin{bmatrix} y^+ \\ y^- \end{bmatrix} \geq c, y^- \geq 0, y^+ \geq 0\right\}\\
&\quad = \min\left\{b^t \cdot (y^+ - y^-) \;\middle|\; A^t \cdot (y^+ - y^-) \geq c, y^- \geq 0, y^+ \geq 0\right\}
\end{aligned}$$

Proof

Primal:

$$\begin{aligned}\max\{c^t x \mid Ax = b, x \geq 0\} \\ &= \max\{c^t x \mid Ax \leq b, -Ax \leq -b, x \geq 0\} \\ &= \max\{c^t x \mid \begin{bmatrix} A \\ -A \end{bmatrix} x \leq \begin{bmatrix} b \\ -b \end{bmatrix}, x \geq 0\}\end{aligned}$$

Dual:

$$\begin{aligned}\min\{[b^t\ -b^t]y \mid [A^t\ -A^t]y \geq c, y \geq 0\} \\ &= \min\left\{[b^t\ -b^t] \cdot \begin{bmatrix} y^+ \\ y^- \end{bmatrix} \,\middle|\, [A^t\ -A^t] \cdot \begin{bmatrix} y^+ \\ y^- \end{bmatrix} \geq c, y^- \geq 0, y^+ \geq 0\right\} \\ &= \min\left\{b^t \cdot (y^+ - y^-) \,\middle|\, A^t \cdot (y^+ - y^-) \geq c, y^- \geq 0, y^+ \geq 0\right\} \\ &= \min\left\{b^t y' \,\middle|\, A^t y' \geq c\right\}\end{aligned}$$

Proof of Optimality Criterion for Simplex

Suppose that we have a basic feasible solution with reduced cost

$$\tilde{c} = c^t - c_B^t A_B^{-1} A \le 0$$

This is equivalent to $A^t (A_B^{-1})^t c_B \ge c$

$y^* = (A_B^{-1})^t c_B$ is solution to the dual $\min\{b^t y | A^t y \ge c\}$.

$$\begin{aligned} b^t y^* &= (Ax^*)^t y^* = (A_B x_B^*)^t y^* \\ &= (A_B x_B^*)^t (A_B^{-1})^t c_B = (x_B^*)^t A_B^t (A_B^{-1})^t c_B \\ &= c^t x^* \end{aligned}$$

Hence, the solution is optimal.

Proof of Optimality Criterion for Simplex

Suppose that we have a basic feasible solution with reduced cost

$$\tilde{c} = c^t - c_B^t A_B^{-1} A \le 0$$

This is equivalent to $A^t (A_B^{-1})^t c_B \ge c$

$y^* = (A_B^{-1})^t c_B$ is solution to the dual $\min\{b^t y | A^t y \ge c\}$.

$$\begin{aligned} b^t y^* &= (Ax^*)^t y^* = (A_B x_B^*)^t y^* \\ &= (A_B x_B^*)^t (A_B^{-1})^t c_B = (x_B^*)^t A_B^t (A_B^{-1})^t c_B \\ &= c^t x^* \end{aligned}$$

Hence, the solution is optimal.

Proof of Optimality Criterion for Simplex

Suppose that we have a basic feasible solution with reduced cost

$$\tilde{c} = c^t - c_B^t A_B^{-1} A \leq 0$$

This is equivalent to $A^t (A_B^{-1})^t c_B \geq c$

$y^* = (A_B^{-1})^t c_B$ is solution to the dual $\min\{b^t y | A^t y \geq c\}$.

$$\begin{aligned} b^t y^* &= (Ax^*)^t y^* = (A_B x_B^*)^t y^* \\ &= (A_B x_B^*)^t (A_B^{-1})^t c_B = (x_B^*)^t A_B^t (A_B^{-1})^t c_B \\ &= c^t x^* \end{aligned}$$

Hence, the solution is optimal.

Proof of Optimality Criterion for Simplex

Suppose that we have a basic feasible solution with reduced cost

$$\tilde{c} = c^t - c_B^t A_B^{-1} A \leq 0$$

This is equivalent to $A^t (A_B^{-1})^t c_B \geq c$

$y^* = (A_B^{-1})^t c_B$ is solution to the dual $\min\{b^t y | A^t y \geq c\}$.

$$\begin{aligned} b^t y^* &= (Ax^*)^t y^* = (A_B x_B^*)^t y^* \\ &= (A_B x_B^*)^t (A_B^{-1})^t c_B = (x_B^*)^t A_B^t (A_B^{-1})^t c_B \\ &= c^t x^* \end{aligned}$$

Hence, the solution is optimal.

Proof of Optimality Criterion for Simplex

Suppose that we have a basic feasible solution with reduced cost

$$\tilde{c} = c^t - c_B^t A_B^{-1} A \leq 0$$

This is equivalent to $A^t (A_B^{-1})^t c_B \geq c$

$y^* = (A_B^{-1})^t c_B$ is solution to the dual $\min\{b^t y | A^t y \geq c\}$.

$$\begin{aligned} b^t y^* &= (Ax^*)^t y^* = (A_B x_B^*)^t y^* \\ &= (A_B x_B^*)^t (A_B^{-1})^t c_B = (x_B^*)^t A_B^t (A_B^{-1})^t c_B \\ &= c^t x^* \end{aligned}$$

Hence, the solution is optimal.

Proof of Optimality Criterion for Simplex

Suppose that we have a basic feasible solution with reduced cost

$$\tilde{c} = c^t - c_B^t A_B^{-1} A \leq 0$$

This is equivalent to $A^t (A_B^{-1})^t c_B \geq c$

$y^* = (A_B^{-1})^t c_B$ is solution to the dual $\min\{b^t y | A^t y \geq c\}$.

$$\begin{aligned} b^t y^* &= (Ax^*)^t y^* = (A_B x_B^*)^t y^* \\ &= (A_B x_B^*)^t (A_B^{-1})^t c_B = (x_B^*)^t A_B^t (A_B^{-1})^t c_B \\ &= c^t x^* \end{aligned}$$

Hence, the solution is optimal.

Proof of Optimality Criterion for Simplex

Suppose that we have a basic feasible solution with reduced cost

$$\tilde{c} = c^t - c_B^t A_B^{-1} A \le 0$$

This is equivalent to $A^t (A_B^{-1})^t c_B \ge c$

$y^* = (A_B^{-1})^t c_B$ is solution to the dual $\min\{b^t y | A^t y \ge c\}$.

$$\begin{aligned} b^t y^* &= (Ax^*)^t y^* = (A_B x_B^*)^t y^* \\ &= (A_B x_B^*)^t (A_B^{-1})^t c_B = (x_B^*)^t A_B^t (A_B^{-1})^t c_B \\ &= c^t x^* \end{aligned}$$

Hence, the solution is optimal.

Proof of Optimality Criterion for Simplex

Suppose that we have a basic feasible solution with reduced cost

$$\tilde{c} = c^t - c_B^t A_B^{-1} A \le 0$$

This is equivalent to $A^t (A_B^{-1})^t c_B \ge c$

$y^* = (A_B^{-1})^t c_B$ is solution to the dual $\min\{b^t y | A^t y \ge c\}$.

$$\begin{aligned} b^t y^* &= (Ax^*)^t y^* = (A_B x_B^*)^t y^* \\ &= (A_B x_B^*)^t (A_B^{-1})^t c_B = (x_B^*)^t A_B^t (A_B^{-1})^t c_B \\ &= c^t x^* \end{aligned}$$

Hence, the solution is optimal.

Proof of Optimality Criterion for Simplex

Suppose that we have a basic feasible solution with reduced cost

$$\tilde{c} = c^t - c_B^t A_B^{-1} A \leq 0$$

This is equivalent to $A^t (A_B^{-1})^t c_B \geq c$

$y^* = (A_B^{-1})^t c_B$ is solution to the dual $\min\{b^t y | A^t y \geq c\}$.

$$\begin{aligned} b^t y^* &= (Ax^*)^t y^* = (A_B x_B^*)^t y^* \\ &= (A_B x_B^*)^t (A_B^{-1})^t c_B = (x_B^*)^t A_B^t (A_B^{-1})^t c_B \\ &= c^t x^* \end{aligned}$$

Hence, the solution is optimal.

Strong Duality

Theorem 18 (Strong Duality)

Let P and D be a primal dual pair of linear programs, and let z^* and w^* denote the optimal solution to P and D, respectively. Then

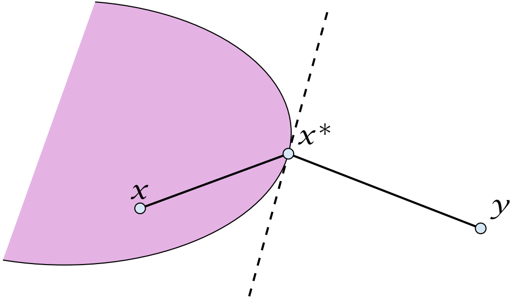
$$z^* = w^*$$

Lemma 19 (Weierstrass)

Let X be a compact set and let $f(x)$ be a continuous function on X. Then $\min\{f(x) : x \in X\}$ exists.

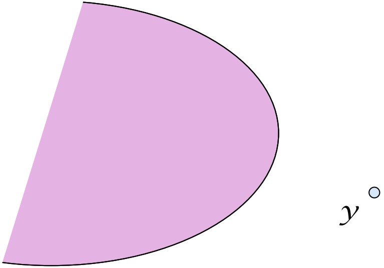
Lemma 20 (Projection Lemma)

Let $X \subseteq \mathbb{R}^m$ be a non-empty convex set, and let $y \notin X$. Then there exist $x^* \in X$ with minimum distance from y. Moreover for all $x \in X$ we have $(y - x^*)^t(x - x^*) \leq 0$.



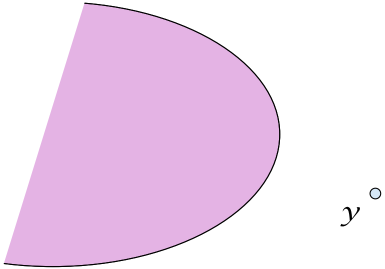
Proof of the Projection Lemma

- Define $f(x) = \|y - x\|$.
- We want to apply Weierstrass but X may not be bounded.
- $X \neq \emptyset$. Hence, there exists $x' \in X$.
- Define $X' = \{x \in X \mid \|y - x\| \leq \|y - x'\|\}$. This set is closed and bounded.
- Applying Weierstrass gives the existence.



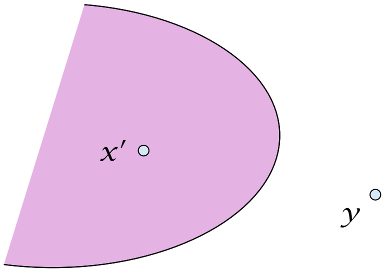
Proof of the Projection Lemma

- Define $f(x) = \|y - x\|$.
- We want to apply Weierstrass but X may not be bounded.
- $X \neq \emptyset$. Hence, there exists $x' \in X$.
- Define $X' = \{x \in X \mid \|y - x\| \leq \|y - x'\|\}$. This set is closed and bounded.
- Applying Weierstrass gives the existence.



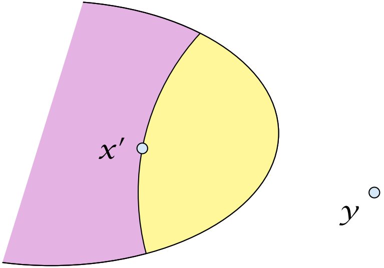
Proof of the Projection Lemma

- Define $f(x) = \|y - x\|$.
- We want to apply Weierstrass but X may not be bounded.
- $X \neq \emptyset$. Hence, there exists $x' \in X$.
- Define $X' = \{x \in X \mid \|y - x\| \leq \|y - x'\|\}$. This set is closed and bounded.
- Applying Weierstrass gives the existence.



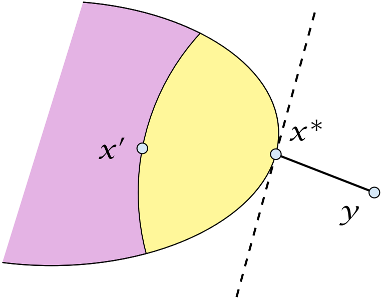
Proof of the Projection Lemma

- Define $f(x) = \|y - x\|$.
- We want to apply Weierstrass but X may not be bounded.
- $X \neq \emptyset$. Hence, there exists $x' \in X$.
- Define $X' = \{x \in X \mid \|y - x\| \leq \|y - x'\|\}$. This set is closed and bounded.
- Applying Weierstrass gives the existence.



Proof of the Projection Lemma

- Define $f(x) = \|y - x\|$.
- We want to apply Weierstrass but X may not be bounded.
- $X \neq \emptyset$. Hence, there exists $x' \in X$.
- Define $X' = \{x \in X \mid \|y - x\| \leq \|y - x'\|\}$. This set is closed and bounded.
- Applying Weierstrass gives the existence.



Proof of the Projection Lemma (continued)

Proof of the Projection Lemma (continued)

x^* is minimum. Hence $\|y - x^*\|^2 \le \|y - x\|^2$ for all $x \in X$.

Proof of the Projection Lemma (continued)

x^* is minimum. Hence $\|y - x^*\|^2 \leq \|y - x\|^2$ for all $x \in X$.

By convexity: $x \in X$ then $x^* + \epsilon(x - x^*) \in X$ for all $0 \leq \epsilon \leq 1$.

Proof of the Projection Lemma (continued)

x^* is minimum. Hence $\|y - x^*\|^2 \leq \|y - x\|^2$ for all $x \in X$.

By convexity: $x \in X$ then $x^* + \epsilon(x - x^*) \in X$ for all $0 \leq \epsilon \leq 1$.

$$\|y - x^*\|^2$$

Proof of the Projection Lemma (continued)

x^* is minimum. Hence $\|y - x^*\|^2 \leq \|y - x\|^2$ for all $x \in X$.

By convexity: $x \in X$ then $x^* + \epsilon(x - x^*) \in X$ for all $0 \leq \epsilon \leq 1$.

$$\|y - x^*\|^2 \leq \|y - x^* - \epsilon(x - x^*)\|^2$$

Proof of the Projection Lemma (continued)

x^* is minimum. Hence $\|y - x^*\|^2 \le \|y - x\|^2$ for all $x \in X$.

By convexity: $x \in X$ then $x^* + \epsilon(x - x^*) \in X$ for all $0 \le \epsilon \le 1$.

$$\begin{aligned}\|y - x^*\|^2 &\le \|y - x^* - \epsilon(x - x^*)\|^2 \\ &= \|y - x^*\|^2 + \epsilon^2\|x - x^*\|^2 - 2\epsilon(y - x^*)^t(x - x^*)\end{aligned}$$

Proof of the Projection Lemma (continued)

x^* is minimum. Hence $\|y - x^*\|^2 \le \|y - x\|^2$ for all $x \in X$.

By convexity: $x \in X$ then $x^* + \epsilon(x - x^*) \in X$ for all $0 \le \epsilon \le 1$.

$$\begin{aligned}\|y - x^*\|^2 &\le \|y - x^* - \epsilon(x - x^*)\|^2 \\ &= \|y - x^*\|^2 + \epsilon^2\|x - x^*\|^2 - 2\epsilon(y - x^*)^t(x - x^*)\end{aligned}$$

Hence, $(y - x^*)^t(x - x^*) \le \frac{1}{2}\epsilon\|x - x^*\|^2$.

Proof of the Projection Lemma (continued)

x^* is minimum. Hence $\|y - x^*\|^2 \le \|y - x\|^2$ for all $x \in X$.

By convexity: $x \in X$ then $x^* + \epsilon(x - x^*) \in X$ for all $0 \le \epsilon \le 1$.

$$\begin{aligned}\|y - x^*\|^2 &\le \|y - x^* - \epsilon(x - x^*)\|^2 \\ &= \|y - x^*\|^2 + \epsilon^2\|x - x^*\|^2 - 2\epsilon(y - x^*)^t(x - x^*)\end{aligned}$$

Hence, $(y - x^*)^t(x - x^*) \le \frac{1}{2}\epsilon\|x - x^*\|^2$.

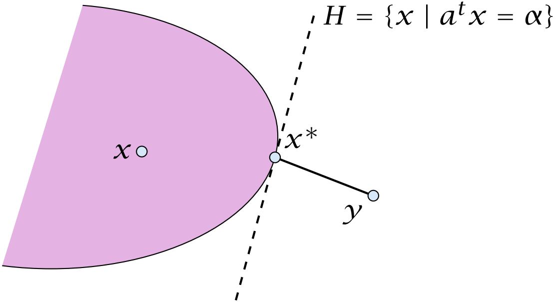
Letting $\epsilon \to 0$ gives the result.

Theorem 21 (Separating Hyperplane)

Let $X \subseteq \mathbb{R}^m$ be a non-empty closed convex set, and let $y \notin X$. Then there exists a separating hyperplane $\{x \in \mathbb{R} : a^t x = \alpha\}$ where $a \in \mathbb{R}^m$, $\alpha \in \mathbb{R}$ that separates y from X. ($a^t y < \alpha$; $a^t x \geq \alpha$ for all $x \in X$)

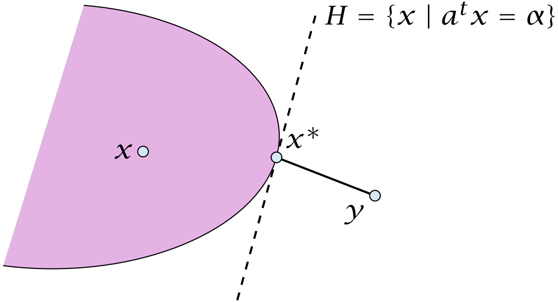
Proof of the Hyperplane Lemma

- Let $x^* \in X$ be closest point to y in X.
- By previous lemma $(y - x^*)^t(x - x^*) \le 0$ for all $x \in X$.
- Choose $a = (x^* - y)$ and $\alpha = a^t x^*$.
- For $x \in X : a^t(x - x^*) \ge 0$, and, hence, $a^t x \ge \alpha$.
- Also, $a^t y = a^t(x^* - a) = \alpha - \|a\|^2 < \alpha$



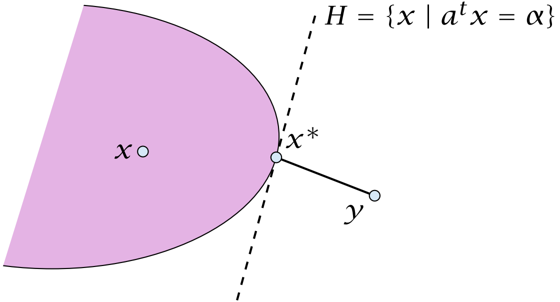
Proof of the Hyperplane Lemma

- Let $x^* \in X$ be closest point to y in X.
- By previous lemma $(y - x^*)^t(x - x^*) \leq 0$ for all $x \in X$.
- Choose $a = (x^* - y)$ and $\alpha = a^t x^*$.
- For $x \in X : a^t(x - x^*) \geq 0$, and, hence, $a^t x \geq \alpha$.
- Also, $a^t y = a^t(x^* - a) = \alpha - \|a\|^2 < \alpha$



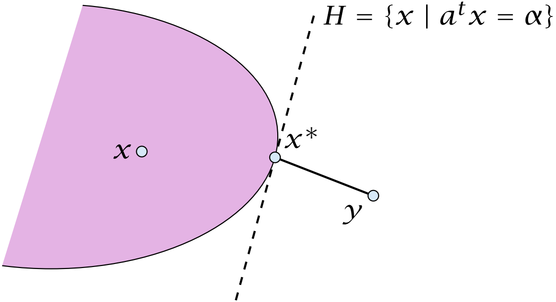
Proof of the Hyperplane Lemma

- Let $x^* \in X$ be closest point to y in X.
- By previous lemma $(y - x^*)^t(x - x^*) \leq 0$ for all $x \in X$.
- Choose $a = (x^* - y)$ and $\alpha = a^t x^*$.
- For $x \in X : a^t(x - x^*) \geq 0$, and, hence, $a^t x \geq \alpha$.
- Also, $a^t y = a^t(x^* - a) = \alpha - \|a\|^2 < \alpha$



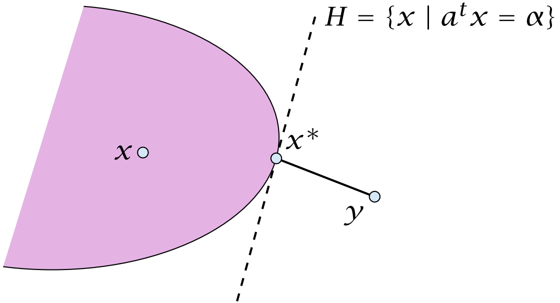
Proof of the Hyperplane Lemma

- Let $x^* \in X$ be closest point to y in X.
- By previous lemma $(y - x^*)^t(x - x^*) \leq 0$ for all $x \in X$.
- Choose $a = (x^* - y)$ and $\alpha = a^t x^*$.
- For $x \in X : a^t(x - x^*) \geq 0$, and, hence, $a^t x \geq \alpha$.
- Also, $a^t y = a^t(x^* - a) = \alpha - \|a\|^2 < \alpha$



Proof of the Hyperplane Lemma

- Let $x^* \in X$ be closest point to y in X.
- By previous lemma $(y - x^*)^t(x - x^*) \leq 0$ for all $x \in X$.
- Choose $a = (x^* - y)$ and $\alpha = a^t x^*$.
- For $x \in X : a^t(x - x^*) \geq 0$, and, hence, $a^t x \geq \alpha$.
- Also, $a^t y = a^t(x^* - a) = \alpha - \|a\|^2 < \alpha$



Lemma 22 (Farkas Lemma)

Let A be an $m \times n$ matrix, $b \in \mathbb{R}^m$. Then exactly one of the following statements holds.

1. $\exists x \in \mathbb{R}^n$ *with* $Ax = b$, $x \geq 0$
2. $\exists y \in \mathbb{R}^m$ *with* $A^t y \geq 0$, $b^t y < 0$

Assume $\hat{x}$ satisfies 1. and $\hat{y}$ satisfies 2. Then

$$0 > y^t b = y^t Ax \geq 0$$

Hence, at most one of the statements can hold.

Lemma 22 (Farkas Lemma)

Let A be an $m \times n$ matrix, $b \in \mathbb{R}^m$. Then exactly one of the following statements holds.

1. $\exists x \in \mathbb{R}^n$ *with* $Ax = b$, $x \geq 0$
2. $\exists y \in \mathbb{R}^m$ *with* $A^t y \geq 0$, $b^t y < 0$

Assume $\hat{x}$ satisfies 1. and $\hat{y}$ satisfies 2. Then

$$0 > y^t b = y^t Ax \geq 0$$

Hence, at most one of the statements can hold.

Lemma 22 (Farkas Lemma)

Let A be an $m \times n$ matrix, $b \in \mathbb{R}^m$. Then exactly one of the following statements holds.

1. $\exists x \in \mathbb{R}^n$ *with* $Ax = b$, $x \geq 0$
2. $\exists y \in \mathbb{R}^m$ *with* $A^t y \geq 0$, $b^t y < 0$

Assume $\hat{x}$ satisfies 1. and $\hat{y}$ satisfies 2. Then

$$0 > y^t b = y^t Ax \geq 0$$

Hence, at most one of the statements can hold.

Proof of Farkas Lemma

Now, assume that 1. does not hold.

Consider $S = \{Ax : x \geq 0\}$ so that S closed, convex, $b \notin S$.

We want to show that there is y with $A^t y \geq 0$, $b^t y < 0$.

Let y be a hyperplane that separates b from S. Hence, $y^t b < \alpha$ and $y^t s \geq \alpha$ for all $s \in S$.

$0 \in S \Rightarrow \alpha \leq 0 \Rightarrow y^t b < 0$

$y^t Ax \geq \alpha$ for all $x \geq 0$. Hence, $y^t A \geq 0$ as we can choose x arbitrarily large.

Proof of Farkas Lemma

Now, assume that 1. does not hold.

Consider $S = \{Ax : x \geq 0\}$ so that S closed, convex, $b \notin S$.

We want to show that there is y with $A^t y \geq 0$, $b^t y < 0$.

Let y be a hyperplane that separates b from S. Hence, $y^t b < \alpha$ and $y^t s \geq \alpha$ for all $s \in S$.

$0 \in S \Rightarrow \alpha \leq 0 \Rightarrow y^t b < 0$

$y^t A x \geq \alpha$ for all $x \geq 0$. Hence, $y^t A \geq 0$ as we can choose x arbitrarily large.

Proof of Farkas Lemma

Now, assume that 1. does not hold.

Consider $S = \{Ax : x \geq 0\}$ so that S closed, convex, $b \notin S$.

We want to show that there is y with $A^t y \geq 0$, $b^t y < 0$.

Let y be a hyperplane that separates b from S. Hence, $y^t b < \alpha$ and $y^t s \geq \alpha$ for all $s \in S$.

$0 \in S \Rightarrow \alpha \leq 0 \Rightarrow y^t b < 0$

$y^t Ax \geq \alpha$ for all $x \geq 0$. Hence, $y^t A \geq 0$ as we can choose x arbitrarily large.

Proof of Farkas Lemma

Now, assume that 1. does not hold.

Consider $S = \{Ax : x \geq 0\}$ so that S closed, convex, $b \notin S$.

We want to show that there is y with $A^t y \geq 0$, $b^t y < 0$.

Let y be a hyperplane that separates b from S. Hence, $y^t b < \alpha$ and $y^t s \geq \alpha$ for all $s \in S$.

$0 \in S \Rightarrow \alpha \leq 0 \Rightarrow y^t b < 0$

$y^t A x \geq \alpha$ for all $x \geq 0$. Hence, $y^t A \geq 0$ as we can choose x arbitrarily large.

Proof of Farkas Lemma

Now, assume that 1. does not hold.

Consider $S = \{Ax : x \geq 0\}$ so that S closed, convex, $b \notin S$.

We want to show that there is y with $A^t y \geq 0$, $b^t y < 0$.

Let y be a hyperplane that separates b from S. Hence, $y^t b < \alpha$ and $y^t s \geq \alpha$ for all $s \in S$.

$0 \in S \Rightarrow \alpha \leq 0 \Rightarrow y^t b < 0$

$y^t Ax \geq \alpha$ for all $x \geq 0$. Hence, $y^t A \geq 0$ as we can choose x arbitrarily large.

Proof of Farkas Lemma

Now, assume that 1. does not hold.

Consider $S = \{Ax : x \geq 0\}$ so that S closed, convex, $b \notin S$.

We want to show that there is y with $A^t y \geq 0$, $b^t y < 0$.

Let y be a hyperplane that separates b from S. Hence, $y^t b < \alpha$ and $y^t s \geq \alpha$ for all $s \in S$.

$0 \in S \Rightarrow \alpha \leq 0 \Rightarrow y^t b < 0$

$y^t Ax \geq \alpha$ for all $x \geq 0$. Hence, $y^t A \geq 0$ as we can choose x arbitrarily large.

Proof of Farkas Lemma

Now, assume that 1. does not hold.

Consider $S = \{Ax : x \geq 0\}$ so that S closed, convex, $b \notin S$.

We want to show that there is y with $A^t y \geq 0$, $b^t y < 0$.

Let y be a hyperplane that separates b from S. Hence, $y^t b < \alpha$ and $y^t s \geq \alpha$ for all $s \in S$.

$0 \in S \Rightarrow \alpha \leq 0 \Rightarrow y^t b < 0$

$y^t Ax \geq \alpha$ for all $x \geq 0$. Hence, $y^t A \geq 0$ as we can choose x arbitrarily large.

Proof of Farkas Lemma

Now, assume that 1. does not hold.

Consider $S = \{Ax : x \geq 0\}$ so that S closed, convex, $b \notin S$.

We want to show that there is y with $A^t y \geq 0$, $b^t y < 0$.

Let y be a hyperplane that separates b from S. Hence, $y^t b < \alpha$ and $y^t s \geq \alpha$ for all $s \in S$.

$0 \in S \Rightarrow \alpha \leq 0 \Rightarrow y^t b < 0$

$y^t Ax \geq \alpha$ for all $x \geq 0$. Hence, $y^t A \geq 0$ as we can choose x arbitrarily large.

Lemma 23 (Farkas Lemma; different version)

Let A be an $m \times n$ matrix, $b \in \mathbb{R}^m$. Then exactly one of the following statements holds.

1. $\exists x \in \mathbb{R}^n$ *with* $Ax \leq b$, $x \geq 0$
2. $\exists y \in \mathbb{R}^m$ *with* $A^t y \geq 0$, $b^t y < 0$, $y \geq 0$

Rewrite the conditions:

1. $\exists x \in \mathbb{R}^n$ with $\begin{bmatrix} A & I \end{bmatrix} \cdot \begin{bmatrix} x \\ s \end{bmatrix} = b$, $x \geq 0$, $s \geq 0$
2. $\exists y \in \mathbb{R}^m$ with $\begin{bmatrix} A^t \\ I \end{bmatrix} y \geq 0$, $b^t y < 0$

Lemma 23 (Farkas Lemma; different version)

Let A be an $m \times n$ matrix, $b \in \mathbb{R}^m$. Then exactly one of the following statements holds.

1. $\exists x \in \mathbb{R}^n$ *with* $Ax \le b$, $x \ge 0$
2. $\exists y \in \mathbb{R}^m$ *with* $A^t y \ge 0$, $b^t y < 0$, $y \ge 0$

Rewrite the conditions:

1. $\exists x \in \mathbb{R}^n$ with $\begin{bmatrix} A \; I \end{bmatrix} \cdot \begin{bmatrix} x \\ s \end{bmatrix} = b$, $x \ge 0$, $s \ge 0$
2. $\exists y \in \mathbb{R}^m$ with $\begin{bmatrix} A^t \\ I \end{bmatrix} y \ge 0$, $b^t y < 0$

Proof of Strong Duality

P: $z = \max\{c^t x \mid Ax \le b, x \ge 0\}$

D: $w = \min\{b^t y \mid A^t y \ge c, y \ge 0\}$

Theorem 24 (Strong Duality)

Let P and D be a primal dual pair of linear programs, and let z and w denote the optimal solution to P and D, respectively (i.e., P and D are non-empty). Then

$$z = w \ .$$

Proof of Strong Duality

Proof of Strong Duality

$z \le w$: follows from weak duality

Proof of Strong Duality

$z \leq w$: follows from weak duality

$z \geq w$:

Proof of Strong Duality

$\boldsymbol{z} \leq \boldsymbol{w}$: follows from weak duality

$\boldsymbol{z} \geq \boldsymbol{w}$:
We show $z < \alpha$ implies $w < \alpha$.

Proof of Strong Duality

$\boldsymbol{z} \leq \boldsymbol{w}$: follows from weak duality

$\boldsymbol{z} \geq \boldsymbol{w}$:
We show $z < \alpha$ implies $w < \alpha$.

$$\begin{array}{llrcl} \exists x \in \mathbb{R}^n & & & & \\ & \text{s.t.} & Ax & \leq & b \\ & & -c^t x & \leq & -\alpha \\ & & x & \geq & 0 \end{array}$$

Proof of Strong Duality

$\boldsymbol{z \leq w}$: follows from weak duality

$\boldsymbol{z \geq w}$:
We show $z < \alpha$ implies $w < \alpha$.

$$\begin{array}{lrcl} \exists x \in \mathbb{R}^n & & & \\ \text{s.t.} & Ax & \leq & b \\ & -c^t x & \leq & -\alpha \\ & x & \geq & 0 \end{array}$$

$$\begin{array}{lrcl} \exists y \in \mathbb{R}^m; v \in \mathbb{R} & & & \\ \text{s.t.} & A^t y - cv & \geq & 0 \\ & b^t y - \alpha v & < & 0 \\ & y, v & \geq & 0 \end{array}$$

Proof of Strong Duality

$z \leq w$: follows from weak duality

$z \geq w$:
We show $z < \alpha$ implies $w < \alpha$.

$$\begin{array}{lrcl} \exists x \in \mathbb{R}^n & & & \\ \text{s.t.} & Ax & \leq & b \\ & -c^t x & \leq & -\alpha \\ & x & \geq & 0 \end{array}$$

$$\begin{array}{lrcl} \exists y \in \mathbb{R}^m; v \in \mathbb{R} & & & \\ \text{s.t.} & A^t y - cv & \geq & 0 \\ & b^t y - \alpha v & < & 0 \\ & y, v & \geq & 0 \end{array}$$

From the definition of α we know that the first system is infeasible; hence the second must be feasible.

Proof of Strong Duality

$$\begin{array}{rrcl} \exists y \in \mathbb{R}^m; v \in \mathbb{R} & & & \\ \text{s.t.} & A^t y - v & \geq & 0 \\ & b^t y - \alpha v & < & 0 \\ & y, v & \geq & 0 \end{array}$$

Proof of Strong Duality

$$
\begin{array}{lrcl}
\exists y \in \mathbb{R}^m; v \in \mathbb{R} & & & \\
\text{s.t.} & A^t y - v & \geq & 0 \\
 & b^t y - \alpha v & < & 0 \\
 & y, v & \geq & 0
\end{array}
$$

If the solution y, v has $v = 0$ we have that

$$
\begin{array}{lrcl}
\exists y \in \mathbb{R}^m & & & \\
\text{s.t.} & A^t y & \geq & 0 \\
 & b^t y & < & 0 \\
 & y & \geq & 0
\end{array}
$$

is feasible.

Proof of Strong Duality

$$\begin{array}{lrcl} \exists y \in \mathbb{R}^m; v \in \mathbb{R} & & & \\ \text{s.t.} & A^t y - v & \geq & 0 \\ & b^t y - \alpha v & < & 0 \\ & y, v & \geq & 0 \end{array}$$

If the solution y, v has $v = 0$ we have that

$$\begin{array}{lrcl} \exists y \in \mathbb{R}^m & & & \\ \text{s.t.} & A^t y & \geq & 0 \\ & b^t y & < & 0 \\ & y & \geq & 0 \end{array}$$

is feasible. By Farkas lemma this gives that LP P is infeasible. Contradiction to the assumption of the lemma.

Proof of Strong Duality

Hence, there exists a solution y, v with $v > 0$.

We can rescale this solution (scaling both y and v) s.t. $v = 1$.

Then y is feasible for the dual but $b^t y < \alpha$. This means that $w < \alpha$.

Proof of Strong Duality

Hence, there exists a solution y, v with $v > 0$.

We can rescale this solution (scaling both y and v) s.t. $v = 1$.

Then y is feasible for the dual but $b^t y < \alpha$. This means that $w < \alpha$.

Proof of Strong Duality

Hence, there exists a solution y, v with $v > 0$.

We can rescale this solution (scaling both y and v) s.t. $v = 1$.

Then y is feasible for the dual but $b^t y < \alpha$. This means that $w < \alpha$.

Proof of Strong Duality

Hence, there exists a solution y, v with $v > 0$.

We can rescale this solution (scaling both y and v) s.t. $v = 1$.

Then y is feasible for the dual but $b^t y < \alpha$. This means that $w < \alpha$.

Fundamental Questions

Definition 25 (Linear Programming Problem (LP))

Let $A \in \mathbb{Q}^{m\times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$ s.t. $Ax = b$, $x \geq 0$, $c^t x \geq \alpha$?

Questions:

- Is LP in NP?
- Is LP in co-NP? yes!
- Is LP in P?

Proof:

- Given a primal maximization problem P and a parameter α. Suppose that $\alpha > \mathrm{opt}(P)$.
- We can prove this by providing an optimal basis for the dual.
- A verifier can check that the associated dual solution fulfills all dual constraints and that it has dual cost $< \alpha$.

Fundamental Questions

Definition 25 (Linear Programming Problem (LP))

Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$ s.t. $Ax = b$, $x \geq 0$, $c^t x \geq \alpha$?

Questions:

- Is LP in NP?
- Is LP in co-NP? yes!
- Is LP in P?

Proof:

- Given a primal maximization problem P and a parameter α. Suppose that $\alpha > \text{opt}(P)$.
- We can prove this by providing an optimal basis for the dual.
- A verifier can check that the associated dual solution fulfills all dual constraints and that it has dual cost $< \alpha$.

Fundamental Questions

Definition 25 (Linear Programming Problem (LP))

Let $A \in \mathbb{Q}^{m \times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$ s.t. $Ax = b$, $x \geq 0$, $c^t x \geq \alpha$?

Questions:

- Is LP in NP?
- Is LP in co-NP? yes!
- Is LP in P?

Proof:

- Given a primal maximization problem P and a parameter α. Suppose that $\alpha > \mathrm{opt}(P)$.
- We can prove this by providing an optimal basis for the dual.
- A verifier can check that the associated dual solution fulfills all dual constraints and that it has dual cost $< \alpha$.

Fundamental Questions

Definition 25 (Linear Programming Problem (LP))

Let $A \in \mathbb{Q}^{m\times n}$, $b \in \mathbb{Q}^m$, $c \in \mathbb{Q}^n$, $\alpha \in \mathbb{Q}$. Does there exist $x \in \mathbb{Q}^n$ s.t. $Ax = b$, $x \geq 0$, $c^t x \geq \alpha$?

Questions:

- Is LP in NP?
- Is LP in co-NP? yes!
- Is LP in P?

Proof:

- Given a primal maximization problem P and a parameter α. Suppose that $\alpha > \text{opt}(P)$.
- We can prove this by providing an optimal basis for the dual.
- A verifier can check that the associated dual solution fulfills all dual constraints and that it has dual cost $< \alpha$.

Complementary Slackness

Lemma 26

Assume a linear program $P = \max\{c^t x \mid Ax \le b; x \ge 0\}$ has solution x^* and its dual $D = \min\{b^t y \mid A^t y \ge c; y \ge 0\}$ has solution y^*.

1. *If $x_j^* > 0$ then the j-th constraint in D is tight.*
2. *If the j-th constraint in D is not tight than $x_j^* = 0$.*
3. *If $y_i^* > 0$ then the i-th constraint in P is tight.*
4. *If the i-th constraint in P is not tight than $y_i^* = 0$.*

Complementary Slackness

Lemma 26

Assume a linear program $P = \max\{c^t x \mid Ax \le b; x \ge 0\}$ has solution x^* and its dual $D = \min\{b^t y \mid A^t y \ge c; y \ge 0\}$ has solution y^*.

1. *If $x_j^* > 0$ then the j-th constraint in D is tight.*
2. *If the j-th constraint in D is not tight than $x_j^* = 0$.*
3. *If $y_i^* > 0$ then the i-th constraint in P is tight.*
4. *If the i-th constraint in P is not tight than $y_i^* = 0$.*

If we say that a variable x_j^* (y_i^*) has slack if $x_j^* > 0$ ($y_i^* > 0$), (i.e., the corresponding variable restriction is not tight) and a contraint has slack if it is not tight, then the above says that for a primal-dual solution pair it is not possible that a constraint **and** its corresponding (dual) variable has slack.

Proof: Complementary Slackness

Analogous to the proof of weak duality we obtain

$$c^t x^* \le {y^*}^t A x^* \le b^t y^*$$

Proof: Complementary Slackness

Analogous to the proof of weak duality we obtain

$$c^t x^* \le y^{*t} A x^* \le b^t y^*$$

Because of strong duality we then get

$$c^t x^* = y^{*t} A x^* = b^t y^*$$

This gives e.g.

$$\sum_j (y^t A - c^t)_j x_j^* = 0$$

Proof: Complementary Slackness

Analogous to the proof of weak duality we obtain

$$c^t x^* \leq y^{*t} A x^* \leq b^t y^*$$

Because of strong duality we then get

$$c^t x^* = y^{*t} A x^* = b^t y^*$$

This gives e.g.

$$\sum_j (y^t A - c^t)_j x_j^* = 0$$

From the constraint of the dual it follows that $y^t A \geq c^t$. Hence the left hand side is a sum over the product of non-negative numbers. Hence, if e.g. $(y^t A - c^t)_j > 0$ (the j-th constraint in the dual is not tight) then $x_j = 0$ (2.). The result for (1./3./4.) follows similarly.

Interpretation of Dual Variables

- Brewer: find mix of ale and beer that maximizes profits

$$\begin{array}{rrcrl} \text{max} & 13a & + & 23b & \\ \text{s.t.} & 5a & + & 15b & \le 480 \\ & 4a & + & 4b & \le 160 \\ & 35a & + & 20b & \le 1190 \\ & & & a,b & \ge 0 \end{array}$$

- Entrepeneur: buy resources from brewer at minimum cost
  C, H, M: unit price for corn, hops and malt.

$$\begin{array}{rrcrcrl} \text{min} & 480C & + & 160H & + & 1190M & \\ \text{s.t.} & 5C & + & 4H & + & 35M & \ge 13 \\ & 15C & + & 4H & + & 20M & \ge 23 \\ & & & & & C,H,M & \ge 0 \end{array}$$

Note that brewer won't sell (at least not all) if e.g. $5C + 4H + 35M < 13$ as then brewing ale would be advantageous.

Interpretation of Dual Variables

- Brewer: find mix of ale and beer that maximizes profits

$$
\begin{array}{lrcrl}
\max & 13a & + & 23b & \\
\text{s.t.} & 5a & + & 15b & \leq 480 \\
 & 4a & + & 4b & \leq 160 \\
 & 35a & + & 20b & \leq 1190 \\
 & & & a, b & \geq 0
\end{array}
$$

- Entrepeneur: buy resources from brewer at minimum cost
  C, H, M: unit price for corn, hops and malt.

$$
\begin{array}{lrcrcrl}
\min & 480C & + & 160H & + & 1190M & \\
\text{s.t.} & 5C & + & 4H & + & 35M & \geq 13 \\
 & 15C & + & 4H & + & 20M & \geq 23 \\
 & & & & & C, H, M & \geq 0
\end{array}
$$

Note that brewer won't sell (at least not all) if e.g.
$5C + 4H + 35M < 13$ as then brewing ale would be advantageous.

Interpretation of Dual Variables

- Brewer: find mix of ale and beer that maximizes profits

$$\begin{array}{rrcrl} \max & 13a & + & 23b & \\ \text{s.t.} & 5a & + & 15b & \le 480 \\ & 4a & + & 4b & \le 160 \\ & 35a & + & 20b & \le 1190 \\ & & & a,b & \ge 0 \end{array}$$

- Entrepeneur: buy resources from brewer at minimum cost
  C, H, M: unit price for corn, hops and malt.

$$\begin{array}{rrcrcrl} \min & 480C & + & 160H & + & 1190M & \\ \text{s.t.} & 5C & + & 4H & + & 35M & \ge 13 \\ & 15C & + & 4H & + & 20M & \ge 23 \\ & & & & & C,H,M & \ge 0 \end{array}$$

Note that brewer won’t sell (at least not all) if e.g.
$5C + 4H + 35M < 13$ as then brewing ale would be advantageous.

Interpretation of Dual Variables

Marginal Price:

- How much money is the brewer willing to pay for additional amount of Corn, Hops, or Malt?
- We are interested in the marginal price, i.e., what happens if we increase the amount of Corn, Hops, and Malt by ε_C, ε_H, and ε_M, respectively.

The profit increases to $\max\{c^t x \mid Ax \le b + \varepsilon; x \ge 0\}$. Because of strong duality this is equal to

$$\begin{array}{lrcl} \min & (b^t + \epsilon^t)y & & \\ \text{s.t.} & A^t y & \ge & c \\ & y & \ge & 0 \end{array}$$

Interpretation of Dual Variables

Marginal Price:

- How much money is the brewer willing to pay for additional amount of Corn, Hops, or Malt?
- We are interested in the marginal price, i.e., what happens if we increase the amount of Corn, Hops, and Malt by ε_C, ε_H, and ε_M, respectively.

The profit increases to $\max\{c^t x \mid Ax \leq b + \varepsilon; x \geq 0\}$. Because of strong duality this is equal to

$$\begin{array}{lrcl} \min & (b^t + \epsilon^t)y & & \\ \text{s.t.} & A^t y & \geq & c \\ & y & \geq & 0 \end{array}$$

Interpretation of Dual Variables

Marginal Price:

- How much money is the brewer willing to pay for additional amount of Corn, Hops, or Malt?
- We are interested in the marginal price, i.e., what happens if we increase the amount of Corn, Hops, and Malt by ε_C, ε_H, and ε_M, respectively.

The profit increases to $\max\{c^t x \mid Ax \le b + \varepsilon; x \ge 0\}$. Because of strong duality this is equal to

$$\begin{array}{lrcl} \min & (b^t + \epsilon^t)y & & \\ \text{s.t.} & A^t y & \ge & c \\ & y & \ge & 0 \end{array}$$

Interpretation of Dual Variables

Marginal Price:

- How much money is the brewer willing to pay for additional amount of Corn, Hops, or Malt?
- We are interested in the marginal price, i.e., what happens if we increase the amount of Corn, Hops, and Malt by ε_C, ε_H, and ε_M, respectively.

The profit increases to $\max\{c^t x \mid Ax \leq b + \varepsilon; x \geq 0\}$. Because of strong duality this is equal to

$$\begin{array}{llcl} \min & (b^t + \epsilon^t)y & & \\ \text{s.t.} & A^t y & \geq & c \\ & y & \geq & 0 \end{array}$$

Interpretation of Dual Variables

If ϵ is "small" enough then the optimum dual solution y^* might not change. Therefore the profit increases by $\sum_i \varepsilon_i y_i^*$.

Therefore we can interpret the dual variables as marginal prices.

Note that with this interpretation, complementary slackness becomes obvious.

- If the brewer has slack of some resource (e.g. corn) then he is not willing to pay anything for it (corresponding dual variable is zero).
- If the dual variable for some resource is non-zero, then an increase of this resource increases the profit of the brewer. Hence, it makes no sense to have left-overs of this resource. Therefore its slack must be zero.

Interpretation of Dual Variables

If ϵ is "small" enough then the optimum dual solution y^* might not change. Therefore the profit increases by $\sum_i \varepsilon_i y_i^*$.

Therefore we can interpret the dual variables as marginal prices.

Note that with this interpretation, complementary slackness becomes obvious.

- If the brewer has slack of some resource (e.g. corn) then he is not willing to pay anything for it (corresponding dual variable is zero).
- If the dual variable for some resource is non-zero, then an increase of this resource increases the profit of the brewer. Hence, it makes no sense to have left-overs of this resource. Therefore its slack must be zero.

Interpretation of Dual Variables

If ϵ is "small" enough then the optimum dual solution y^* might not change. Therefore the profit increases by $\sum_i \varepsilon_i y_i^*$.

Therefore we can interpret the dual variables as marginal prices.

Note that with this interpretation, complementary slackness becomes obvious.

- If the brewer has slack of some resource (e.g. corn) then he is not willing to pay anything for it (corresponding dual variable is zero).
- If the dual variable for some resource is non-zero, then an increase of this resource increases the profit of the brewer. Hence, it makes no sense to have left-overs of this resource. Therefore its slack must be zero.

Interpretation of Dual Variables

If ϵ is "small" enough then the optimum dual solution y^* might not change. Therefore the profit increases by $\sum_i \varepsilon_i y_i^*$.

Therefore we can interpret the dual variables as marginal prices.

Note that with this interpretation, complementary slackness becomes obvious.

- If the brewer has slack of some resource (e.g. corn) then he is not willing to pay anything for it (corresponding dual variable is zero).
- If the dual variable for some resource is non-zero, then an increase of this resource increases the profit of the brewer. Hence, it makes no sense to have left-overs of this resource. Therefore its slack must be zero.

Interpretation of Dual Variables

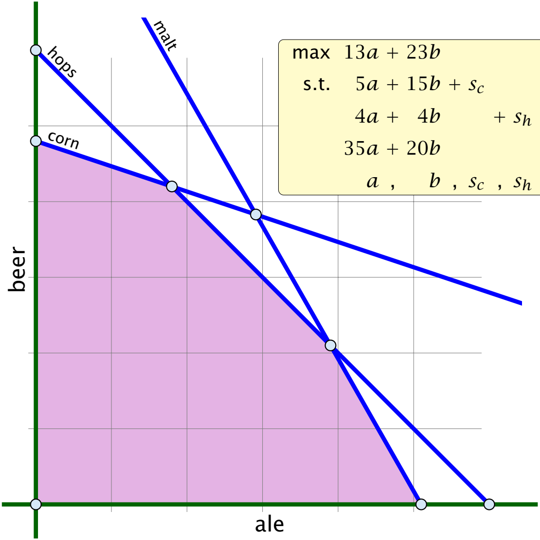
If ϵ is "small" enough then the optimum dual solution y^* might not change. Therefore the profit increases by $\sum_i \varepsilon_i y_i^*$.

Therefore we can interpret the dual variables as marginal prices.

Note that with this interpretation, complementary slackness becomes obvious.

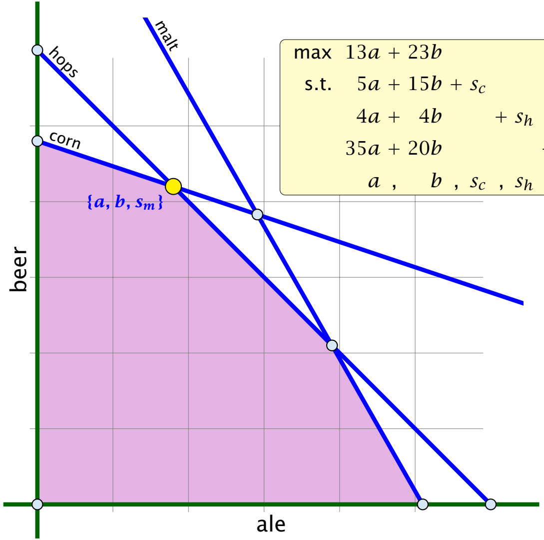
- If the brewer has slack of some resource (e.g. corn) then he is not willing to pay anything for it (corresponding dual variable is zero).
- If the dual variable for some resource is non-zero, then an increase of this resource increases the profit of the brewer. Hence, it makes no sense to have left-overs of this resource. Therefore its slack must be zero.

Example



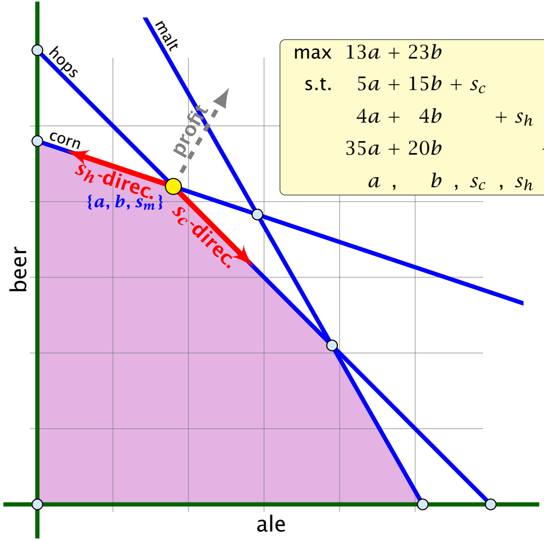
$$
\begin{array}{llllllll}
\text{max} & 13a + 23b & & & & & & \\
\text{s.t.} & 5a + 15b & + s_c & & & = & 480 \\
 & 4a + 4b & & + s_h & & = & 160 \\
 & 35a + 20b & & & + s_m & = & 1190 \\
 & a \ , \ b & , s_c & , s_h & , s_m & \geq & 0
\end{array}
$$

Example



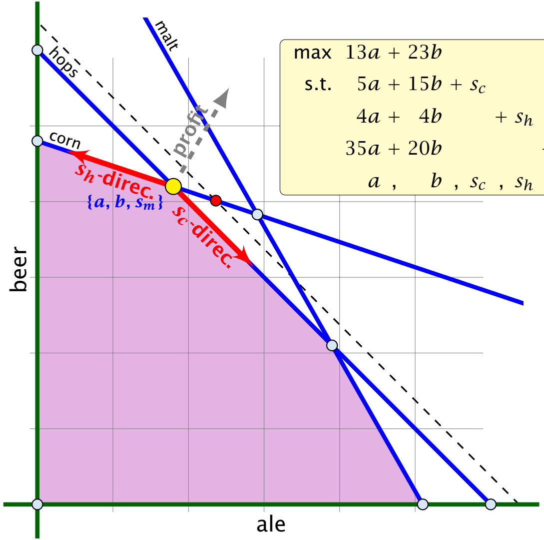
$$
\begin{array}{lrcl}
\max & 13a + 23b & & \\
\text{s.t.} & 5a + 15b + s_c & = & 480 \\
 & 4a + 4b + s_h & = & 160 \\
 & 35a + 20b + s_m & = & 1190 \\
 & a\ ,\ b\ ,\ s_c\ ,\ s_h\ ,\ s_m & \geq & 0
\end{array}
$$

Example



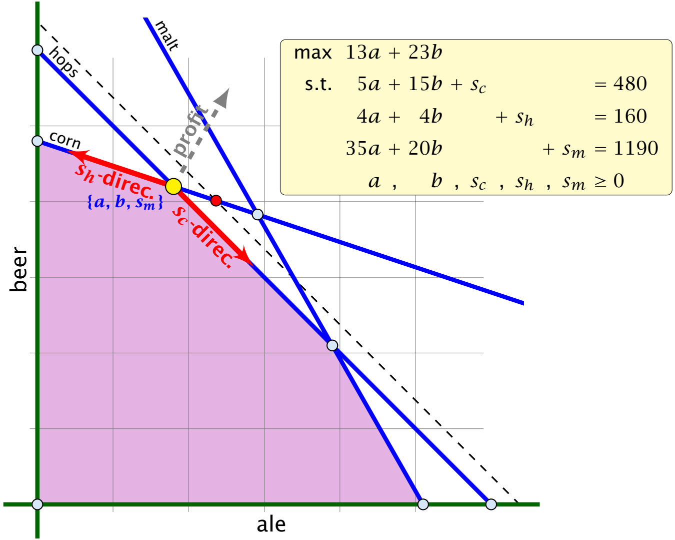
$$
\begin{array}{lrcrcrcrcrcl}
\max & 13a & + & 23b & & & & & & & & \\
\text{s.t.} & 5a & + & 15b & + & s_c & & & & & = & 480 \\
 & 4a & + & 4b & & & + & s_h & & & = & 160 \\
 & 35a & + & 20b & & & & & + & s_m & = & 1190 \\
 & a & , & b & , & s_c & , & s_h & , & s_m & \geq & 0
\end{array}
$$

Example



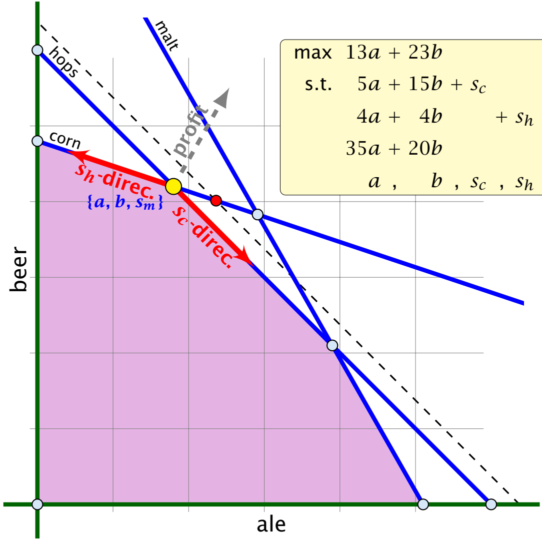
$$
\begin{array}{llllllll}
\text{max} & 13a + 23b & & & & & & \\
\text{s.t.} & 5a + 15b & + s_c & & & = 480 \\
 & 4a + 4b & & + s_h & & = 160 \\
 & 35a + 20b & & & + s_m & = 1190 \\
 & a \;,\; b & ,\, s_c & ,\, s_h & ,\, s_m & \geq 0
\end{array}
$$

Example



$$
\begin{array}{llllllll}
\max & 13a + 23b & & & & & \\
\text{s.t.} & 5a + 15b & + s_c & & & = & 480 \\
 & 4a + 4b & & + s_h & & = & 160 \\
 & 35a + 20b & & & + s_m & = & 1190 \\
 & a \;,\; b & , s_c & , s_h & , s_m & \geq & 0
\end{array}
$$

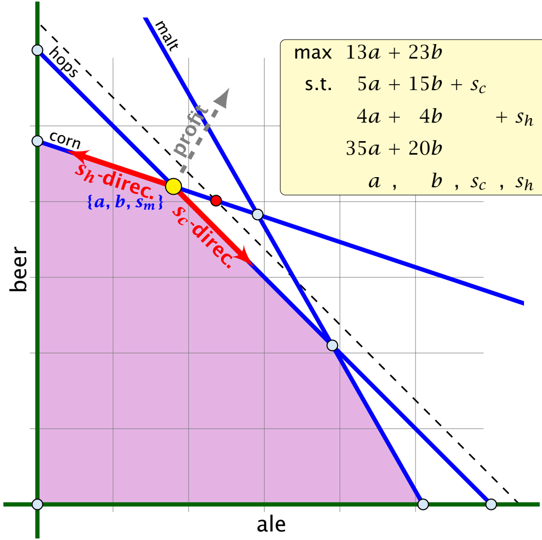
Example



$$
\begin{array}{lrcl}
\max & 13a + 23b & & \\
\text{s.t.} & 5a + 15b + s_c & = & 480 \\
& 4a + 4b + s_h & = & 160 \\
& 35a + 20b + s_m & = & 1190 \\
& a, b, s_c, s_h, s_m & \geq & 0
\end{array}
$$

The change in profit when increasing hops by one unit is $= c_B^t A_B^{-1} e_h$.

Example



$$\begin{array}{llllllll}
\text{max} & 13a + 23b & & & & & & \\
\text{s.t.} & 5a + 15b & + s_c & & & = 480 \\
 & 4a + 4b & & + s_h & & = 160 \\
 & 35a + 20b & & & + s_m & = 1190 \\
 & a, \; b, & s_c, & s_h, & s_m & \geq 0
\end{array}$$

The change in profit when increasing hops by one unit is

$$= \underbrace{c_B^t A_B^{-1}}_{y^*} e_h .$$

Of course, the previous argument about the increase in the primal objective only holds for the non-degenerate case.

If the optimum basis is degenerate then increasing the supply of one resource may not allow the objective value to increase.

Flows

Definition 27

An (s,t)-flow in a (complete) directed graph $G = (V, V \times V, c)$ is a function $f : V \times V \mapsto \mathbb{R}_0^+$ that satisfies

1. For each edge (x,y)

$$0 \le f_{xy} \le c_{xy} \ .$$

   (capacity constraints)

2. For each $v \in V \setminus \{s,t\}$

$$\sum_x f_{vx} = \sum_x f_{xv} \ .$$

   (flow conservation constraints)

Flows

Definition 27

An (s, t)-flow in a (complete) directed graph $G = (V, V \times V, c)$ is a function $f : V \times V \mapsto \mathbb{R}_0^+$ that satisfies

1. For each edge (x, y)

$$0 \leq f_{xy} \leq c_{xy} \ .$$

   (capacity constraints)

2. For each $v \in V \setminus \{s, t\}$

$$\sum_x f_{vx} = \sum_x f_{xv} \ .$$

   (flow conservation constraints)

Flows

Definition 28

The value of an (s,t)-flow f is defined as

$$\mathrm{val}(f) = \sum_x f_{sx} - \sum_x f_{xs} \quad .$$

Maximum Flow Problem:

Find an (s,t)-flow with maximum value.

Flows

Definition 28

The value of an (s,t)-flow f is defined as

$$\text{val}(f) = \sum_x f_{sx} - \sum_x f_{xs} \quad .$$

Maximum Flow Problem:

Find an (s,t)-flow with maximum value.

LP-Formulation of Maxflow

$$
\begin{array}{llrcll}
\max & & \sum_z f_{sz} - \sum_z f_{zs} & & & \\
\text{s.t.} & \forall (z,w) \in V \times V & f_{zw} & \le & c_{zw} & \ell_{zw} \\
 & \forall w \neq s,t & \sum_z f_{zw} - \sum_z f_{wz} & = & 0 & p_w \\
 & & f_{zw} & \ge & 0 &
\end{array}
$$

$$
\begin{array}{llrcl}
\min & & \sum_{(xy)} c_{xy}\ell_{xy} & & \\
\text{s.t.} & f_{xy}\ (x,y \neq s,t): & 1\ell_{xy} - 1p_x + 1p_y & \ge & 0 \\
 & f_{sy}\ (y \neq s,t): & 1\ell_{sy} \qquad + 1p_y & \ge & 1 \\
 & f_{xs}\ (x \neq s,t): & 1\ell_{xs} - 1p_x \qquad & \ge & -1 \\
 & f_{ty}\ (y \neq s,t): & 1\ell_{ty} \qquad + 1p_y & \ge & 0 \\
 & f_{xt}\ (x \neq s,t): & 1\ell_{xt} - 1p_x \qquad & \ge & 0 \\
 & f_{st}: & 1\ell_{st} & \ge & 1 \\
 & f_{ts}: & 1\ell_{ts} & \ge & -1 \\
 & & \ell_{xy} & \ge & 0
\end{array}
$$

LP-Formulation of Maxflow

$$
\begin{array}{llrcl}
\min & & \sum_{(xy)} c_{xy}\ell_{xy} & & \\
\text{s.t.} & f_{xy}\ (x,y \neq s,t): & 1\ell_{xy} - 1p_x + 1p_y & \geq & 0 \\
 & f_{sy}\ (y \neq s,t): & 1\ell_{sy} - \ \ 1 + 1p_y & \geq & 0 \\
 & f_{xs}\ (x \neq s,t): & 1\ell_{xs} - 1p_x + \ \ 1 & \geq & 0 \\
 & f_{ty}\ (y \neq s,t): & 1\ell_{ty} - \ \ 0 + 1p_y & \geq & 0 \\
 & f_{xt}\ (x \neq s,t): & 1\ell_{xt} - 1p_x + \ \ 0 & \geq & 0 \\
 & f_{st}: & 1\ell_{st} - \ \ 1 + \ \ 0 & \geq & 0 \\
 & f_{ts}: & 1\ell_{ts} - \ \ 0 + \ \ 1 & \geq & 0 \\
 & & \ell_{xy} & \geq & 0
\end{array}
$$

LP-Formulation of Maxflow

$$
\begin{array}{llrcl}
\min & & \sum_{(xy)} c_{xy}\ell_{xy} & & \\
\text{s.t.} & f_{xy}\ (x,y \neq s,t): & 1\ell_{xy} - 1p_x + 1p_y & \geq & 0 \\
 & f_{sy}\ (y \neq s,t): & 1\ell_{sy} - p_s + 1p_y & \geq & 0 \\
 & f_{xs}\ (x \neq s,t): & 1\ell_{xs} - 1p_x + p_s & \geq & 0 \\
 & f_{ty}\ (y \neq s,t): & 1\ell_{ty} - p_t + 1p_y & \geq & 0 \\
 & f_{xt}\ (x \neq s,t): & 1\ell_{xt} - 1p_x + p_t & \geq & 0 \\
 & f_{st}: & 1\ell_{st} - p_s + p_t & \geq & 0 \\
 & f_{ts}: & 1\ell_{ts} - p_t + p_s & \geq & 0 \\
 & & \ell_{xy} & \geq & 0
\end{array}
$$

with $p_t = 0$ and $p_s = 1$.

LP-Formulation of Maxflow

$$
\begin{array}{llrcl}
\text{min} & & \sum_{(xy)} c_{xy}\ell_{xy} & & \\
\text{s.t.} & f_{xy}: & 1\ell_{xy} - 1p_x + 1p_y & \geq & 0 \\
& & \ell_{xy} & \geq & 0 \\
& & p_s & = & 1 \\
& & p_t & = & 0
\end{array}
$$

We can interpret the ℓ_{xy} value as assigning a length to every edge.

The value p_x for a variable, then can be seen as the distance of x to t (where the distance from s to t is required to be 1 since $p_s = 1$).

The constraint $p_x \leq \ell_{xy} + p_y$ then simply follows from triangle inequality ($d(x,t) \leq d(x,y) + d(y,t) \Rightarrow d(x,t) \leq \ell_{xy} + d(y,t)$).

LP-Formulation of Maxflow

$$
\begin{array}{llrcl}
\text{min} & & \sum_{(xy)} c_{xy}\ell_{xy} & & \\
\text{s.t.} & f_{xy}: & 1\ell_{xy} - 1p_x + 1p_y & \geq & 0 \\
 & & \ell_{xy} & \geq & 0 \\
 & & p_s & = & 1 \\
 & & p_t & = & 0
\end{array}
$$

We can interpret the ℓ_{xy} value as assigning a length to every edge.

The value p_x for a variable, then can be seen as the distance of x to t (where the distance from s to t is required to be 1 since $p_s = 1$).

The constraint $p_x \leq \ell_{xy} + p_y$ then simply follows from triangle inequality ($d(x,t) \leq d(x,y) + d(y,t) \Rightarrow d(x,t) \leq \ell_{xy} + d(y,t)$).

LP-Formulation of Maxflow

$$\begin{array}{llrcl} \min & & \sum_{(xy)} c_{xy}\ell_{xy} & & \\ \text{s.t.} & f_{xy}: & 1\ell_{xy} - 1p_x + 1p_y & \geq & 0 \\ & & \ell_{xy} & \geq & 0 \\ & & p_s & = & 1 \\ & & p_t & = & 0 \end{array}$$

We can interpret the ℓ_{xy} value as assigning a length to every edge.

The value p_x for a variable, then can be seen as the distance of x to t (where the distance from s to t is required to be 1 since $p_s = 1$).

The constraint $p_x \leq \ell_{xy} + p_y$ then simply follows from triangle inequality ($d(x,t) \leq d(x,y) + d(y,t) \Rightarrow d(x,t) \leq \ell_{xy} + d(y,t)$).

LP-Formulation of Maxflow

$$
\begin{array}{llrcl}
\text{min} & & \sum_{(xy)} c_{xy}\ell_{xy} & & \\
\text{s.t.} & f_{xy}: & 1\ell_{xy} - 1p_x + 1p_y & \geq & 0 \\
 & & \ell_{xy} & \geq & 0 \\
 & & p_s & = & 1 \\
 & & p_t & = & 0
\end{array}
$$

We can interpret the ℓ_{xy} value as assigning a length to every edge.

The value p_x for a variable, then can be seen as the distance of x to t (where the distance from s to t is required to be 1 since $p_s = 1$).

The constraint $p_x \leq \ell_{xy} + p_y$ then simply follows from triangle inequality ($d(x,t) \leq d(x,y) + d(y,t) \Rightarrow d(x,t) \leq \ell_{xy} + d(y,t)$).

One can show that there is an optimum LP-solution for the dual problem that gives an integral assignment of variables.

This means $p_x = 1$ or $p_x = 0$ for our case. This gives rise to a cut in the graph with vertices having value 1 on one side and the other vertices on the other side. The objective function then evaluates the capacity of this cut.

This shows that the Maxflow/Mincut theorem follows from linear programming duality.

One can show that there is an optimum LP-solution for the dual problem that gives an integral assignment of variables.

This means $p_x = 1$ or $p_x = 0$ for our case. This gives rise to a cut in the graph with vertices having value 1 on one side and the other vertices on the other side. The objective function then evaluates the capacity of this cut.

This shows that the Maxflow/Mincut theorem follows from linear programming duality.

One can show that there is an optimum LP-solution for the dual problem that gives an integral assignment of variables.

This means $p_x = 1$ or $p_x = 0$ for our case. This gives rise to a cut in the graph with vertices having value 1 on one side and the other vertices on the other side. The objective function then evaluates the capacity of this cut.

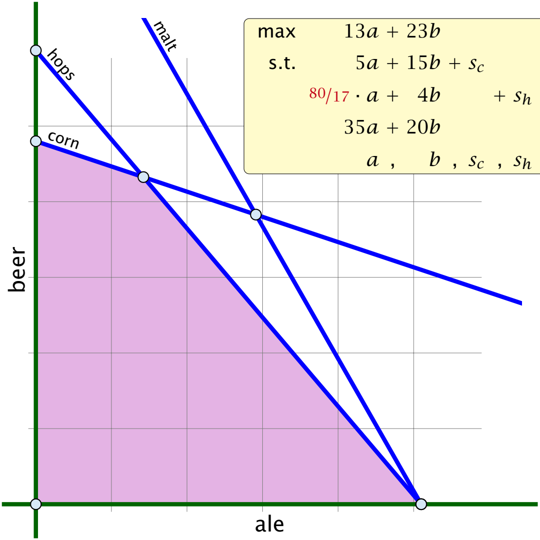
This shows that the Maxflow/Mincut theorem follows from linear programming duality.

Degeneracy Revisited

Degeneracy Revisited

If a basis variable is 0 in the basic feasible solution then we may not make progress during an iteration of simplex.

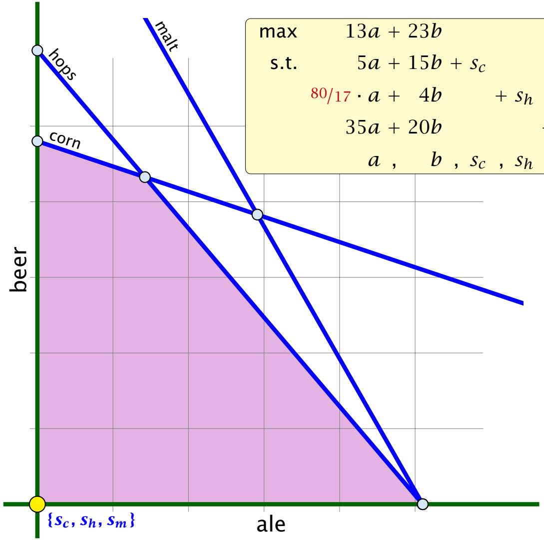
Degenerate Example



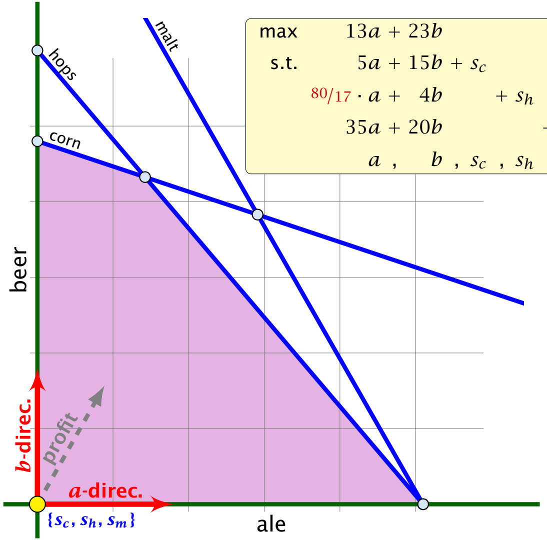
$$
\begin{array}{llll}
\text{max} & 13a + 23b & & \\
\text{s.t.} & 5a + 15b + s_c & & = 480 \\
 & \color{red}{80/17} \cdot a + 4b & + s_h & = 160 \\
 & 35a + 20b & + s_m & = 1190 \\
 & a, \ b, \ s_c, \ s_h, \ s_m \geq 0 & &
\end{array}
$$

Degenerate Example

$$
\begin{array}{lrl}
\text{max} & 13a + 23b & \\
\text{s.t.} & 5a + 15b + s_c & = 480 \\
 & 80/17 \cdot a + 4b + s_h & = 160 \\
 & 35a + 20b + s_m & = 1190 \\
 & a, b, s_c, s_h, s_m & \geq 0
\end{array}
$$

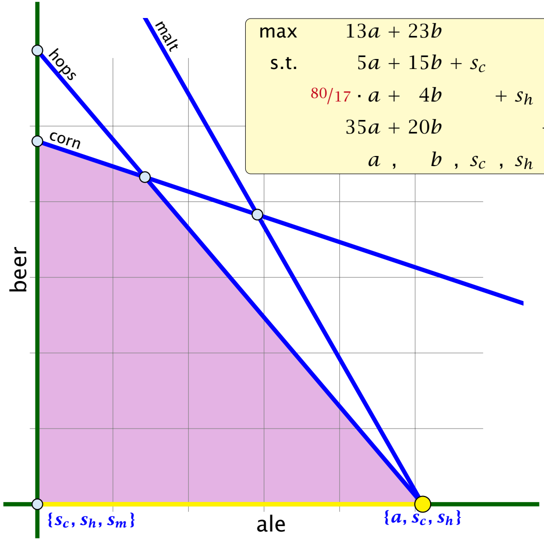


Degenerate Example



$$
\begin{array}{lrcl}
\text{max} & 13a + 23b & & \\
\text{s.t.} & 5a + 15b + s_c & = & 480 \\
 & {}^{80}/_{17} \cdot a + 4b + s_h & = & 160 \\
 & 35a + 20b + s_m & = & 1190 \\
 & a\ ,\ b\ ,\ s_c\ ,\ s_h\ ,\ s_m & \geq & 0
\end{array}
$$

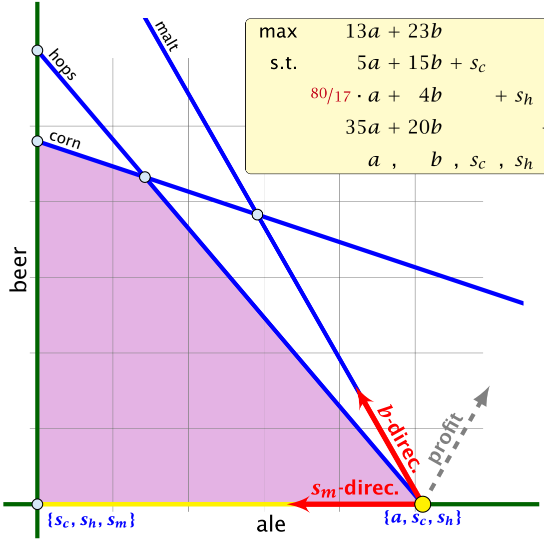
Degenerate Example



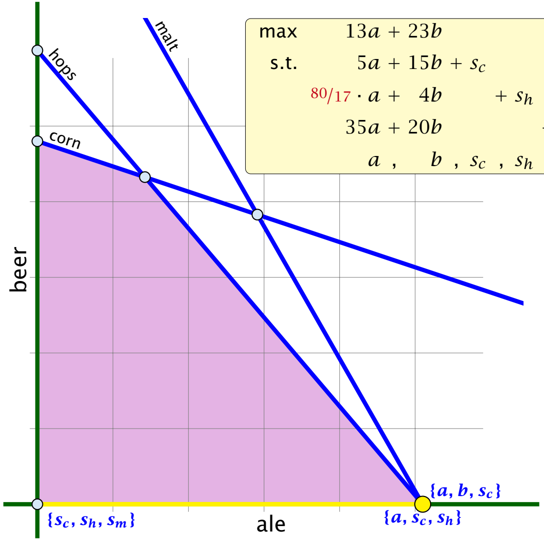
$$
\begin{array}{lrcl}
\max & 13a + 23b & & \\
\text{s.t.} & 5a + 15b + s_c & = & 480 \\
 & \tfrac{80}{17} \cdot a + 4b + s_h & = & 160 \\
 & 35a + 20b + s_m & = & 1190 \\
 & a, b, s_c, s_h, s_m & \geq & 0
\end{array}
$$

Degenerate Example

$$
\begin{array}{lrcl}
\max & 13a + 23b & & \\
\text{s.t.} & 5a + 15b + s_c & = & 480 \\
 & {}^{80}/_{17} \cdot a + 4b + s_h & = & 160 \\
 & 35a + 20b + s_m & = & 1190 \\
 & a\ ,\ b\ ,\ s_c\ ,\ s_h\ ,\ s_m & \geq & 0
\end{array}
$$



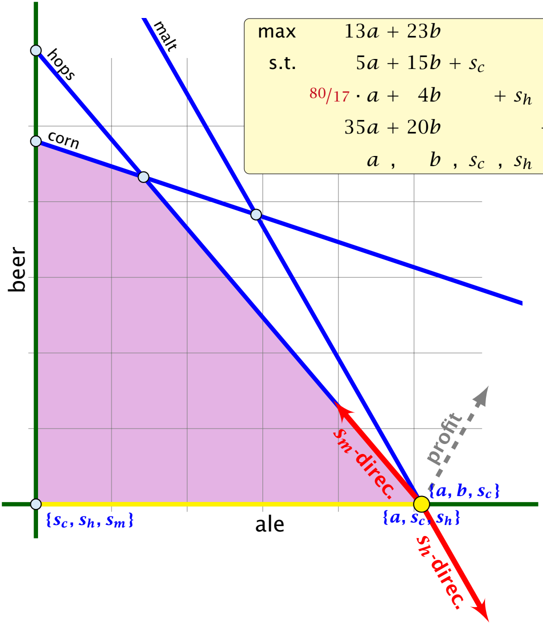
Degenerate Example



$$
\begin{array}{llcl}
\max & 13a + 23b & & \\
\text{s.t.} & 5a + 15b + s_c & = & 480 \\
 & \tfrac{80}{17} \cdot a + 4b + s_h & = & 160 \\
 & 35a + 20b + s_m & = & 1190 \\
 & a, \; b, \; s_c, \; s_h, \; s_m & \geq & 0
\end{array}
$$

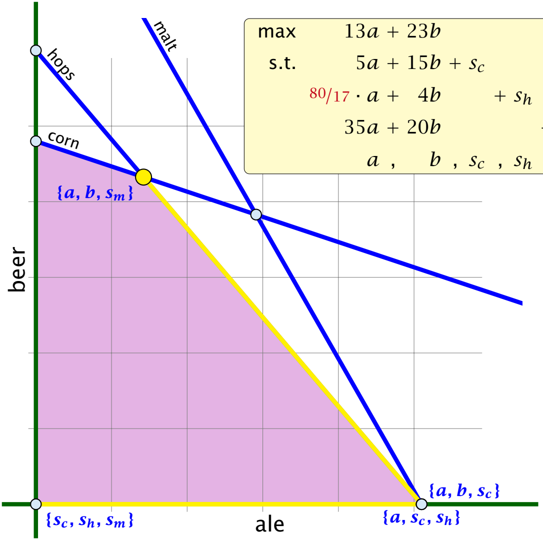
Degenerate Example

$$
\begin{array}{llllllll}
\max & 13a + 23b & & & & & & \\
\text{s.t.} & 5a + 15b & + s_c & & & & = 480 \\
& \color{red}{80/17} \cdot a + 4b & & + s_h & & & = 160 \\
& 35a + 20b & & & + s_m & & = 1190 \\
& a\;,\; b & ,\; s_c & ,\; s_h & ,\; s_m & & \geq 0
\end{array}
$$

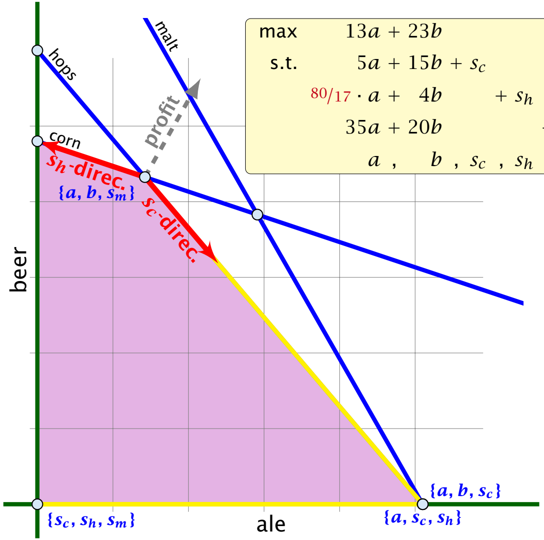


Degenerate Example

$$\begin{array}{llllllll}
\text{max} & 13a & + 23b & & & & & \\
\text{s.t.} & 5a & + 15b & + s_c & & & = & 480 \\
 & 80/17 \cdot a & + 4b & & + s_h & & = & 160 \\
 & 35a & + 20b & & & + s_m & = & 1190 \\
 & a, & b, & s_c, & s_h, & s_m & \geq & 0
\end{array}$$



Degenerate Example



$$
\begin{array}{llllllll}
\text{max} & 13a + 23b & & & & & & \\
\text{s.t.} & 5a + 15b & + s_c & & & = 480 \\
& \tfrac{80}{17} \cdot a + 4b & & + s_h & & = 160 \\
& 35a + 20b & & & + s_m & = 1190 \\
& a \;,\; b \;,\; s_c \;,\; s_h \;,\; s_m \geq 0 & & & & &
\end{array}
$$

Degeneracy Revisited

If a basis variable is 0 in the basic feasible solution then we may not make progress during an iteration of simplex.

Degeneracy Revisited

If a basis variable is 0 in the basic feasible solution then we may not make progress during an iteration of simplex.

Idea:
Given feasible $\mathrm{LP} := \max\{c^t x, Ax = b; x \geq 0\}$. Change it into $\mathrm{LP}' := \max\{c^t x, Ax = b', x \geq 0\}$ such that

- LP' is feasible
- If a set B of basis variables corresponds to an infeasible basis (i.e. $A_B^{-1}b \not\geq 0$) then B corresponds to an infeasible basis in LP' (note that columns in A_B are linearly independent).
- LP' has no degenerate basic solutions

Degeneracy Revisited

If a basis variable is 0 in the basic feasible solution then we may not make progress during an iteration of simplex.

Idea:
Given feasible $\mathrm{LP} := \max\{c^t x, Ax = b; x \geq 0\}$. Change it into $\mathrm{LP}' := \max\{c^t x, Ax = b', x \geq 0\}$ such that

I. LP' is feasible

II. If a set B of basis variables corresponds to an infeasible basis (i.e. $A_B^{-1} b \not\geq 0$) then B corresponds to an infeasible basis in LP' (note that columns in A_B are linearly independent).

III. LP' has no degenerate basic solutions

Degeneracy Revisited

If a basis variable is 0 in the basic feasible solution then we may not make progress during an iteration of simplex.

Idea:
Given feasible $\mathrm{LP} := \max\{c^t x, Ax = b; x \geq 0\}$. Change it into $\mathrm{LP}' := \max\{c^t x, Ax = b', x \geq 0\}$ such that

I. LP' is feasible

II. If a set B of basis variables corresponds to an infeasible basis (i.e. $A_B^{-1} b \ngeq 0$) then B corresponds to an infeasible basis in LP' (note that columns in A_B are linearly independent).

III. LP' has no degenerate basic solutions

Degeneracy Revisited

If a basis variable is 0 in the basic feasible solution then we may not make progress during an iteration of simplex.

Idea:
Given feasible $\mathrm{LP} := \max\{c^t x, Ax = b; x \geq 0\}$. Change it into $\mathrm{LP}' := \max\{c^t x, Ax = b', x \geq 0\}$ such that

I. LP' is feasible

II. If a set B of basis variables corresponds to an infeasible basis (i.e. $A_B^{-1} b \not\geq 0$) then B corresponds to an infeasible basis in LP' (note that columns in A_B are linearly independent).

III. LP' has no degenerate basic solutions

Perturbation

Let B be index set of some basis with basic solution

$$x_B^* = A_B^{-1}b \geq 0, x_N^* = 0 \quad \text{(i.e. } B \text{ is feasible)}$$

Fix

$$b' := b + A_B \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix} \text{ for } \varepsilon > 0 \ .$$

This is the perturbation that we are using.

Perturbation

Let B be index set of some basis with basic solution

$$x_B^* = A_B^{-1}b \geq 0, x_N^* = 0 \quad \text{(i.e. } B \text{ is feasible)}$$

Fix

$$b' := b + A_B \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix} \text{ for } \varepsilon > 0 \enspace .$$

This is the perturbation that we are using.

Property I

The new LP is feasible because the set B of basis variables provides a feasible basis:

$$A_B^{-1}\left(b + A_B\begin{pmatrix}\varepsilon\\ \vdots\\ \varepsilon^m\end{pmatrix}\right) = x_B^* + \begin{pmatrix}\varepsilon\\ \vdots\\ \varepsilon^m\end{pmatrix} \geq 0 \ .$$

Property I

The new LP is feasible because the set B of basis variables provides a feasible basis:

$$A_B^{-1}\left(b + A_B\begin{pmatrix}\varepsilon\\ \vdots\\ \varepsilon^m\end{pmatrix}\right) = x_B^* + \begin{pmatrix}\varepsilon\\ \vdots\\ \varepsilon^m\end{pmatrix} \geq 0 \enspace .$$

Property II

Let $\tilde{B}$ be a non-feasible basis. This means $(A_{\tilde{B}}^{-1}b)_i < 0$ for some row i.

Property II

Let $\tilde{B}$ be a non-feasible basis. This means $(A_{\tilde{B}}^{-1}b)_i < 0$ for some row i.

Then for small enough $\epsilon > 0$

$$\left(A_{\tilde{B}}^{-1}\left(b + A_B\begin{pmatrix}\varepsilon\\ \vdots\\ \varepsilon^m\end{pmatrix}\right)\right)_i$$

Property II

Let $\tilde{B}$ be a non-feasible basis. This means $(A_{\tilde{B}}^{-1}b)_i < 0$ for some row i.

Then for small enough $\epsilon > 0$

$$\left(A_{\tilde{B}}^{-1}\left(b + A_B\begin{pmatrix}\varepsilon\\ \vdots\\ \varepsilon^m\end{pmatrix}\right)\right)_i = (A_{\tilde{B}}^{-1}b)_i + \left(A_{\tilde{B}}^{-1}A_B\begin{pmatrix}\varepsilon\\ \vdots\\ \varepsilon^m\end{pmatrix}\right)_i < 0$$

Property II

Let $\tilde{B}$ be a non-feasible basis. This means $(A_{\tilde{B}}^{-1}b)_i < 0$ for some row i.

Then for small enough $\epsilon > 0$

$$\left(A_{\tilde{B}}^{-1}\left(b + A_B\begin{pmatrix}\varepsilon\\ \vdots\\ \varepsilon^m\end{pmatrix}\right)\right)_i = (A_{\tilde{B}}^{-1}b)_i + \left(A_{\tilde{B}}^{-1}A_B\begin{pmatrix}\varepsilon\\ \vdots\\ \varepsilon^m\end{pmatrix}\right)_i < 0$$

Hence, $\tilde{B}$ is not feasible.

Property III

Let $\tilde{B}$ be a basis. It has an associated solution

$$x^*_{\tilde{B}} = A_{\tilde{B}}^{-1}b + A_{\tilde{B}}^{-1}A_B \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix}$$

in the perturbed instance.

We can view each component of the vector as a polynom with variable ε of degree at most m.

$A_{\tilde{B}}^{-1}A_B$ has rank m. Therefore no polynom is 0.

A polynom of degree at most m has at most m roots (Nullstellen).

Hence, $\epsilon > 0$ small enough gives that no component of the above vector is 0. Hence, no degeneracies.

Property III

Let $\tilde{B}$ be a basis. It has an associated solution

$$x^*_{\tilde{B}} = A^{-1}_{\tilde{B}} b + A^{-1}_{\tilde{B}} A_B \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix}$$

in the perturbed instance.

We can view each component of the vector as a polynom with variable ε of degree at most m.

$A^{-1}_{\tilde{B}} A_B$ has rank m. Therefore no polynom is 0.

A polynom of degree at most m has at most m roots (Nullstellen).

Hence, $\epsilon > 0$ small enough gives that no component of the above vector is 0. Hence, no degeneracies.

Property III

Let $\tilde{B}$ be a basis. It has an associated solution

$$x^*_{\tilde{B}} = A^{-1}_{\tilde{B}} b + A^{-1}_{\tilde{B}} A_B \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix}$$

in the perturbed instance.

We can view each component of the vector as a polynom with variable ε of degree at most m.

$A^{-1}_{\tilde{B}} A_B$ has rank m. Therefore no polynom is 0.

A polynom of degree at most m has at most m roots (Nullstellen).

Hence, $\epsilon > 0$ small enough gives that no component of the above vector is 0. Hence, no degeneracies.

Property III

Let $\tilde{B}$ be a basis. It has an associated solution

$$x^*_{\tilde{B}} = A_{\tilde{B}}^{-1}b + A_{\tilde{B}}^{-1}A_B \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix}$$

in the perturbed instance.

We can view each component of the vector as a polynom with variable ε of degree at most m.

$A_{\tilde{B}}^{-1}A_B$ has rank m. Therefore no polynom is 0.

A polynom of degree at most m has at most m roots (Nullstellen).

Hence, $\epsilon > 0$ small enough gives that no component of the above vector is 0. Hence, no degeneracies.

Property III

Let $\tilde{B}$ be a basis. It has an associated solution

$$x^*_{\tilde{B}} = A_{\tilde{B}}^{-1} b + A_{\tilde{B}}^{-1} A_B \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix}$$

in the perturbed instance.

We can view each component of the vector as a polynom with variable ε of degree at most m.

$A_{\tilde{B}}^{-1} A_B$ has rank m. Therefore no polynom is 0.

A polynom of degree at most m has at most m roots (Nullstellen).

Hence, $\epsilon > 0$ small enough gives that no component of the above vector is 0. Hence, no degeneracies.

Property III

Let $\tilde{B}$ be a basis. It has an associated solution

$$x^*_{\tilde{B}} = A_{\tilde{B}}^{-1} b + A_{\tilde{B}}^{-1} A_B \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix}$$

in the perturbed instance.

We can view each component of the vector as a polynom with variable ε of degree at most m.

$A_{\tilde{B}}^{-1} A_B$ has rank m. Therefore no polynom is 0.

A polynom of degree at most m has at most m roots (Nullstellen).

Hence, $\epsilon > 0$ small enough gives that no component of the above vector is 0. Hence, no degeneracies.

Since, there are no degeneracies Simplex will terminate when run on LP'.

Since, there are no degeneracies Simplex will terminate when run on LP'.

- If it terminates because the reduced cost vector fulfills

$$\tilde{c} = (c^t - c_B^t A_B^{-1} A) \leq 0$$

then we have found an optimal basis.

Since, there are no degeneracies Simplex will terminate when run on LP'.

- If it terminates because the reduced cost vector fulfills

$$\tilde{c} = (c^t - c_B^t A_B^{-1} A) \leq 0$$

then we have found an optimal basis. Note that this basis is also optimal for LP, as the above constraint does not depend on b.

Since, there are no degeneracies Simplex will terminate when run on LP'.

- If it terminates because the reduced cost vector fulfills

$$\tilde{c} = (c^t - c_B^t A_B^{-1} A) \le 0$$

  then we have found an optimal basis. Note that this basis is also optimal for LP, as the above constraint does not depend on b.

- If it terminates because it finds a variable x_j with $\tilde{c}_j > 0$ for which the j-th basis direction d, fulfills $d \ge 0$ we know that LP' is unbounded. The basis direction does not depend on b. Hence, we also know that LP is unbounded.

Lexicographic Pivoting

Doing calculations with perturbed instances may be costly. Also the right choice of ε is difficult.

Idea:

Simulate behaviour of LP' without explicitly doing a perturbation.

Lexicographic Pivoting

Doing calculations with perturbed instances may be costly. Also the right choice of ε is difficult.

Idea:

Simulate behaviour of LP' without explicitly doing a perturbation.

Lexicographic Pivoting

Doing calculations with perturbed instances may be costly. Also the right choice of ε is difficult.

Idea:
Simulate behaviour of LP' without explicitly doing a perturbation.

Lexicographic Pivoting

We choose the entering variable arbitrarily as before ($\tilde{c}_e > 0$, of course).

If we do not have a choice for the leaving variable then LP' and LP do the same (i.e., choose the same variable).

Otherwise we have to be careful.

Lexicographic Pivoting

We choose the entering variable arbitrarily as before ($\tilde{c}_e > 0$, of course).

If we do not have a choice for the leaving variable then LP' and LP do the same (i.e., choose the same variable).

Otherwise we have to be careful.

Lexicographic Pivoting

We choose the entering variable arbitrarily as before ($\tilde{c}_e > 0$, of course).

If we do not have a choice for the leaving variable then LP' and LP do the same (i.e., choose the same variable).

Otherwise we have to be careful.

Lexicographic Pivoting

We choose the entering variable arbitrarily as before ($\tilde{c}_e > 0$, of course).

If we do not have a choice for the leaving variable then LP' and LP do the same (i.e., choose the same variable).

Otherwise we have to be careful.

Lexicographic Pivoting

In the following we assume that $b \geq 0$. This can be obtained by replacing the initial system $(A_B \mid b)$ by $(A_B^{-1}A \mid A_B^{-1}b)$ where B is the index set of a feasible basis (found e.g. by the first phase of the Two-phase algorithm).

Then the perturbed instance is

$$b' = b + \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix}$$

Lexicographic Pivoting

In the following we assume that $b \geq 0$. This can be obtained by replacing the initial system $(A_B \mid b)$ by $(A_B^{-1}A \mid A_B^{-1}b)$ where B is the index set of a feasible basis (found e.g. by the first phase of the Two-phase algorithm).

Then the perturbed instance is

$$b' = b + \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix}$$

Matrix View

Let our linear program be

$$\begin{array}{ccccc} c_B^t x_B & + & c_N^t x_N & = & Z \\ A_B x_B & + & A_N x_N & = & b \\ x_B & , & x_N & \geq & 0 \end{array}$$

The simplex tableaux for basis B is

$$\begin{array}{ccccc} & & (c_N^t - c_B^t A_B^{-1} A_N) x_N & = & Z - c_B^t A_B^{-1} b \\ I x_B & + & A_B^{-1} A_N x_N & = & A_B^{-1} b \\ x_B & , & x_N & \geq & 0 \end{array}$$

The BFS is given by $x_N = 0, x_B = A_B^{-1} b$.

If $(c_N^t - c_B^t A_B^{-1} A_N) \leq 0$ we know that we have an optimum solution.

Lexicographic Pivoting

LP chooses an arbitrary leaving variable that has $\hat{A}_{\ell e} > 0$ and minimizes

$$\theta_\ell = \frac{\hat{b}_\ell}{\hat{A}_{\ell e}} = \frac{(A_B^{-1}b)_\ell}{(A_B^{-1}A_{*e})_\ell} .$$

ℓ is the index of a leaving variable within B. This means if e.g. $B = \{1, 3, 7, 14\}$ and leaving variable is 3 then $\ell = 2$.

Lexicographic Pivoting

LP chooses an arbitrary leaving variable that has $\hat{A}_{\ell e} > 0$ and minimizes

$$\theta_\ell = \frac{\hat{b}_\ell}{\hat{A}_{\ell e}} = \frac{(A_B^{-1}b)_\ell}{(A_B^{-1}A_{*e})_\ell} .$$

ℓ is the index of a leaving variable within B. This means if e.g. $B = \{1, 3, 7, 14\}$ and leaving variable is 3 then $\ell = 2$.

Lexicographic Pivoting

LP chooses an arbitrary leaving variable that has $\hat{A}_{\ell e} > 0$ and minimizes

$$\theta_\ell = \frac{\hat{b}_\ell}{\hat{A}_{\ell e}} = \frac{(A_B^{-1}b)_\ell}{(A_B^{-1}A_{*e})_\ell} \enspace .$$

ℓ is the index of a leaving variable within B. This means if e.g. $B = \{1, 3, 7, 14\}$ and leaving variable is 3 then $\ell = 2$.

Lexicographic Pivoting

LP chooses an arbitrary leaving variable that has $\hat{A}_{\ell e} > 0$ and minimizes

$$\theta_\ell = \frac{\hat{b}_\ell}{\hat{A}_{\ell e}} = \frac{(A_B^{-1}b)_\ell}{(A_B^{-1}A_{*e})_\ell} \enspace .$$

ℓ is the index of a leaving variable within B. This means if e.g. $B = \{1,3,7,14\}$ and leaving variable is 3 then $\ell = 2$.

Lexicographic Pivoting

Definition 29
$u \leq_{\text{lex}} v$ if and only if the first component in which u and v differ fulfills $u_i \leq v_i$.

Lexicographic Pivoting

LP′ chooses an index that minimizes

θ_ℓ

Lexicographic Pivoting

LP′ chooses an index that minimizes

$$\theta_\ell = \frac{\left(A_B^{-1}\left(b + \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix}\right)\right)_\ell}{(A_B^{-1}A_{*e})_\ell}$$

Lexicographic Pivoting

LP′ chooses an index that minimizes

$$\theta_\ell = \frac{\left(A_B^{-1}\left(b + \begin{pmatrix}\varepsilon\\ \vdots\\ \varepsilon^m\end{pmatrix}\right)\right)_\ell}{(A_B^{-1}A_{*e})_\ell} = \frac{\left(A_B^{-1}(b \mid I)\begin{pmatrix}1\\ \varepsilon\\ \vdots\\ \varepsilon^m\end{pmatrix}\right)_\ell}{(A_B^{-1}A_{*e})_\ell}$$

Lexicographic Pivoting

LP' chooses an index that minimizes

$$\theta_\ell = \frac{\left(A_B^{-1}\left(b + \begin{pmatrix}\varepsilon\\ \vdots\\ \varepsilon^m\end{pmatrix}\right)\right)_\ell}{(A_B^{-1}A_{*e})_\ell} = \frac{\left(A_B^{-1}(b \mid I)\begin{pmatrix}1\\ \varepsilon\\ \vdots\\ \varepsilon^m\end{pmatrix}\right)_\ell}{(A_B^{-1}A_{*e})_\ell}$$

$$= \frac{\ell\text{-th row of } A_B^{-1}(b \mid I)}{(A_B^{-1}A_{*e})_\ell}\begin{pmatrix}1\\ \varepsilon\\ \vdots\\ \varepsilon^m\end{pmatrix}$$

Lexicographic Pivoting

This means you can choose the variable/row ℓ for which the vector

$$\frac{\ell\text{-th row of } A_B^{-1}(b \mid I)}{(A_B^{-1}A_{*e})_\ell}$$

is lexicographically minimal.

Of course only including rows with $(A_B^{-1}A_{*e})_\ell > 0$.

This technique guarantees that your pivoting is the same as in the perturbed case. This guarantees that cycling does not occur.

Lexicographic Pivoting

This means you can choose the variable/row ℓ for which the vector

$$\frac{\ell\text{-th row of } A_B^{-1}(b \mid I)}{(A_B^{-1}A_{*e})_\ell}$$

is lexicographically minimal.

Of course only including rows with $(A_B^{-1}A_{*e})_\ell > 0$.

This technique guarantees that your pivoting is the same as in the perturbed case. This guarantees that cycling does not occur.

Lexicographic Pivoting

This means you can choose the variable/row ℓ for which the vector

$$\frac{\ell\text{-th row of } A_B^{-1}(b \mid I)}{(A_B^{-1}A_{*e})_\ell}$$

is lexicographically minimal.

Of course only including rows with $(A_B^{-1}A_{*e})_\ell > 0$.

This technique guarantees that your pivoting is the same as in the perturbed case. This guarantees that cycling does not occur.

Number of Simplex Iterations

Number of Simplex Iterations

Each iteration of Simplex can be implemented in polynomial time.

Number of Simplex Iterations

Each iteration of Simplex can be implemented in polynomial time.

If we use lexicographic pivoting we know that Simplex requires at most $\binom{n}{m}$ iterations, because it will not visit a basis twice.

Number of Simplex Iterations

Each iteration of Simplex can be implemented in polynomial time.

If we use lexicographic pivoting we know that Simplex requires at most $\binom{n}{m}$ iterations, because it will not visit a basis twice.

The input size is $L \cdot n \cdot m$, where n is the number of variables, m is the number of constraints, and L is the length of the binary representation of the largest coefficient in the matrix A.

Number of Simplex Iterations

Each iteration of Simplex can be implemented in polynomial time.

If we use lexicographic pivoting we know that Simplex requires at most $\binom{n}{m}$ iterations, because it will not visit a basis twice.

The input size is $L \cdot n \cdot m$, where n is the number of variables, m is the number of constraints, and L is the length of the binary representation of the largest coefficient in the matrix A.

If we really require $\binom{n}{m}$ iterations then Simplex is not a polynomial time algorithm.

Number of Simplex Iterations

Each iteration of Simplex can be implemented in polynomial time.

If we use lexicographic pivoting we know that Simplex requires at most $\binom{n}{m}$ iterations, because it will not visit a basis twice.

The input size is $L \cdot n \cdot m$, where n is the number of variables, m is the number of constraints, and L is the length of the binary representation of the largest coefficient in the matrix A.

If we really require $\binom{n}{m}$ iterations then Simplex is not a polynomial time algorithm.

Can we obtain a better analysis?

Number of Simplex Iterations

Observation

Simplex visits every feasible basis at most once.

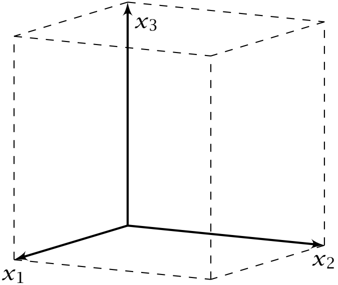
Number of Simplex Iterations

Observation
Simplex visits every feasible basis at most once.

However, also the number of feasible bases can be very large.

Example

$$\begin{aligned} \max\ & c^t x \\ \text{s.t.}\ & 0 \le x_1 \le 1 \\ & 0 \le x_2 \le 1 \\ & \vdots \\ & 0 \le x_n \le 1 \end{aligned}$$

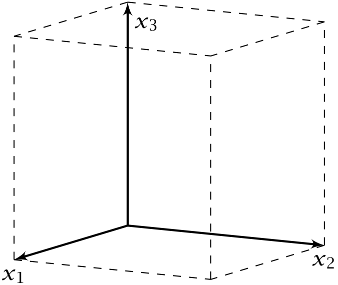


$2n$ constraint on n variables define an n-dimensional hypercube as feasible region.

The feasible region has 2^n vertices.

Example

$$\begin{aligned} \max\ & c^t x \\ \text{s.t.}\ & 0 \le x_1 \le 1 \\ & 0 \le x_2 \le 1 \\ & \vdots \\ & 0 \le x_n \le 1 \end{aligned}$$



However, Simplex may still run quickly as it usually does not visit all feasible bases.

In the following we give an example of a feasible region for which there is a bad Pivoting Rule.

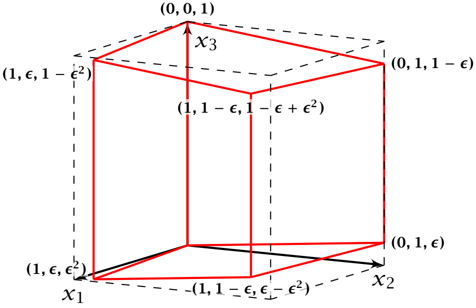
Pivoting Rule

A Pivoting Rule defines how to choose the entering and leaving variable for an iteration of Simplex.

In the non-degenerate case after choosing the entering variable the leaving variable is unique.

Klee Minty Cube

$$\begin{aligned}
\max\ & x_n \\
\text{s.t.}\quad & 0 \le x_1 \le 1 \\
& \epsilon x_1 \le x_2 \le 1 - \epsilon x_1 \\
& \epsilon x_2 \le x_3 \le 1 - \epsilon x_2 \\
& \vdots \\
& \epsilon x_{n-1} \le x_n \le 1 - \epsilon x_{n-1} \\
& x_i \ge 0
\end{aligned}$$



Observations

- We have $2n$ constraints, and $3n$ variables (after adding slack variables to every constraint).
- Every basis is defined by $2n$ variables, and n non-basic variables.
- There exist degenerate vertices.
- The degeneracies come from the non-negativity constraints, which are superfluous.
- In the following all variables x_i stay in the basis at all times.
- Then, we can uniquely specify a basis by choosing for each variable whether it should be equal to its lower bound, or equal to its upper bound (the slack variable corresponding to the non-tight constraint is part of the basis).
- We can also simply identify each basis/vertex with the corresponding hypercube vertex obtained by letting $\epsilon \to 0$.

Observations

- We have $2n$ constraints, and $3n$ variables (after adding slack variables to every constraint).
- Every basis is defined by $2n$ variables, and n non-basic variables.
- There exist degenerate vertices.
- The degeneracies come from the non-negativity constraints, which are superfluous.
- In the following all variables x_i stay in the basis at all times.
- Then, we can uniquely specify a basis by choosing for each variable whether it should be equal to its lower bound, or equal to its upper bound (the slack variable corresponding to the non-tight constraint is part of the basis).
- We can also simply identify each basis/vertex with the corresponding hypercube vertex obtained by letting $\epsilon \to 0$.

Observations

- We have $2n$ constraints, and $3n$ variables (after adding slack variables to every constraint).
- Every basis is defined by $2n$ variables, and n non-basic variables.
- There exist degenerate vertices.
- The degeneracies come from the non-negativity constraints, which are superfluous.
- In the following all variables x_i stay in the basis at all times.
- Then, we can uniquely specify a basis by choosing for each variable whether it should be equal to its lower bound, or equal to its upper bound (the slack variable corresponding to the non-tight constraint is part of the basis).
- We can also simply identify each basis/vertex with the corresponding hypercube vertex obtained by letting $\epsilon \to 0$.

Observations

- We have $2n$ constraints, and $3n$ variables (after adding slack variables to every constraint).
- Every basis is defined by $2n$ variables, and n non-basic variables.
- There exist degenerate vertices.
- The degeneracies come from the non-negativity constraints, which are superfluous.
- In the following all variables x_i stay in the basis at all times.
- Then, we can uniquely specify a basis by choosing for each variable whether it should be equal to its lower bound, or equal to its upper bound (the slack variable corresponding to the non-tight constraint is part of the basis).
- We can also simply identify each basis/vertex with the corresponding hypercube vertex obtained by letting $\epsilon \to 0$.

Observations

- We have $2n$ constraints, and $3n$ variables (after adding slack variables to every constraint).
- Every basis is defined by $2n$ variables, and n non-basic variables.
- There exist degenerate vertices.
- The degeneracies come from the non-negativity constraints, which are superfluous.
- In the following all variables x_i stay in the basis at all times.
- Then, we can uniquely specify a basis by choosing for each variable whether it should be equal to its lower bound, or equal to its upper bound (the slack variable corresponding to the non-tight constraint is part of the basis).
- We can also simply identify each basis/vertex with the corresponding hypercube vertex obtained by letting $\epsilon \to 0$.

Observations

- We have $2n$ constraints, and $3n$ variables (after adding slack variables to every constraint).
- Every basis is defined by $2n$ variables, and n non-basic variables.
- There exist degenerate vertices.
- The degeneracies come from the non-negativity constraints, which are superfluous.
- In the following all variables x_i stay in the basis at all times.
- Then, we can uniquely specify a basis by choosing for each variable whether it should be equal to its lower bound, or equal to its upper bound (the slack variable corresponding to the non-tight constraint is part of the basis).
- We can also simply identify each basis/vertex with the corresponding hypercube vertex obtained by letting $\epsilon \to 0$.

Observations

- We have $2n$ constraints, and $3n$ variables (after adding slack variables to every constraint).
- Every basis is defined by $2n$ variables, and n non-basic variables.
- There exist degenerate vertices.
- The degeneracies come from the non-negativity constraints, which are superfluous.
- In the following all variables x_i stay in the basis at all times.
- Then, we can uniquely specify a basis by choosing for each variable whether it should be equal to its lower bound, or equal to its upper bound (the slack variable corresponding to the non-tight constraint is part of the basis).
- We can also simply identify each basis/vertex with the corresponding hypercube vertex obtained by letting $\epsilon \to 0$.

Analysis

- In the following we specify a sequence of bases (identified by the corresponding hypercube node) along which the objective function strictly increases.
- The basis $(0,\ldots,0,1)$ is the unique optimal basis.
- Our sequence S_n starts at $(0,\ldots,0)$ ends with $(0,\ldots,0,1)$ and visits every node of the hypercube.
- An unfortunate Pivoting Rule may choose this sequence, and, hence, require an exponential number of iterations.

Analysis

- In the following we specify a sequence of bases (identified by the corresponding hypercube node) along which the objective function strictly increases.
- The basis $(0, \ldots, 0, 1)$ is the unique optimal basis.
- Our sequence S_n starts at $(0, \ldots, 0)$ ends with $(0, \ldots, 0, 1)$ and visits every node of the hypercube.
- An unfortunate Pivoting Rule may choose this sequence, and, hence, require an exponential number of iterations.

Analysis

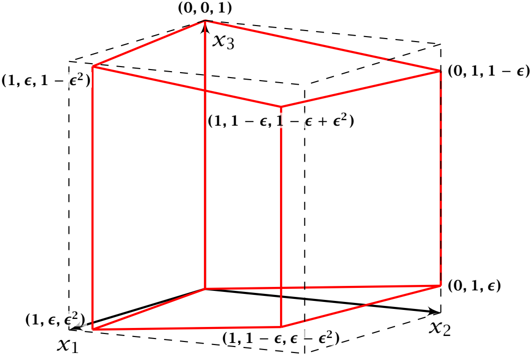
- In the following we specify a sequence of bases (identified by the corresponding hypercube node) along which the objective function strictly increases.
- The basis $(0, \ldots, 0, 1)$ is the unique optimal basis.
- Our sequence S_n starts at $(0, \ldots, 0)$ ends with $(0, \ldots, 0, 1)$ and visits every node of the hypercube.
- An unfortunate Pivoting Rule may choose this sequence, and, hence, require an exponential number of iterations.

Analysis

- In the following we specify a sequence of bases (identified by the corresponding hypercube node) along which the objective function strictly increases.
- The basis $(0, \dots, 0, 1)$ is the unique optimal basis.
- Our sequence S_n starts at $(0, \dots, 0)$ ends with $(0, \dots, 0, 1)$ and visits every node of the hypercube.
- An unfortunate Pivoting Rule may choose this sequence, and, hence, require an exponential number of iterations.

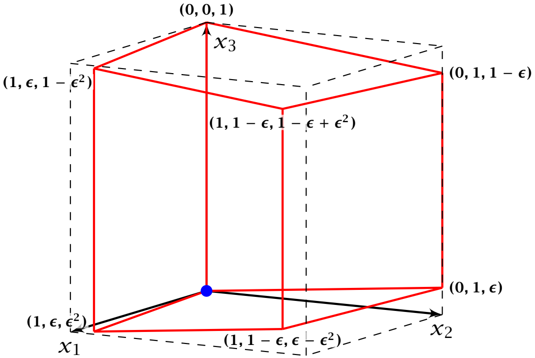
Klee Minty Cube

$$\begin{aligned} \max\ & x_n \\ \text{s.t.}\quad & 0 \le x_1 \le 1 \\ & \epsilon x_1 \le x_2 \le 1 - \epsilon x_1 \\ & \epsilon x_2 \le x_3 \le 1 - \epsilon x_2 \end{aligned}$$



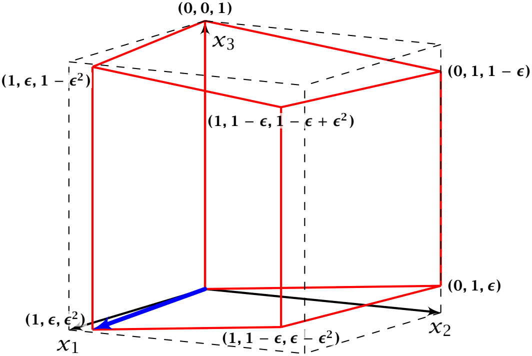
Klee Minty Cube

$$\begin{aligned} \max\ & x_n \\ \text{s.t.}\ & 0 \le x_1 \le 1 \\ & \epsilon x_1 \le x_2 \le 1 - \epsilon x_1 \\ & \epsilon x_2 \le x_3 \le 1 - \epsilon x_2 \end{aligned}$$



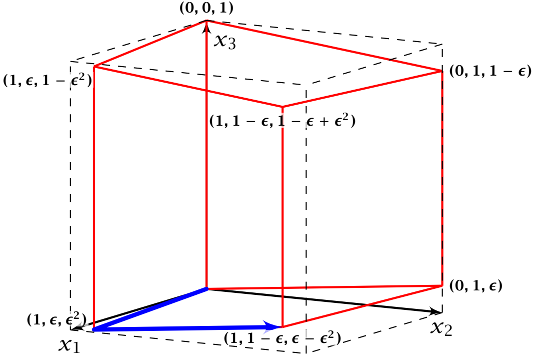
Klee Minty Cube

$$\begin{aligned} \max\ & x_n \\ \text{s.t.}\quad & 0 \le x_1 \le 1 \\ & \epsilon x_1 \le x_2 \le 1 - \epsilon x_1 \\ & \epsilon x_2 \le x_3 \le 1 - \epsilon x_2 \end{aligned}$$



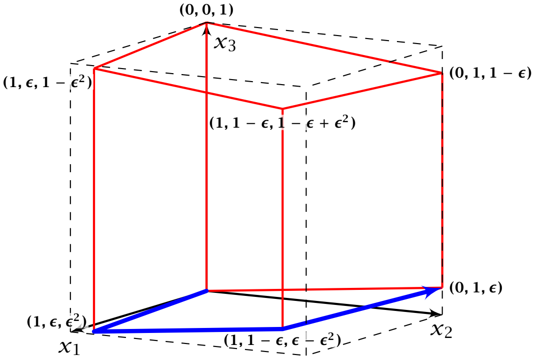
Klee Minty Cube

$$\begin{aligned} \max\ & x_n \\ \text{s.t.}\quad & 0 \le x_1 \le 1 \\ & \epsilon x_1 \le x_2 \le 1 - \epsilon x_1 \\ & \epsilon x_2 \le x_3 \le 1 - \epsilon x_2 \end{aligned}$$



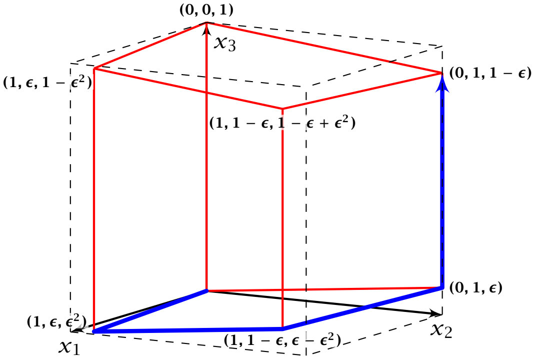
Klee Minty Cube

$$\begin{aligned} \max\ & x_n \\ \text{s.t.}\ & 0 \le x_1 \le 1 \\ & \epsilon x_1 \le x_2 \le 1 - \epsilon x_1 \\ & \epsilon x_2 \le x_3 \le 1 - \epsilon x_2 \end{aligned}$$



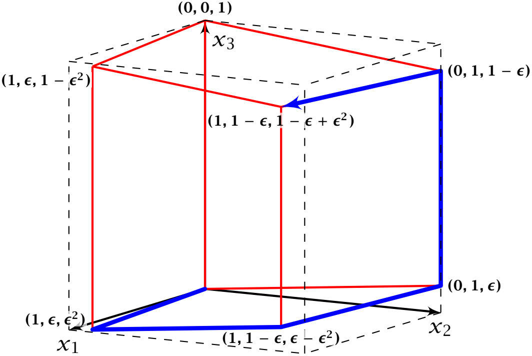
Klee Minty Cube

$$\begin{aligned} \max\ & x_n \\ \text{s.t.}\quad & 0 \le x_1 \le 1 \\ & \epsilon x_1 \le x_2 \le 1 - \epsilon x_1 \\ & \epsilon x_2 \le x_3 \le 1 - \epsilon x_2 \end{aligned}$$



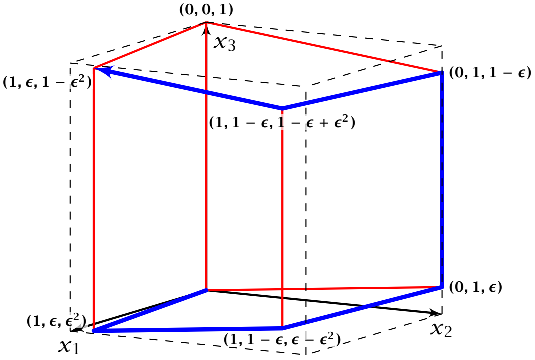
Klee Minty Cube

$$\max x_n$$
$$\text{s.t.} \quad 0 \le x_1 \le 1$$
$$\epsilon x_1 \le x_2 \le 1 - \epsilon x_1$$
$$\epsilon x_2 \le x_3 \le 1 - \epsilon x_2$$



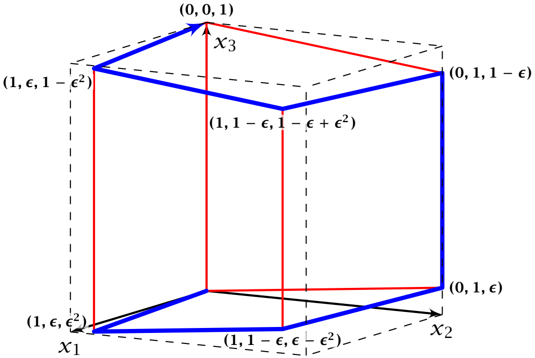
Klee Minty Cube

$$\begin{aligned} \max\ & x_n \\ \text{s.t.}\quad & 0 \le x_1 \le 1 \\ & \epsilon x_1 \le x_2 \le 1 - \epsilon x_1 \\ & \epsilon x_2 \le x_3 \le 1 - \epsilon x_2 \end{aligned}$$



Klee Minty Cube

$$\begin{aligned} \max\ & x_n \\ \text{s.t.}\ & 0 \le x_1 \le 1 \\ & \epsilon x_1 \le x_2 \le 1 - \epsilon x_1 \\ & \epsilon x_2 \le x_3 \le 1 - \epsilon x_2 \end{aligned}$$



Analysis

The sequence S_n that visits every node of the hypercube is defined recursively

$$\begin{array}{c}
(0,\ldots,0,0,0) \\
\big\downarrow S_{n-1} \\
(0,\ldots,0,1,0) \\
\big\downarrow \\
(0,\ldots,0,1,1) \\
\big\downarrow S_{n-1}^{\mathrm{rev}} \\
(0,\ldots,0,0,1)
\end{array}
\quad \Big\downarrow S_n$$

The non-recursive case is $S_1 = 0 \to 1$

Analysis

Lemma 30

The objective value x_n is increasing along path S_n.

Proof by induction:

$n = 1$: obvious, since $S_1 = 0 \to 1$, and $1 > 0$.

$n - 1 \to n$

- For the first part the value of $x_n = \epsilon x_{n-1}$.
- By induction hypothesis x_{n-1} is increasing along S_{n-1}, hence, also x_n.
- Going from $(0, \dots, 0, 1, 0)$ to $(0, \dots, 0, 1, 1)$ increases x_n for small enough ϵ.
- For the remaining path S_{n-1}^{rev} we have $x_n = 1 - \epsilon x_{n-1}$.
- By induction hypothesis x_{n-1} is increasing along S_{n-1}, hence $-\epsilon x_{n-1}$ is increasing along S_{n-1}^{rev}.

Analysis

Lemma 30
The objective value x_n is increasing along path S_n.

Proof by induction:

$n = 1$: obvious, since $S_1 = 0 \to 1$, and $1 > 0$.

$n - 1 \to n$

- For the first part the value of $x_n = \epsilon x_{n-1}$.
- By induction hypothesis x_{n-1} is increasing along S_{n-1}, hence, also x_n.
- Going from $(0, \ldots, 0, 1, 0)$ to $(0, \ldots, 0, 1, 1)$ increases x_n for small enough ϵ.
- For the remaining path S_{n-1}^{rev} we have $x_n = 1 - \epsilon x_{n-1}$.
- By induction hypothesis x_{n-1} is increasing along S_{n-1}, hence $-\epsilon x_{n-1}$ is increasing along S_{n-1}^{rev}.

Analysis

Lemma 30

The objective value x_n is increasing along path S_n.

Proof by induction:

$\boldsymbol{n} = \mathbf{1}$: obvious, since $S_1 = 0 \to 1$, and $1 > 0$.

$\boldsymbol{n - 1 \to n}$

- For the first part the value of $x_n = \epsilon x_{n-1}$.
- By induction hypothesis x_{n-1} is increasing along S_{n-1}, hence, also x_n.
- Going from $(0, \ldots, 0, 1, 0)$ to $(0, \ldots, 0, 1, 1)$ increases x_n for small enough ϵ.
- For the remaining path S_{n-1}^{rev} we have $x_n = 1 - \epsilon x_{n-1}$.
- By induction hypothesis x_{n-1} is increasing along S_{n-1}, hence $-\epsilon x_{n-1}$ is increasing along S_{n-1}^{rev}.

Analysis

Lemma 30

The objective value x_n is increasing along path S_n.

Proof by induction:

$\boldsymbol{n = 1}$: obvious, since $S_1 = 0 \rightarrow 1$, and $1 > 0$.

$\boldsymbol{n - 1 \rightarrow n}$

- For the first part the value of $x_n = \epsilon x_{n-1}$.
- By induction hypothesis x_{n-1} is increasing along S_{n-1}, hence, also x_n.
- Going from $(0, \ldots, 0, 1, 0)$ to $(0, \ldots, 0, 1, 1)$ increases x_n for small enough ϵ.
- For the remaining path S_{n-1}^{rev} we have $x_n = 1 - \epsilon x_{n-1}$.
- By induction hypothesis x_{n-1} is increasing along S_{n-1}, hence $-\epsilon x_{n-1}$ is increasing along S_{n-1}^{rev}.

Analysis

Lemma 30

The objective value x_n is increasing along path S_n.

Proof by induction:

$\boldsymbol{n = 1}$: obvious, since $S_1 = 0 \to 1$, and $1 > 0$.

$\boldsymbol{n - 1 \to n}$

- For the first part the value of $x_n = \epsilon x_{n-1}$.
- By induction hypothesis x_{n-1} is increasing along S_{n-1}, hence, also x_n.
- Going from $(0, \dots, 0, 1, 0)$ to $(0, \dots, 0, 1, 1)$ increases x_n for small enough ϵ.
- For the remaining path S_{n-1}^{rev} we have $x_n = 1 - \epsilon x_{n-1}$.
- By induction hypothesis x_{n-1} is increasing along S_{n-1}, hence $-\epsilon x_{n-1}$ is increasing along S_{n-1}^{rev}.

Analysis

Lemma 30

The objective value x_n is increasing along path S_n.

Proof by induction:

$\boldsymbol{n = 1}$: obvious, since $S_1 = 0 \rightarrow 1$, and $1 > 0$.

$\boldsymbol{n - 1 \rightarrow n}$

- For the first part the value of $x_n = \epsilon x_{n-1}$.
- By induction hypothesis x_{n-1} is increasing along S_{n-1}, hence, also x_n.
- Going from $(0, \ldots, 0, 1, 0)$ to $(0, \ldots, 0, 1, 1)$ increases x_n for small enough ϵ.
- For the remaining path S_{n-1}^{rev} we have $x_n = 1 - \epsilon x_{n-1}$.
- By induction hypothesis x_{n-1} is increasing along S_{n-1}, hence $-\epsilon x_{n-1}$ is increasing along S_{n-1}^{rev}.

Analysis

Lemma 30
The objective value x_n is increasing along path S_n.

Proof by induction:

$\boldsymbol{n = 1}$: obvious, since $S_1 = 0 \rightarrow 1$, and $1 > 0$.

$\boldsymbol{n - 1 \rightarrow n}$

- For the first part the value of $x_n = \epsilon x_{n-1}$.
- By induction hypothesis x_{n-1} is increasing along S_{n-1}, hence, also x_n.
- Going from $(0, \ldots, 0, 1, 0)$ to $(0, \ldots, 0, 1, 1)$ increases x_n for small enough ϵ.
- For the remaining path S_{n-1}^{rev} we have $x_n = 1 - \epsilon x_{n-1}$.
- By induction hypothesis x_{n-1} is increasing along S_{n-1}, hence $-\epsilon x_{n-1}$ is increasing along S_{n-1}^{rev}.

Analysis

Lemma 30

The objective value x_n is increasing along path S_n.

Proof by induction:

$\boldsymbol{n = 1}$: obvious, since $S_1 = 0 \to 1$, and $1 > 0$.

$\boldsymbol{n - 1 \to n}$

- For the first part the value of $x_n = \epsilon x_{n-1}$.
- By induction hypothesis x_{n-1} is increasing along S_{n-1}, hence, also x_n.
- Going from $(0, \ldots, 0, 1, 0)$ to $(0, \ldots, 0, 1, 1)$ increases x_n for small enough ϵ.
- For the remaining path S_{n-1}^{rev} we have $x_n = 1 - \epsilon x_{n-1}$.
- By induction hypothesis x_{n-1} is increasing along S_{n-1}, hence $-\epsilon x_{n-1}$ is increasing along S_{n-1}^{rev}.

Remarks about Simplex

Observation

The simplex algorithm takes at most $\binom{n}{m}$ iterations. Each iteration can be implemented in time $\mathcal{O}(mn)$.

In practise it usually takes a linear number of iterations.

Remarks about Simplex

Theorem

For almost all known deterministic pivoting rules (rules for choosing entering and leaving variables) there exist lower bounds that require the algorithm to have exponential running time ($\Omega(2^{\Omega(n)})$) (e.g. Klee Minty 1972).

Remarks about Simplex

Theorem
For some standard randomized pivoting rules there exist subexponential lower bounds ($\Omega(2^{\Omega(n^\alpha)})$ for $\alpha > 0$) (Friedmann, Hansen, Zwick 2011).

Remarks about Simplex

Conjecture (Hirsch 1957)
The edge-vertex graph of an m-facet polytope in d-dimensional Euclidean space has diameter no more than $m - d$.

The conjecture has been proven wrong in 2010.

But the question whether the diameter is perhaps of the form $\mathcal{O}(\text{poly}(m, d))$ is open.

8 Seidels LP-algorithm

- Suppose we want to solve $\min\{c^t x \mid Ax \geq b; x \geq 0\}$, where $x \in \mathbb{R}^d$ and we have m constraints.
- In the worst-case Simplex runs in time roughly $\mathcal{O}(m(m+d)\binom{m+d}{m}) \approx (m+d)^m$. (slightly better bounds on the running time exist, but will not be discussed here).
- If d is much smaller than m one can do a lot better.
- In the following we develop an algorithm with running time $\mathcal{O}(d! \cdot m)$, i.e., linear in m.

8 Seidels LP-algorithm

- Suppose we want to solve $\min\{c^t x \mid Ax \geq b; x \geq 0\}$, where $x \in \mathbb{R}^d$ and we have m constraints.
- In the worst-case Simplex runs in time roughly $\mathcal{O}(m(m+d)\binom{m+d}{m}) \approx (m+d)^m$. (slightly better bounds on the running time exist, but will not be discussed here).
- If d is much smaller than m one can do a lot better.
- In the following we develop an algorithm with running time $\mathcal{O}(d! \cdot m)$, i.e., linear in m.

8 Seidels LP-algorithm

- Suppose we want to solve $\min\{c^t x \mid Ax \geq b; x \geq 0\}$, where $x \in \mathbb{R}^d$ and we have m constraints.
- In the worst-case Simplex runs in time roughly $\mathcal{O}(m(m+d)\binom{m+d}{m}) \approx (m+d)^m$. (slightly better bounds on the running time exist, but will not be discussed here).
- If d is much smaller than m one can do a lot better.
- In the following we develop an algorithm with running time $\mathcal{O}(d! \cdot m)$, i.e., linear in m.

8 Seidels LP-algorithm

- Suppose we want to solve $\min\{c^t x \mid Ax \geq b; x \geq 0\}$, where $x \in \mathbb{R}^d$ and we have m constraints.
- In the worst-case Simplex runs in time roughly $\mathcal{O}(m(m+d)\binom{m+d}{m}) \approx (m+d)^m$. (slightly better bounds on the running time exist, but will not be discussed here).
- If d is much smaller than m one can do a lot better.
- In the following we develop an algorithm with running time $\mathcal{O}(d! \cdot m)$, i.e., linear in m.

8 Seidels LP-algorithm

Setting:

- We assume an LP of the form

$$\begin{array}{lrcl} \min & c^t x & & \\ \text{s.t.} & Ax & \geq & b \\ & x & \geq & 0 \end{array}$$

- We assume that the LP is bounded.

Ensuring Conditions

Given a standard minimization LP

$$
\begin{array}{lrcl}
\min & c^t x & & \\
\text{s.t.} & Ax & \geq & b \\
 & x & \geq & 0
\end{array}
$$

how can we obtain an LP of the required form?

- **Compute a lower bound on $c^t x$ for any basic feasible solution.**

Computing a Lower Bound

Let s denote the smallest common multiple of all denominators of entries in A, b.

Multiply entries in A, b by s to obtain integral entries. This does not change the feasible region.

Add slack variables to A; denote the resulting matrix with $\bar{A}$.

If B is an optimal basis then x_B with $\bar{A}_B x_B = b$, gives an optimal assignment to the basis variables (non-basic variables are 0).

Computing a Lower Bound

Let s denote the smallest common multiple of all denominators of entries in A, b.

Multiply entries in A, b by s to obtain integral entries. This does not change the feasible region.

Add slack variables to A; denote the resulting matrix with $\bar{A}$.

If B is an optimal basis then x_B with $\bar{A}_B x_B = b$, gives an optimal assignment to the basis variables (non-basic variables are 0).

Computing a Lower Bound

Let s denote the smallest common multiple of all denominators of entries in A, b.

Multiply entries in A, b by s to obtain integral entries. This does not change the feasible region.

Add slack variables to A; denote the resulting matrix with $\bar{A}$.

If B is an optimal basis then x_B with $\bar{A}_B x_B = b$, gives an optimal assignment to the basis variables (non-basic variables are 0).

Computing a Lower Bound

Let s denote the smallest common multiple of all denominators of entries in A, b.

Multiply entries in A, b by s to obtain integral entries. This does not change the feasible region.

Add slack variables to A; denote the resulting matrix with $\bar{A}$.

If B is an optimal basis then x_B with $\bar{A}_B x_B = b$, gives an optimal assignment to the basis variables (non-basic variables are 0).

Theorem 31 (Cramers Rule)

Let M be a matrix with $\det(M) \neq 0$. Then the solution to the system $Mx = b$ is given by

$$x_j = \frac{\det(M_j)}{\det(M)} \enspace ,$$

where M_j is the matrix obtained from M by replacing the j-th column by the vector b.

Proof:

Proof:

- Define

$$X_j = \begin{pmatrix} | & & | & | & | & & | \\ e_1 & \cdots & e_{j-1} & x & e_{j+1} & \cdots & e_n \\ | & & | & | & | & & | \end{pmatrix}$$

Note that expanding along the j-th column gives that $\det(X_j) = x_j$.

- Further, we have

$$MX_j = \begin{pmatrix} | & & | & | & | & & | \\ Me_1 & \cdots & Me_{j-1} & Mx & Me_{j+1} & \cdots & Me_n \\ | & & | & | & | & & | \end{pmatrix} = M_j$$

- Hence,

$$x_j = \det(X_j) = \frac{\det(M_j)}{\det(M)}$$

Proof:

- Define

$$X_j = \begin{pmatrix} | & & | & | & | & & | \\ e_1 & \cdots & e_{j-1} & x & e_{j+1} & \cdots & e_n \\ | & & | & | & | & & | \end{pmatrix}$$

Note that expanding along the j-th column gives that $\det(X_j) = x_j$.

- Further, we have

$$MX_j = \begin{pmatrix} | & & | & | & | & & | \\ Me_1 & \cdots & Me_{j-1} & Mx & Me_{j+1} & \cdots & Me_n \\ | & & | & | & | & & | \end{pmatrix} = M_j$$

- Hence,

$$x_j = \det(X_j) = \frac{\det(M_j)}{\det(M)}$$

Proof:

- Define

$$X_j = \begin{pmatrix} | & & | & | & | & & | \\ e_1 & \cdots & e_{j-1} & x & e_{j+1} & \cdots & e_n \\ | & & | & | & | & & | \end{pmatrix}$$

  Note that expanding along the j-th column gives that $\det(X_j) = x_j$.

- Further, we have

$$MX_j = \begin{pmatrix} | & & | & | & | & & | \\ Me_1 & \cdots & Me_{j-1} & Mx & Me_{j+1} & \cdots & Me_n \\ | & & | & | & | & & | \end{pmatrix} = M_j$$

- Hence,

$$x_j = \det(X_j) = \frac{\det(M_j)}{\det(M)}$$

Proof:

- Define

$$X_j = \begin{pmatrix} | & & | & | & | & & | \\ e_1 & \cdots & e_{j-1} & x & e_{j+1} & \cdots & e_n \\ | & & | & | & | & & | \end{pmatrix}$$

  Note that expanding along the j-th column gives that $\det(X_j) = x_j$.

- Further, we have

$$MX_j = \begin{pmatrix} | & & | & | & | & & | \\ Me_1 & \cdots & Me_{j-1} & Mx & Me_{j+1} & \cdots & Me_n \\ | & & | & | & | & & | \end{pmatrix} = M_j$$

- Hence,

$$x_j = \det(X_j) = \frac{\det(M_j)}{\det(M)}$$

Bounding the Determinant

Let Z be the maximum absolute entry occuring in $\bar{A}, \bar{b}$ or c. Let C denote the matrix obtained from $\bar{A}_B$ by replacing the j-th column with vector $\bar{b}$.

Observe that

$$|\det(C)|$$

Bounding the Determinant

Let Z be the maximum absolute entry occuring in $\bar{A}, \bar{b}$ or c. Let C denote the matrix obtained from $\bar{A}_B$ by replacing the j-th column with vector $\bar{b}$.

Observe that

$$|\det(C)| = \left| \sum_{\pi \in S_m} \prod_{1 \le i \le m} \operatorname{sgn}(\pi) C_{i\pi(i)} \right|$$

Bounding the Determinant

Let Z be the maximum absolute entry occuring in $\bar{A}, \bar{b}$ or c. Let C denote the matrix obtained from $\bar{A}_B$ by replacing the j-th column with vector $\bar{b}$.

Observe that

$$|\det(C)| = \left| \sum_{\pi \in S_m} \prod_{1 \le i \le m} \operatorname{sgn}(\pi) C_{i\pi(i)} \right| \le \sum_{\pi \in S_m} \prod_{1 \le i \le m} |C_{i\pi(i)}|$$

Bounding the Determinant

Let Z be the maximum absolute entry occuring in $\bar{A}, \bar{b}$ or c. Let C denote the matrix obtained from $\bar{A}_B$ by replacing the j-th column with vector $\bar{b}$.

Observe that

$$
\begin{aligned}
|\det(C)| &= \left| \sum_{\pi \in S_m} \prod_{1 \le i \le m} \operatorname{sgn}(\pi) C_{i\pi(i)} \right| \\
&\le \sum_{\pi \in S_m} \prod_{1 \le i \le m} |C_{i\pi(i)}| \\
&\le m! \cdot Z^m \enspace .
\end{aligned}
$$

Bounding the Determinant

Alternatively, Hadamards inequality gives

$$|\det(C)|$$

Bounding the Determinant

Alternatively, Hadamards inequality gives

$$|\det(C)| \leq \prod_{i=1}^{m} \|C_{*i}\|$$

Bounding the Determinant

Alternatively, Hadamards inequality gives

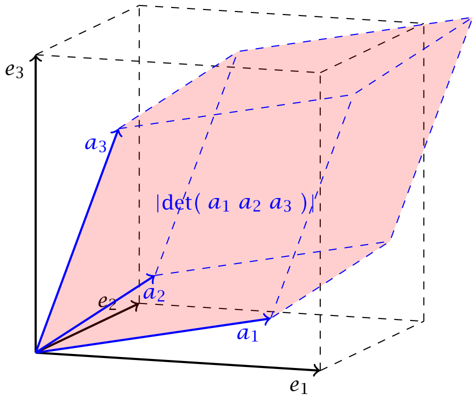
$$|\det(C)| \leq \prod_{i=1}^{m} \|C_{*i}\| \leq \prod_{i=1}^{m} (\sqrt{m}Z)$$

Bounding the Determinant

Alternatively, Hadamards inequality gives

$$
\begin{aligned}
|\det(C)| &\le \prod_{i=1}^{m} \|C_{*i}\| \le \prod_{i=1}^{m} (\sqrt{m}Z) \\
&\le m^{m/2} Z^m \quad .
\end{aligned}
$$

Hadamards Inequality



Hadamards inequality says that the volume of the red parallelepiped (Spat) is smaller than the volume in the black cube (if $\|e_1\| = \|a_1\|$, $\|e_2\| = \|a_2\|$, $\|e_3\| = \|a_3\|$).

Ensuring Conditions

Given a standard minimization LP

$$\begin{array}{lrcl} \min & c^t x & & \\ \text{s.t.} & Ax & \geq & b \\ & x & \geq & 0 \end{array}$$

how can we obtain an LP of the required form?

- **Compute a lower bound on $c^t x$ for any basic feasible solution.** Add the constraint $c^t x \geq -mZ(m! \cdot Z^m) - 1$. Note that this constraint is superfluous unless the LP is unbounded.

Ensuring Conditions

Compute an optimum basis for the new LP.

- If the cost is $c^t x = -(mZ)(m! \cdot Z^m) - 1$ we know that the original LP is unbounded.
- Otw. we have an optimum basis.

In the following we use $\mathcal{H}$ to denote the set of all constraints apart from the constraint $c^t x \geq -mZ(m! \cdot Z^m) - 1$.

We give a routine SeidelLP($\mathcal{H}, d$) that is given a set $\mathcal{H}$ of explicit, non-degenerate constraints over d variables, and minimizes $c^t x$ over all feasible points.

In addition it obeys the implicit constraint $c^t x \geq -(mZ)(m! \cdot Z^m) - 1$.

In the following we use $\mathcal{H}$ to denote the set of all constraints apart from the constraint $c^t x \geq -mZ(m! \cdot Z^m) - 1$.

We give a routine SeidelLP($\mathcal{H}, d$) that is given a set $\mathcal{H}$ of explicit, non-degenerate constraints over d variables, and minimizes $c^t x$ over all feasible points.

In addition it obeys the implicit constraint $c^t x \geq -(mZ)(m! \cdot Z^m) - 1$.

In the following we use $\mathcal{H}$ to denote the set of all constraints apart from the constraint $c^t x \geq -mZ(m! \cdot Z^m) - 1$.

We give a routine SeidelLP$(\mathcal{H}, d)$ that is given a set $\mathcal{H}$ of explicit, non-degenerate constraints over d variables, and minimizes $c^t x$ over all feasible points.

In addition it obeys the implicit constraint $c^t x \geq -(mZ)(m! \cdot Z^m) - 1$.

In the following we use $\mathcal{H}$ to denote the set of all constraints apart from the constraint $c^t x \geq -mZ(m! \cdot Z^m) - 1$.

We give a routine SeidelLP$(\mathcal{H}, d)$ that is given a set $\mathcal{H}$ of explicit, non-degenerate constraints over d variables, and minimizes $c^t x$ over all feasible points.

In addition it obeys the implicit constraint
$c^t x \geq -(mZ)(m! \cdot Z^m) - 1$.

Algorithm 1 SeidelLP($\mathcal{H}, d$)

1: **if** $d = 1$ **then** solve 1-dimensional problem and return;

Algorithm 1 SeidelLP($\mathcal{H}, d$)

1: **if** $d = 1$ **then** solve 1-dimensional problem and return;
2: **if** $\mathcal{H} = \emptyset$ **then** return x on implicit constraint hyperplane

Algorithm 1 SeidelLP($\mathcal{H}, d$)

if $d = 1$ **then** solve 1-dimensional problem and return;
if $\mathcal{H} = \emptyset$ **then** return x on implicit constraint hyperplane
choose random constraint $h \in \mathcal{H}$

Algorithm 1 SeidelLP($\mathcal{H}, d$)

if $d = 1$ **then** solve 1-dimensional problem and return;
if $\mathcal{H} = \emptyset$ **then** return x on implicit constraint hyperplane
choose random constraint $h \in \mathcal{H}$
$\hat{\mathcal{H}} \leftarrow \mathcal{H} \setminus \{h\}$

Algorithm 1 SeidelLP($\mathcal{H}, d$)

```
if d = 1 then solve 1-dimensional problem and return;
if H = ∅ then return x on implicit constraint hyperplane
choose random constraint h ∈ H
Ĥ ← H \ {h}
x̂* ← SeidelLP(Ĥ, d)
```

Algorithm 1 SeidelLP($\mathcal{H}, d$)

if $d = 1$ **then** solve 1-dimensional problem and return;
if $\mathcal{H} = \emptyset$ **then** return x on implicit constraint hyperplane
choose random constraint $h \in \mathcal{H}$
$\hat{\mathcal{H}} \leftarrow \mathcal{H} \setminus \{h\}$
$\hat{x}^* \leftarrow$ SeidelLP($\hat{\mathcal{H}}, d$)
if $\hat{x}^* =$ infeasible **then return** infeasible

Algorithm 1 SeidelLP($\mathcal{H}, d$)

if $d = 1$ **then** solve 1-dimensional problem and return;
if $\mathcal{H} = \emptyset$ **then** return x on implicit constraint hyperplane
choose random constraint $h \in \mathcal{H}$
$\hat{\mathcal{H}} \leftarrow \mathcal{H} \setminus \{h\}$
$\hat{x}^* \leftarrow$ SeidelLP($\hat{\mathcal{H}}, d$)
if $\hat{x}^* =$ infeasible **then return** infeasible
if $\hat{x}^*$ fulfills h **then return** $\hat{x}^*$

Algorithm 1 SeidelLP($\mathcal{H}, d$)

```
if d = 1 then solve 1-dimensional problem and return;
if H = ∅ then return x on implicit constraint hyperplane
choose random constraint h ∈ H
Ĥ ← H \ {h}
x̂* ← SeidelLP(Ĥ, d)
if x̂* = infeasible then return infeasible
if x̂* fulfills h then return x̂*
// optimal solution fulfills h with equality, i.e., A_h x = b_h
```

Algorithm 1 SeidelLP($\mathcal{H}, d$)

if $d = 1$ **then** solve 1-dimensional problem and return;
if $\mathcal{H} = \emptyset$ **then** return x on implicit constraint hyperplane
choose random constraint $h \in \mathcal{H}$
$\hat{\mathcal{H}} \leftarrow \mathcal{H} \setminus \{h\}$
$\hat{x}^* \leftarrow$ SeidelLP($\hat{\mathcal{H}}, d$)
if $\hat{x}^* =$ infeasible **then return** infeasible
if $\hat{x}^*$ fulfills h **then return** $\hat{x}^*$
// optimal solution fulfills h with equality, i.e., $A_h x = b_h$
solve $A_h x = b_h$ for some variable x_ℓ;
eliminate x_ℓ in constraints from $\hat{\mathcal{H}}$ and in implicit constr.;

Algorithm 1 SeidelLP($\mathcal{H}, d$)

if $d = 1$ **then** solve 1-dimensional problem and return;
if $\mathcal{H} = \emptyset$ **then** return x on implicit constraint hyperplane
choose random constraint $h \in \mathcal{H}$
$\hat{\mathcal{H}} \leftarrow \mathcal{H} \setminus \{h\}$
$\hat{x}^* \leftarrow$ SeidelLP($\hat{\mathcal{H}}, d$)
if $\hat{x}^* =$ infeasible **then return** infeasible
if $\hat{x}^*$ fulfills h **then return** $\hat{x}^*$
// optimal solution fulfills h with equality, i.e., $A_h x = b_h$
solve $A_h x = b_h$ for some variable x_ℓ;
eliminate x_ℓ in constraints from $\hat{\mathcal{H}}$ and in implicit constr.;
$\hat{x}^* \leftarrow$ SeidelLP($\hat{\mathcal{H}}, d - 1$)

Algorithm 1 SeidelLP($\mathcal{H}, d$)

if $d = 1$ **then** solve 1-dimensional problem and return;
if $\mathcal{H} = \emptyset$ **then** return x on implicit constraint hyperplane
choose random constraint $h \in \mathcal{H}$
$\hat{\mathcal{H}} \leftarrow \mathcal{H} \setminus \{h\}$
$\hat{x}^* \leftarrow$ SeidelLP($\hat{\mathcal{H}}, d$)
if $\hat{x}^* =$ infeasible **then return** infeasible
if $\hat{x}^*$ fulfills h **then return** $\hat{x}^*$
// optimal solution fulfills h with equality, i.e., $A_h x = b_h$
solve $A_h x = b_h$ for some variable x_ℓ;
eliminate x_ℓ in constraints from $\hat{\mathcal{H}}$ and in implicit constr.;
$\hat{x}^* \leftarrow$ SeidelLP($\hat{\mathcal{H}}, d - 1$)
if $\hat{x}^* =$ infeasible **then**
    return infeasible
else
    add the value of x_ℓ to $\hat{x}^*$ and return the solution

8 Seidels LP-algorithm

- If $d = 1$ we can solve the 1-dimensional problem in time $\mathcal{O}(m)$.
- If $d > 1$ and $m = 0$ we take time $\mathcal{O}(d)$ to return d-dimensional vector x.
- The first recursive call takes time $T(m-1, d)$ for the call plus $\mathcal{O}(d)$ for checking whether the solution fulfills h.
- If we are unlucky and $\hat{x}^*$ does not fulfill h we need time $\mathcal{O}(d(m+1)) = \mathcal{O}(dm)$ to eliminate x_ℓ. Then we make a recursive call that takes time $T(m-1, d-1)$.
- The probability of being unlucky is at most d/m as there are at most d constraints whose removal will decrease the objective function

8 Seidels LP-algorithm

- If $d = 1$ we can solve the 1-dimensional problem in time $\mathcal{O}(m)$.
- If $d > 1$ and $m = 0$ we take time $\mathcal{O}(d)$ to return d-dimensional vector x.
- The first recursive call takes time $T(m-1, d)$ for the call plus $\mathcal{O}(d)$ for checking whether the solution fulfills h.
- If we are unlucky and $\hat{x}^*$ does not fulfill h we need time $\mathcal{O}(d(m+1)) = \mathcal{O}(dm)$ to eliminate x_ℓ. Then we make a recursive call that takes time $T(m-1, d-1)$.
- The probability of being unlucky is at most d/m as there are at most d constraints whose removal will decrease the objective function

8 Seidels LP-algorithm

- If $d = 1$ we can solve the 1-dimensional problem in time $\mathcal{O}(m)$.
- If $d > 1$ and $m = 0$ we take time $\mathcal{O}(d)$ to return d-dimensional vector x.
- The first recursive call takes time $T(m-1, d)$ for the call plus $\mathcal{O}(d)$ for checking whether the solution fulfills h.
- If we are unlucky and $\hat{x}^*$ does not fulfill h we need time $\mathcal{O}(d(m+1)) = \mathcal{O}(dm)$ to eliminate x_ℓ. Then we make a recursive call that takes time $T(m-1, d-1)$.
- The probability of being unlucky is at most d/m as there are at most d constraints whose removal will decrease the objective function

8 Seidels LP-algorithm

- If $d = 1$ we can solve the 1-dimensional problem in time $\mathcal{O}(m)$.
- If $d > 1$ and $m = 0$ we take time $\mathcal{O}(d)$ to return d-dimensional vector x.
- The first recursive call takes time $T(m-1, d)$ for the call plus $\mathcal{O}(d)$ for checking whether the solution fulfills h.
- If we are unlucky and $\hat{x}^*$ does not fulfill h we need time $\mathcal{O}(d(m+1)) = \mathcal{O}(dm)$ to eliminate x_ℓ. Then we make a recursive call that takes time $T(m-1, d-1)$.
- The probability of being unlucky is at most d/m as there are at most d constraints whose removal will decrease the objective function

8 Seidels LP-algorithm

- If $d = 1$ we can solve the 1-dimensional problem in time $\mathcal{O}(m)$.
- If $d > 1$ and $m = 0$ we take time $\mathcal{O}(d)$ to return d-dimensional vector x.
- The first recursive call takes time $T(m-1, d)$ for the call plus $\mathcal{O}(d)$ for checking whether the solution fulfills h.
- If we are unlucky and $\hat{x}^*$ does not fulfill h we need time $\mathcal{O}(d(m+1)) = \mathcal{O}(dm)$ to eliminate x_ℓ. Then we make a recursive call that takes time $T(m-1, d-1)$.
- The probability of being unlucky is at most d/m as there are at most d constraints whose removal will decrease the objective function

8 Seidels LP-algorithm

This gives the recurrence

$$T(m,d) = \begin{cases} \mathcal{O}(m) & \text{if } d = 1 \\ \mathcal{O}(d) & \text{if } d > 1 \text{ and } m = 0 \\ \mathcal{O}(d) + T(m-1,d) + & \\ \frac{d}{m}(\mathcal{O}(dm) + T(m-1,d-1)) & \text{otw.} \end{cases}$$

Note that $T(m,d)$ denotes the expected running time.

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$-notations.

$$T(m,d) = \begin{cases} Cm & \text{if } d = 1 \\ Cd & \text{if } d > 1 \text{ and } m = 0 \\ Cd + T(m-1,d) + & \\ \frac{d}{m}(Cdm + T(m-1,d-1)) & \text{otw.} \end{cases}$$

Note that $T(m,d)$ denotes the expected running time.

8 Seidels LP-algorithm

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$-notations.

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$-notations.

We show $T(m,d) \leq Cf(d)\max\{1,m\}$.

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$-notations.

We show $T(m,d) \le Cf(d)\max\{1,m\}$.

d = 1:

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$-notations.

We show $T(m,d) \leq Cf(d)\max\{1,m\}$.

d = 1:

$$T(m,1)$$

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$-notations.

We show $T(m,d) \le Cf(d)\max\{1,m\}$.

$d = 1$:

$$T(m,1) \le Cm$$

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$-notations.

We show $T(m,d) \le Cf(d)\max\{1,m\}$.

$\boldsymbol{d} = \mathbf{1}$**:**

$$T(m,1) \le Cm \le Cf(1)\max\{1,m\}$$

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$-notations.

We show $T(m,d) \leq Cf(d)\max\{1,m\}$.

$\boldsymbol{d = 1}$**:**

$$T(m,1) \leq Cm \leq Cf(1)\max\{1,m\} \text{ for } f(1) \geq 1$$

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$-notations.

We show $T(m,d) \le Cf(d)\max\{1,m\}$.

$\boldsymbol{d = 1}$**:**

$T(m,1) \le Cm \le Cf(1)\max\{1,m\}$ for $f(1) \ge 1$

$\boldsymbol{d > 1; m = 0}$ **:**

$T(0,d) \le \mathcal{O}(d)$

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$-notations.

We show $T(m,d) \leq Cf(d)\max\{1,m\}$.

$\boldsymbol{d = 1}$**:**

$T(m,1) \leq Cm \leq Cf(1)\max\{1,m\}$ for $f(1) \geq 1$

$\boldsymbol{d > 1; m = 0}$ **:**

$T(0,d) \leq \mathcal{O}(d) \leq Cd$

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$-notations.

We show $T(m,d) \le Cf(d)\max\{1,m\}$.

$d = 1$:

$T(m,1) \le Cm \le Cf(1)\max\{1,m\}$ for $f(1) \ge 1$

$d > 1; m = 0$:

$T(0,d) \le \mathcal{O}(d) \le Cd \le Cf(d)\max\{1,m\}$

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$-notations.

We show $T(m,d) \le Cf(d)\max\{1,m\}$.

$\boldsymbol{d = 1}$**:**

$$T(m,1) \le Cm \le Cf(1)\max\{1,m\} \text{ for } f(1) \ge 1$$

$\boldsymbol{d > 1; m = 0}$ **:**

$$T(0,d) \le \mathcal{O}(d) \le Cd \le Cf(d)\max\{1,m\} \text{ for } f(d) \ge d$$

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$-notations.

We show $T(m,d) \le Cf(d)\max\{1,m\}$.

$d = 1$:

$$T(m,1) \le Cm \le Cf(1)\max\{1,m\} \text{ for } f(1) \ge 1$$

$d > 1; m = 0$:

$$T(0,d) \le \mathcal{O}(d) \le Cd \le Cf(d)\max\{1,m\} \text{ for } f(d) \ge d$$

$d > 1; m = 1$:

$$T(1,d) = \mathcal{O}(d) + T(0,d) + d\Big(\mathcal{O}(d) + T(0,d-1)\Big)$$

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$-notations.

We show $T(m,d) \le Cf(d)\max\{1,m\}$.

$d = 1$:

$$T(m,1) \le Cm \le Cf(1)\max\{1,m\} \text{ for } f(1) \ge 1$$

$d > 1; m = 0$:

$$T(0,d) \le \mathcal{O}(d) \le Cd \le Cf(d)\max\{1,m\} \text{ for } f(d) \ge d$$

$d > 1; m = 1$:

$$\begin{aligned} T(1,d) &= \mathcal{O}(d) + T(0,d) + d\Big(\mathcal{O}(d) + T(0,d-1)\Big) \\ &\le Cd + Cd + Cd^2 + dCf(d-1) \end{aligned}$$

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$-notations.

We show $T(m,d) \le Cf(d)\max\{1,m\}$.

$\boldsymbol{d = 1}$**:**

$$T(m,1) \le Cm \le Cf(1)\max\{1,m\} \text{ for } f(1) \ge 1$$

$\boldsymbol{d > 1; m = 0}$ **:**

$$T(0,d) \le \mathcal{O}(d) \le Cd \le Cf(d)\max\{1,m\} \text{ for } f(d) \ge d$$

$\boldsymbol{d > 1; m = 1}$ **:**

$$\begin{aligned} T(1,d) &= \mathcal{O}(d) + T(0,d) + d\Big(\mathcal{O}(d) + T(0,d-1)\Big) \\ &\le Cd + Cd + Cd^2 + dCf(d-1) \\ &\le Cf(d)\max\{1,m\} \end{aligned}$$

8 Seidels LP-algorithm

Let C be the largest constant in the $\mathcal{O}$-notations.

We show $T(m,d) \le Cf(d)\max\{1,m\}$.

$d = 1$:

$$T(m,1) \le Cm \le Cf(1)\max\{1,m\} \text{ for } f(1) \ge 1$$

$d > 1; m = 0$:

$$T(0,d) \le \mathcal{O}(d) \le Cd \le Cf(d)\max\{1,m\} \text{ for } f(d) \ge d$$

$d > 1; m = 1$:

$$\begin{aligned} T(1,d) &= \mathcal{O}(d) + T(0,d) + d\Big(\mathcal{O}(d) + T(0,d-1)\Big) \\ &\le Cd + Cd + Cd^2 + dCf(d-1) \\ &\le Cf(d)\max\{1,m\} \text{ for } f(d) \ge 3d^2 + df(d-1) \end{aligned}$$

8 Seidels LP-algorithm

$\boldsymbol{d > 1; m > 1:}$

(by induction hypothesis statm. true for $d' < d, m' \geq 0$;
and for $d' = d$, $m' < m$)

8 Seidels LP-algorithm

$d > 1; m > 1$:
(by induction hypothesis statm. true for $d' < d, m' \geq 0$;
and for $d' = d$, $m' < m$)

$$T(m,d) = \mathcal{O}(d) + T(m-1,d) + \frac{d}{m}\Big(\mathcal{O}(dm) + T(m-1,d-1)\Big)$$

8 Seidels LP-algorithm

$d > 1; m > 1$:
(by induction hypothesis statm. true for $d' < d, m' \geq 0$;
and for $d' = d$, $m' < m$)

$$T(m,d) = \mathcal{O}(d) + T(m-1,d) + \frac{d}{m}\Big(\mathcal{O}(dm) + T(m-1,d-1)\Big)$$

$$\leq Cd + Cf(d)(m-1) + Cd^2 + \frac{d}{m}Cf(d-1)(m-1)$$

8 Seidels LP-algorithm

$\boldsymbol{d > 1; m > 1:}$

(by induction hypothesis statm. true for $d' < d, m' \geq 0$;
and for $d' = d$, $m' < m$)

$$\begin{aligned}T(m,d) &= \mathcal{O}(d) + T(m-1,d) + \frac{d}{m}\Big(\mathcal{O}(dm) + T(m-1,d-1)\Big)\\ &\leq Cd + Cf(d)(m-1) + Cd^2 + \frac{d}{m}Cf(d-1)(m-1)\\ &\leq 2Cd^2 + Cf(d)(m-1) + dCf(d-1)\end{aligned}$$

8 Seidels LP-algorithm

$\boldsymbol{d > 1; m > 1:}$

(by induction hypothesis statm. true for $d' < d, m' \geq 0$;
and for $d' = d$, $m' < m$)

$$\begin{aligned}
T(m,d) &= \mathcal{O}(d) + T(m-1,d) + \frac{d}{m}\Big(\mathcal{O}(dm) + T(m-1,d-1)\Big)\\
&\leq Cd + Cf(d)(m-1) + Cd^2 + \frac{d}{m}Cf(d-1)(m-1)\\
&\leq 2Cd^2 + Cf(d)(m-1) + dCf(d-1)\\
&\leq Cf(d)m
\end{aligned}$$

8 Seidels LP-algorithm

$\boldsymbol{d > 1; m > 1:}$

(by induction hypothesis statm. true for $d' < d, m' \geq 0$;
and for $d' = d$, $m' < m$)

$$
\begin{aligned}
T(m,d) &= \mathcal{O}(d) + T(m-1,d) + \frac{d}{m}\Big(\mathcal{O}(dm) + T(m-1,d-1)\Big) \\
&\leq Cd + Cf(d)(m-1) + Cd^2 + \frac{d}{m}Cf(d-1)(m-1) \\
&\leq 2Cd^2 + Cf(d)(m-1) + dCf(d-1) \\
&\leq Cf(d)m
\end{aligned}
$$

if $f(d) \geq df(d-1) + 2d^2$.

8 Seidels LP-algorithm

- Define $f(1) = 3 \cdot 1^2$ and $f(d) = df(d-1) + 3d^2$ for $d > 1$.

8 Seidels LP-algorithm

- Define $f(1) = 3 \cdot 1^2$ and $f(d) = df(d-1) + 3d^2$ for $d > 1$.

Then

$$f(d)$$

8 Seidels LP-algorithm

- Define $f(1) = 3 \cdot 1^2$ and $f(d) = df(d-1) + 3d^2$ for $d > 1$.

Then

$$f(d) = 3d^2 + df(d-1)$$

8 Seidels LP-algorithm

- Define $f(1) = 3 \cdot 1^2$ and $f(d) = df(d-1) + 3d^2$ for $d > 1$.

Then

$$
\begin{aligned}
f(d) &= 3d^2 + df(d-1) \\
&= 3d^2 + d\Big[3(d-1)^2 + (d-1)f(d-2)\Big]
\end{aligned}
$$

8 Seidels LP-algorithm

- Define $f(1) = 3 \cdot 1^2$ and $f(d) = df(d-1) + 3d^2$ for $d > 1$.

Then

$$\begin{aligned} f(d) &= 3d^2 + df(d-1) \\ &= 3d^2 + d\left[3(d-1)^2 + (d-1)f(d-2)\right] \\ &= 3d^2 + d\left[3(d-1)^2 + (d-1)\left[3(d-2)^2 + (d-2)f(d-3)\right]\right] \end{aligned}$$

8 Seidels LP-algorithm

- Define $f(1) = 3 \cdot 1^2$ and $f(d) = df(d-1) + 3d^2$ for $d > 1$.

Then

$$
\begin{aligned}
f(d) &= 3d^2 + df(d-1) \\
&= 3d^2 + d\left[3(d-1)^2 + (d-1)f(d-2)\right] \\
&= 3d^2 + d\left[3(d-1)^2 + (d-1)\left[3(d-2)^2 + (d-2)f(d-3)\right]\right] \\
&= 3d^2 + 3d(d-1)^2 + 3d(d-1)(d-2)^2 + \dots \\
&\quad + 3d(d-1)(d-2) \cdot \dots \cdot 4 \cdot 3 \cdot 1^2
\end{aligned}
$$

8 Seidels LP-algorithm

- Define $f(1) = 3 \cdot 1^2$ and $f(d) = df(d-1) + 3d^2$ for $d > 1$.

Then

$$\begin{aligned}
f(d) &= 3d^2 + df(d-1) \\
&= 3d^2 + d\Big[3(d-1)^2 + (d-1)f(d-2)\Big] \\
&= 3d^2 + d\Big[3(d-1)^2 + (d-1)\Big[3(d-2)^2 + (d-2)f(d-3)\Big]\Big] \\
&= 3d^2 + 3d(d-1)^2 + 3d(d-1)(d-2)^2 + \dots \\
&\quad + 3d(d-1)(d-2)\cdot \dots \cdot 4 \cdot 3 \cdot 1^2 \\
&= 3d!\left(\frac{d^2}{d!} + \frac{(d-1)^2}{(d-1)!} + \frac{(d-2)^2}{(d-2)!} + \dots\right)
\end{aligned}$$

8 Seidels LP-algorithm

- Define $f(1) = 3 \cdot 1^2$ and $f(d) = df(d-1) + 3d^2$ for $d > 1$.

Then

$$\begin{aligned}
f(d) &= 3d^2 + df(d-1) \\
&= 3d^2 + d\left[3(d-1)^2 + (d-1)f(d-2)\right] \\
&= 3d^2 + d\left[3(d-1)^2 + (d-1)\left[3(d-2)^2 + (d-2)f(d-3)\right]\right] \\
&= 3d^2 + 3d(d-1)^2 + 3d(d-1)(d-2)^2 + \dots \\
&\quad + 3d(d-1)(d-2) \cdot \ldots \cdot 4 \cdot 3 \cdot 1^2 \\
&= 3d!\left(\frac{d^2}{d!} + \frac{(d-1)^2}{(d-1)!} + \frac{(d-2)^2}{(d-2)!} + \dots\right) \\
&= \mathcal{O}(d!)
\end{aligned}$$

8 Seidels LP-algorithm

- Define $f(1) = 3 \cdot 1^2$ and $f(d) = df(d-1) + 3d^2$ for $d > 1$.

Then

$$
\begin{aligned}
f(d) &= 3d^2 + df(d-1) \\
&= 3d^2 + d\Big[3(d-1)^2 + (d-1)f(d-2)\Big] \\
&= 3d^2 + d\Big[3(d-1)^2 + (d-1)\Big[3(d-2)^2 + (d-2)f(d-3)\Big]\Big] \\
&= 3d^2 + 3d(d-1)^2 + 3d(d-1)(d-2)^2 + \dots \\
&\quad + 3d(d-1)(d-2)\cdot \ldots \cdot 4 \cdot 3 \cdot 1^2 \\
&= 3d!\left(\frac{d^2}{d!} + \frac{(d-1)^2}{(d-1)!} + \frac{(d-2)^2}{(d-2)!} + \dots\right) \\
&= \mathcal{O}(d!)
\end{aligned}
$$

since $\sum_{i\geq 1} \frac{i^2}{i!}$ is a constant.

Complexity

LP Feasibility Problem (LP feasibility)

- Given $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$. Does there exist $x \in \mathbb{R}$ with $Ax = b$, $x \geq 0$?
- Note that allowing A, b to contain rational numbers does not make a difference, as we can multiply every number by a suitable large constant so that everything becomes integral but the feasible region does not change.

Is this problem in NP or even in P?

Complexity

LP Feasibility Problem (LP feasibility)

- Given $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$. Does there exist $x \in \mathbb{R}$ with $Ax = b$, $x \geq 0$?
- Note that allowing A, b to contain rational numbers does not make a difference, as we can multiply every number by a suitable large constant so that everything becomes integral but the feasible region does not change.

Is this problem in NP or even in P?

Complexity

LP Feasibility Problem (LP feasibility)

- Given $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$. Does there exist $x \in \mathbb{R}$ with $Ax = b$, $x \geq 0$?
- Note that allowing A, b to contain rational numbers does not make a difference, as we can multiply every number by a suitable large constant so that everything becomes integral but the feasible region does not change.

Is this problem in NP or even in P?

The Bit Model

Input size

- The number of bits to represent a number $a \in \mathbb{Z}$ is

$$\lceil \log_2(|a|) \rceil + 1$$

- Let for an $m \times n$ matrix M, $L(M)$ denote the number of bits required to encode all the numbers in M.

$$L(M) := \sum_{i,j} \lceil \log_2(|m_{ij}|) + 1 \rceil$$

- In the following we assume that input matrices are encoded in a standard way, where each number is encoded in binary and then suitable separators are added in order to separate distinct number from each other.
- Then the input length is $\Theta(L([A|b]))$.

The Bit Model

Input size

- The number of bits to represent a number $a \in \mathbb{Z}$ is

$$\lceil \log_2(|a|) \rceil + 1$$

- Let for an $m \times n$ matrix M, $L(M)$ denote the number of bits required to encode all the numbers in M.

$$L(M) := \sum_{i,j} \lceil \log_2(|m_{ij}|) + 1 \rceil$$

- In the following we assume that input matrices are encoded in a standard way, where each number is encoded in binary and then suitable separators are added in order to separate distinct number from each other.
- Then the input length is $\Theta(L([A|b]))$.

The Bit Model

Input size

- The number of bits to represent a number $a \in \mathbb{Z}$ is

$$\lceil \log_2(|a|) \rceil + 1$$

- Let for an $m \times n$ matrix M, $L(M)$ denote the number of bits required to encode all the numbers in M.

$$L(M) := \sum_{i,j} \lceil \log_2(|m_{ij}|) + 1 \rceil$$

- In the following we assume that input matrices are encoded in a standard way, where each number is encoded in binary and then suitable separators are added in order to separate distinct number from each other.

- Then the input length is $\Theta(L([A|b]))$.

The Bit Model

Input size

- The number of bits to represent a number $a \in \mathbb{Z}$ is

$$\lceil \log_2(|a|) \rceil + 1$$

- Let for an $m \times n$ matrix M, $L(M)$ denote the number of bits required to encode all the numbers in M.

$$L(M) := \sum_{i,j} \lceil \log_2(|m_{ij}|) + 1 \rceil$$

- In the following we assume that input matrices are encoded in a standard way, where each number is encoded in binary and then suitable separators are added in order to separate distinct number from each other.
- Then the input length is $\Theta(L([A|b]))$.

- In the following we sometimes refer to $L := L([A|b])$ as the input size (even though the real input size is something in $\Theta(L([A|b]))$).
- In order to show that LP-decision is in NP we show that if there is a solution x then there exists a small solution for which feasibility can be verified in polynomial time (polynomial in $L([A|b])$).

Suppose that $Ax = b; x \geq 0$ is feasible.

Then there exists a basic feasible solution. This means a set B of basic variables such that

$$x_B = A_B^{-1} b$$

and all other entries in x are 0.

Suppose that $Ax = b; x \geq 0$ is feasible.

Then there exists a basic feasible solution. This means a set B of basic variables such that

$$x_B = A_B^{-1} b$$

and all other entries in x are 0.

Size of a Basic Feasible Solution

Lemma 32

Let $M \in \mathbb{Z}^{m \times m}$ be an invertable matrix and let $b \in \mathbb{Z}^m$. Further define $L' = L([M \mid b]) + n \log_2 n$. Then a solution to $Mx = b$ has rational components x_j of the form $\frac{D_j}{D}$, where $|D_j| \le 2^{L'}$ and $|D| \le 2^{L'}$.

Proof:

Cramers rules says that we can compute x_j as

$$x_j = \frac{\det(M_j)}{\det(M)}$$

where M_j is the matrix obtained from M by replacing the j-th column by the vector b.

Size of a Basic Feasible Solution

Lemma 32

Let $M \in \mathbb{Z}^{m\times m}$ be an invertable matrix and let $b \in \mathbb{Z}^m$. Further define $L' = L([M \mid b]) + n\log_2 n$. Then a solution to $Mx = b$ has rational components x_j of the form $\frac{D_j}{D}$, where $|D_j| \le 2^{L'}$ and $|D| \le 2^{L'}$.

Proof:

Cramers rules says that we can compute x_j as

$$x_j = \frac{\det(M_j)}{\det(M)}$$

where M_j is the matrix obtained from M by replacing the j-th column by the vector b.

Bounding the Determinant

Let $X = A_B$. Then

$$|\det(X)|$$

Bounding the Determinant

Let $X = A_B$. Then

$$|\det(X)| = \left|\sum_{\pi\in S_n}\prod_{1\le i\le n}\operatorname{sgn}(\pi)X_{i\pi(i)}\right|$$

Bounding the Determinant

Let $X = A_B$. Then

$$\begin{aligned}|\det(X)| &= \left|\sum_{\pi\in S_n}\prod_{1\le i\le n}\operatorname{sgn}(\pi)X_{i\pi(i)}\right|\\ &\le \sum_{\pi\in S_n}\prod_{1\le i\le n}|X_{i\pi(i)}|\end{aligned}$$

Bounding the Determinant

Let $X = A_B$. Then

$$\begin{aligned}|\det(X)| &= \left|\sum_{\pi\in S_n}\prod_{1\le i\le n}\operatorname{sgn}(\pi)X_{i\pi(i)}\right| \\ &\le \sum_{\pi\in S_n}\prod_{1\le i\le n}|X_{i\pi(i)}| \\ &\le n!\cdot 2^{L([A|b])}\end{aligned}$$

Bounding the Determinant

Let $X = A_B$. Then

$$
\begin{aligned}
|\det(X)| &= \left| \sum_{\pi \in S_n} \prod_{1 \le i \le n} \operatorname{sgn}(\pi) X_{i\pi(i)} \right| \\
&\le \sum_{\pi \in S_n} \prod_{1 \le i \le n} |X_{i\pi(i)}| \\
&\le n! \cdot 2^{L([A|b])} \le n^n 2^L
\end{aligned}
$$

Bounding the Determinant

Let $X = A_B$. Then

$$\begin{aligned}|\det(X)| &= \left|\sum_{\pi\in S_n}\prod_{1\le i\le n}\operatorname{sgn}(\pi)X_{i\pi(i)}\right|\\ &\le \sum_{\pi\in S_n}\prod_{1\le i\le n}|X_{i\pi(i)}|\\ &\le n!\cdot 2^{L([A|b])} \le n^n 2^L \le 2^{L'}\ .\end{aligned}$$

Bounding the Determinant

Let $X = A_B$. Then

$$
\begin{aligned}
|\det(X)| &= \left| \sum_{\pi \in S_n} \prod_{1 \le i \le n} \operatorname{sgn}(\pi) X_{i\pi(i)} \right| \\
&\le \sum_{\pi \in S_n} \prod_{1 \le i \le n} |X_{i\pi(i)}| \\
&\le n! \cdot 2^{L([A|b])} \le n^n 2^L \le 2^{L'} \quad .
\end{aligned}
$$

Analogously for $\det(M_j)$.

This means if $Ax = b$, $x \geq 0$ is feasible we only need to consider vectors x where an entry x_j can be represented by a rational number with encoding length polynomial in the input length L.

Hence, the x that we have to guess is of length polynomial in the input-length L.

For a given vector x of polynomial length we can check for feasibility in polynomial time.

Hence, LP feasibility is in NP.

This means if $Ax = b$, $x \geq 0$ is feasible we only need to consider vectors x where an entry x_j can be represented by a rational number with encoding length polynomial in the input length L.

Hence, the x that we have to guess is of length polynomial in the input-length L.

For a given vector x of polynomial length we can check for feasibility in polynomial time.

Hence, LP feasibility is in NP.

This means if $Ax = b$, $x \geq 0$ is feasible we only need to consider vectors x where an entry x_j can be represented by a rational number with encoding length polynomial in the input length L.

Hence, the x that we have to guess is of length polynomial in the input-length L.

For a given vector x of polynomial length we can check for feasibility in polynomial time.

Hence, LP feasibility is in NP.

This means if $Ax = b$, $x \geq 0$ is feasible we only need to consider vectors x where an entry x_j can be represented by a rational number with encoding length polynomial in the input length L.

Hence, the x that we have to guess is of length polynomial in the input-length L.

For a given vector x of polynomial length we can check for feasibility in polynomial time.

Hence, LP feasibility is in NP.

This means if $Ax = b$, $x \geq 0$ is feasible we only need to consider vectors x where an entry x_j can be represented by a rational number with encoding length polynomial in the input length L.

Hence, the x that we have to guess is of length polynomial in the input-length L.

For a given vector x of polynomial length we can check for feasibility in polynomial time.

Hence, LP feasibility is in NP.

Reducing LP-solving to LP decision.

Given an LP $\max\{c^t x \mid Ax = b; x \geq 0\}$ do a binary search for the optimum solution

(Add constraint $c^t x - \delta = M$; $\delta \geq 0$ or ($c^t x \geq M$). Then checking for feasibility shows whether optimum solution is larger or smaller than M).

If the LP is feasible then the binary search finishes in at most

$$\log_2\left(\frac{2n2^{2L'}}{1/2^{L'}}\right) = \mathcal{O}(L') \enspace ,$$

as the range of the search is at most $-n2^{2L'}, \dots, n2^{2L'}$ and the distance between two adjacent values is at least $\frac{1}{\det(A)} \geq \frac{1}{2^{L'}}$.

Here we use $L' = L([A \mid b \mid c]) + n\log_2 n$ (it also includes the encoding size of c).

Reducing LP-solving to LP decision.

Given an LP $\max\{c^t x \mid Ax = b; x \geq 0\}$ do a binary search for the optimum solution

(Add constraint $c^t x - \delta = M$; $\delta \geq 0$ or ($c^t x \geq M$). Then checking for feasibility shows whether optimum solution is larger or smaller than M).

If the LP is feasible then the binary search finishes in at most

$$\log_2\left(\frac{2n2^{2L'}}{1/2^{L'}}\right) = \mathcal{O}(L') \ ,$$

as the range of the search is at most $-n2^{2L'}, \dots, n2^{2L'}$ and the distance between two adjacent values is at least $\frac{1}{\det(A)} \geq \frac{1}{2^{L'}}$.

Here we use $L' = L([A \mid b \mid c]) + n \log_2 n$ (it also includes the encoding size of c).

Reducing LP-solving to LP decision.

Given an LP $\max\{c^t x \mid Ax = b; x \geq 0\}$ do a binary search for the optimum solution

(Add constraint $c^t x - \delta = M$; $\delta \geq 0$ or ($c^t x \geq M$). Then checking for feasibility shows whether optimum solution is larger or smaller than M).

If the LP is feasible then the binary search finishes in at most

$$\log_2\left(\frac{2n2^{2L'}}{1/2^{L'}}\right) = \mathcal{O}(L') \ ,$$

as the range of the search is at most $-n2^{2L'}, \dots, n2^{2L'}$ and the distance between two adjacent values is at least $\frac{1}{\det(A)} \geq \frac{1}{2^{L'}}$.

Here we use $L' = L([A \mid b \mid c]) + n\log_2 n$ (it also includes the encoding size of c).

Reducing LP-solving to LP decision.

Given an LP $\max\{c^t x \mid Ax = b; x \geq 0\}$ do a binary search for the optimum solution

(Add constraint $c^t x - \delta = M$; $\delta \geq 0$ or ($c^t x \geq M$). Then checking for feasibility shows whether optimum solution is larger or smaller than M).

If the LP is feasible then the binary search finishes in at most

$$\log_2\left(\frac{2n2^{2L'}}{1/2^{L'}}\right) = \mathcal{O}(L') \ ,$$

as the range of the search is at most $-n2^{2L'}, \dots, n2^{2L'}$ and the distance between two adjacent values is at least $\frac{1}{\det(A)} \geq \frac{1}{2^{L'}}$.

Here we use $L' = L([A \mid b \mid c]) + n \log_2 n$ (it also includes the encoding size of c).

Reducing LP-solving to LP decision.

Given an LP $\max\{c^t x \mid Ax = b; x \geq 0\}$ do a binary search for the optimum solution

(Add constraint $c^t x - \delta = M$; $\delta \geq 0$ or ($c^t x \geq M$). Then checking for feasibility shows whether optimum solution is larger or smaller than M).

If the LP is feasible then the binary search finishes in at most

$$\log_2\left(\frac{2n2^{2L'}}{1/2^{L'}}\right) = \mathcal{O}(L') \ ,$$

as the range of the search is at most $-n2^{2L'}, \dots, n2^{2L'}$ and the distance between two adjacent values is at least $\frac{1}{\det(A)} \geq \frac{1}{2^{L'}}$.

Here we use $L' = L([A \mid b \mid c]) + n \log_2 n$ (it also includes the encoding size of c).

How do we detect whether the LP is unbounded?

Let $M_{\max} = n2^{2L'}$ be an upper bound on the objective value of a basic feasible solution.

We can add a constraint $c^t x \geq M_{\max} + 1$ and check for feasibility.

How do we detect whether the LP is unbounded?

Let $M_{\max} = n2^{2L'}$ be an upper bound on the objective value of a basic feasible solution.

We can add a constraint $c^t x \geq M_{\max} + 1$ and check for feasibility.

How do we detect whether the LP is unbounded?

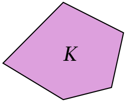
Let $M_{\max} = n2^{2L'}$ be an upper bound on the objective value of a basic feasible solution.

We can add a constraint $c^t x \geq M_{\max} + 1$ and check for feasibility.

Ellipsoid Method

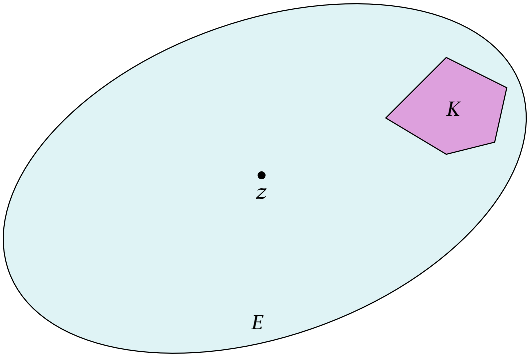
Ellipsoid Method

- Let K be a convex set.



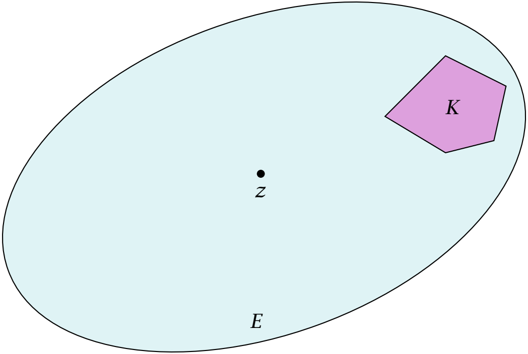
Ellipsoid Method

- Let K be a convex set.
- Maintain ellipsoid E that is guaranteed to contain K provided that K is non-empty.



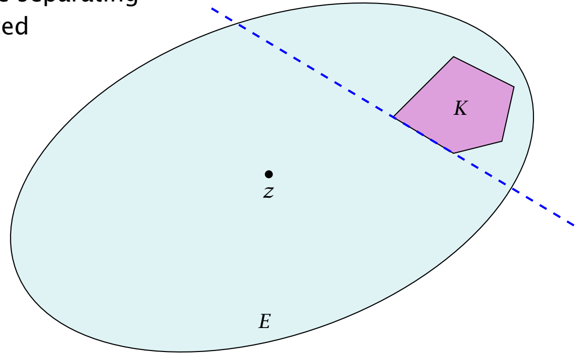
Ellipsoid Method

- Let K be a convex set.
- Maintain ellipsoid E that is guaranteed to contain K provided that K is non-empty.
- If center $z \in K$ STOP.



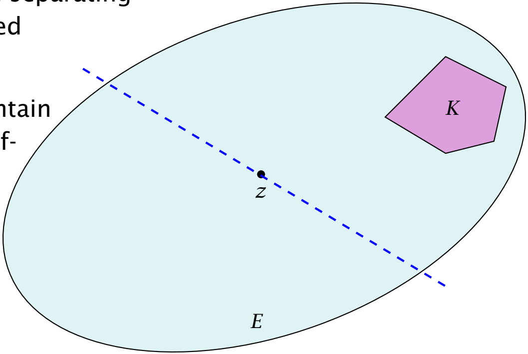
Ellipsoid Method

- Let K be a convex set.
- Maintain ellipsoid E that is guaranteed to contain K provided that K is non-empty.
- If center $z \in K$ STOP.
- Otw. find a hyperplane separating K from z (e.g. a violated constraint in the LP).



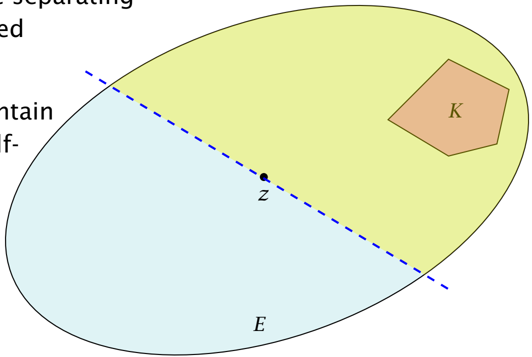
Ellipsoid Method

- Let K be a convex set.
- Maintain ellipsoid E that is guaranteed to contain K provided that K is non-empty.
- If center $z \in K$ STOP.
- Otw. find a hyperplane separating K from z (e.g. a violated constraint in the LP).
- Shift hyperplane to contain node z. H denotes half-space that contains K.



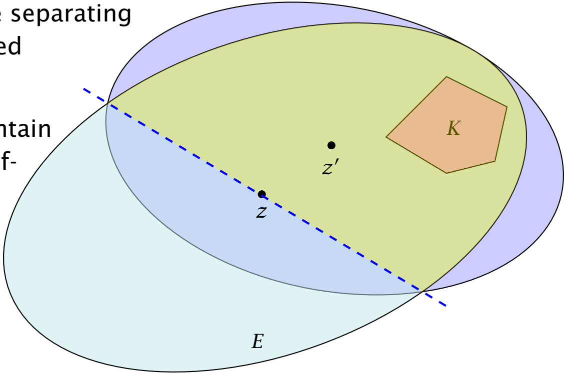
Ellipsoid Method

- Let K be a convex set.
- Maintain ellipsoid E that is guaranteed to contain K provided that K is non-empty.
- If center $z \in K$ STOP.
- Otw. find a hyperplane separating K from z (e.g. a violated constraint in the LP).
- Shift hyperplane to contain node z. H denotes half-space that contains K.



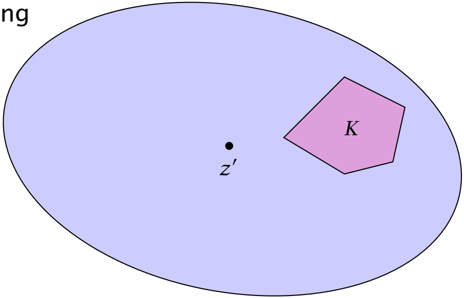
Ellipsoid Method

- Let K be a convex set.
- Maintain ellipsoid E that is guaranteed to contain K provided that K is non-empty.
- If center $z \in K$ STOP.
- Otw. find a hyperplane separating K from z (e.g. a violated constraint in the LP).
- Shift hyperplane to contain node z. H denotes half-space that contains K.
- Compute (smallest) ellipsoid E' that contains $K \cap H$.



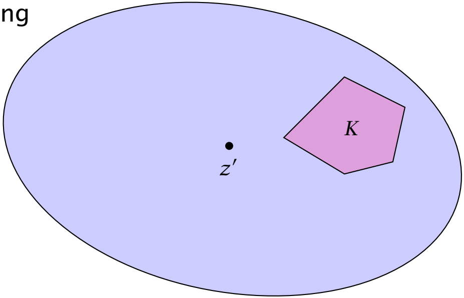
Ellipsoid Method

- Let K be a convex set.
- Maintain ellipsoid E that is guaranteed to contain K provided that K is non-empty.
- If center $z \in K$ STOP.
- Otw. find a hyperplane separating K from z (e.g. a violated constraint in the LP).
- Shift hyperplane to contain node z. H denotes half-space that contains K.
- Compute (smallest) ellipsoid E' that contains $K \cap H$.



Ellipsoid Method

- Let K be a convex set.
- Maintain ellipsoid E that is guaranteed to contain K provided that K is non-empty.
- If center $z \in K$ STOP.
- Otw. find a hyperplane separating K from z (e.g. a violated constraint in the LP).
- Shift hyperplane to contain node z. H denotes half-space that contains K.
- Compute (smallest) ellipsoid E' that contains $K \cap H$.
- REPEAT



Issues/Questions:

- How do you choose the first Ellipsoid? What is its volume?
- What if the polytop K is unbounded?
- How do you measure progress? By how much does the volume decrease in each iteration?
- When can you stop? What is the minimum volume of a non-empty polytop?

Definition 33

A mapping $f : \mathbb{R}^n \to \mathbb{R}^n$ with $f(x) = Lx + t$, where L is an invertible matrix is called an affine transformation.

Definition 34

A ball in $\mathbb{R}^n$ with center c and radius r is given by

$$\begin{aligned} B(c,r) &= \{x \mid (x-c)^t(x-c) \le r^2\} \\ &= \{x \mid \sum_i (x-c)_i^2/r^2 \le 1\} \end{aligned}$$

$B(0,1)$ is called the unit ball.

Definition 35

An affine transformation of the unit ball is called an ellipsoid.

Definition 35

An affine transformation of the unit ball is called an ellipsoid.

From $f(x) = Lx + t$ follows $x = L^{-1}(f(x) - t)$.

Definition 35

An affine transformation of the unit ball is called an ellipsoid.

From $f(x) = Lx + t$ follows $x = L^{-1}(f(x) - t)$.

$$f(B(0,1))$$

Definition 35

An affine transformation of the unit ball is called an ellipsoid.

From $f(x) = Lx + t$ follows $x = L^{-1}(f(x) - t)$.

$$f(B(0,1)) = \{f(x) \mid x \in B(0,1)\}$$

Definition 35

An affine transformation of the unit ball is called an ellipsoid.

From $f(x) = Lx + t$ follows $x = L^{-1}(f(x) - t)$.

$$\begin{aligned} f(B(0,1)) &= \{f(x) \mid x \in B(0,1)\} \\ &= \{y \in \mathbb{R}^n \mid L^{-1}(y - t) \in B(0,1)\} \end{aligned}$$

Definition 35

An affine transformation of the unit ball is called an ellipsoid.

From $f(x) = Lx + t$ follows $x = L^{-1}(f(x) - t)$.

$$\begin{aligned} f(B(0,1)) &= \{f(x) \mid x \in B(0,1)\} \\ &= \{y \in \mathbb{R}^n \mid L^{-1}(y - t) \in B(0,1)\} \\ &= \{y \in \mathbb{R}^n \mid (y - t)^t L^{-1^t} L^{-1}(y - t) \leq 1\} \end{aligned}$$

Definition 35

An affine transformation of the unit ball is called an ellipsoid.

From $f(x) = Lx + t$ follows $x = L^{-1}(f(x) - t)$.

$$\begin{aligned} f(B(0,1)) &= \{f(x) \mid x \in B(0,1)\} \\ &= \{y \in \mathbb{R}^n \mid L^{-1}(y-t) \in B(0,1)\} \\ &= \{y \in \mathbb{R}^n \mid (y-t)^t {L^{-1}}^t L^{-1}(y-t) \le 1\} \\ &= \{y \in \mathbb{R}^n \mid (y-t)^t Q^{-1}(y-t) \le 1\} \end{aligned}$$

Definition 35

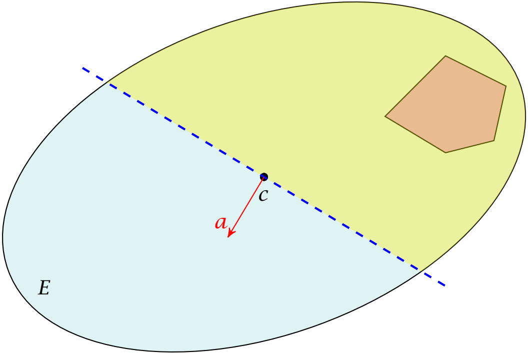
An affine transformation of the unit ball is called an ellipsoid.

From $f(x) = Lx + t$ follows $x = L^{-1}(f(x) - t)$.

$$\begin{aligned} f(B(0,1)) &= \{f(x) \mid x \in B(0,1)\} \\ &= \{y \in \mathbb{R}^n \mid L^{-1}(y-t) \in B(0,1)\} \\ &= \{y \in \mathbb{R}^n \mid (y-t)^t {L^{-1}}^t L^{-1}(y-t) \le 1\} \\ &= \{y \in \mathbb{R}^n \mid (y-t)^t Q^{-1}(y-t) \le 1\} \end{aligned}$$

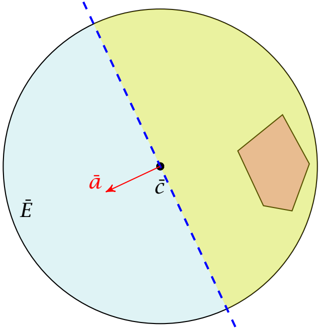
where $Q = LL^t$ is an invertible matrix.

How to Compute the New Ellipsoid



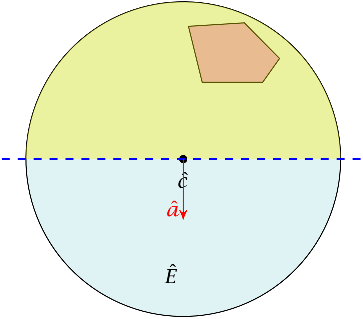
How to Compute the New Ellipsoid

- Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to rotate/distort the ellipsoid (back) into the unit ball.



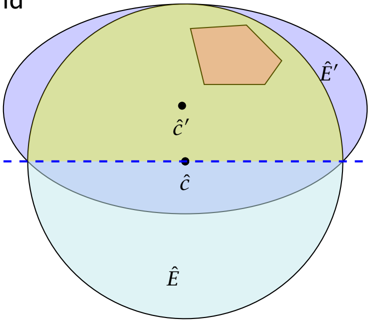
How to Compute the New Ellipsoid

- Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to rotate/distort the ellipsoid (back) into the unit ball.
- Use a rotation R^{-1} to rotate the unit ball such that the normal vector of the halfspace is parallel to e_1.



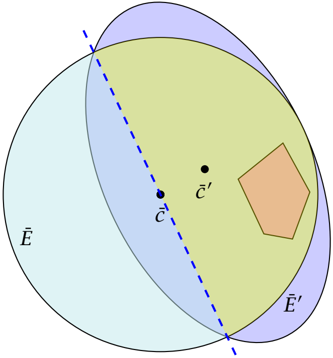
How to Compute the New Ellipsoid

- Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to rotate/distort the ellipsoid (back) into the unit ball.
- Use a rotation R^{-1} to rotate the unit ball such that the normal vector of the halfspace is parallel to e_1.
- Compute the new center $\hat{c}'$ and the new matrix $\hat{Q}'$ for this simplified setting.



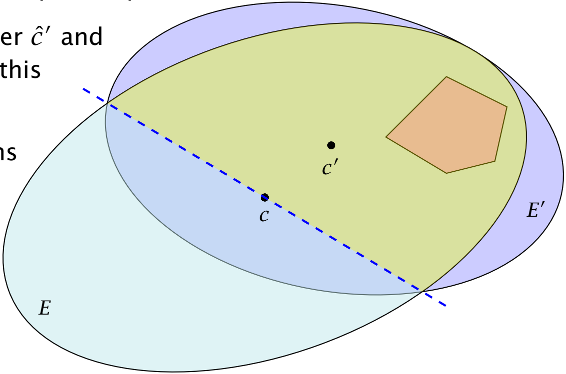
How to Compute the New Ellipsoid

- Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to rotate/distort the ellipsoid (back) into the unit ball.
- Use a rotation R^{-1} to rotate the unit ball such that the normal vector of the halfspace is parallel to e_1.
- Compute the new center $\hat{c}'$ and the new matrix $\hat{Q}'$ for this simplified setting.
- Use the transformations R and f to get the new center c' and the new matrix Q' for the original ellipsoid E.

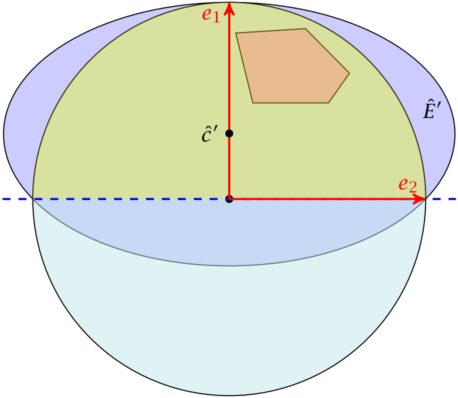


How to Compute the New Ellipsoid

- Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to rotate/distort the ellipsoid (back) into the unit ball.
- Use a rotation R^{-1} to rotate the unit ball such that the normal vector of the halfspace is parallel to e_1.
- Compute the new center $\hat{c}'$ and the new matrix $\hat{Q}'$ for this simplified setting.
- Use the transformations R and f to get the new center c' and the new matrix Q' for the original ellipsoid E.

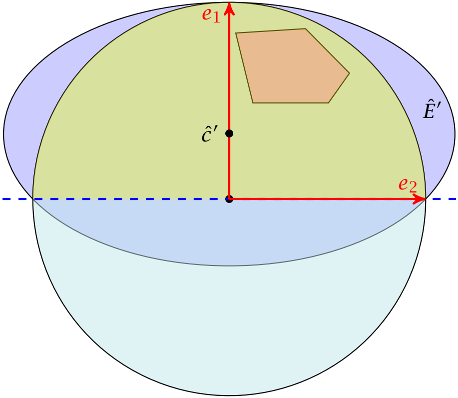


The Easy Case



- The new center lies on axis x_1. Hence, $\hat{c}' = te_1$ for $t > 0$.
- The vectors $e_1, e_2, \ldots$ have to fulfill the ellipsoid constraint with equality. Hence $(e_i - \hat{c}')^t \hat{Q}'^{-1}(e_i - \hat{c}') = 1$.

The Easy Case



- The new center lies on axis x_1. Hence, $\hat{c}' = te_1$ for $t > 0$.
- The vectors $e_1, e_2, \dots$ have to fulfill the ellipsoid constraint with equality. Hence $(e_i - \hat{c}')^t \hat{Q}'^{-1}(e_i - \hat{c}') = 1$.

The Easy Case

- The obtain the matrix $\hat{Q}'^{-1}$ for our ellipsoid $\hat{E}'$ note that $\hat{E}'$ is axis-parallel.
- Let a denote the radius along the x_1-axis and let b denote the (common) radius for the other axes.
- The matrix
$$\hat{L}' = \begin{pmatrix} a & 0 & \dots & 0 \\ 0 & b & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b \end{pmatrix}$$
maps the unit ball (via function $\hat{f}'(x) = \hat{L}'x$) to an axis-parallel ellipsoid with radius a in direction x_1 and b in all other directions.

The Easy Case

- The obtain the matrix $\hat{Q}'^{-1}$ for our ellipsoid $\hat{E}'$ note that $\hat{E}'$ is axis-parallel.
- Let a denote the radius along the x_1-axis and let b denote the (common) radius for the other axes.
- The matrix

$$\hat{L}' = \begin{pmatrix} a & 0 & \dots & 0 \\ 0 & b & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b \end{pmatrix}$$

maps the unit ball (via function $\hat{f}'(x) = \hat{L}'x$) to an axis-parallel ellipsoid with radius a in direction x_1 and b in all other directions.

The Easy Case

- The obtain the matrix $\hat{Q}'^{-1}$ for our ellipsoid $\hat{E}'$ note that $\hat{E}'$ is axis-parallel.
- Let a denote the radius along the x_1-axis and let b denote the (common) radius for the other axes.
- The matrix

$$\hat{L}' = \begin{pmatrix} a & 0 & \dots & 0 \\ 0 & b & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b \end{pmatrix}$$

  maps the unit ball (via function $\hat{f}'(x) = \hat{L}'x$) to an axis-parallel ellipsoid with radius a in direction x_1 and b in all other directions.

The Easy Case

- As $\hat{Q}' = \hat{L}'\hat{L}'^t$ the matrix $\hat{Q}'^{-1}$ is of the form

$$\hat{Q}'^{-1} = \begin{pmatrix} \frac{1}{a^2} & 0 & \dots & 0 \\ 0 & \frac{1}{b^2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{b^2} \end{pmatrix}$$

The Easy Case

- $(e_1 - \hat{c}')^t \hat{Q}'^{-1}(e_1 - \hat{c}') = 1$ gives

$$\begin{pmatrix} 1-t \\ 0 \\ \vdots \\ 0 \end{pmatrix}^t \cdot \begin{pmatrix} \frac{1}{a^2} & 0 & \dots & 0 \\ 0 & \frac{1}{b^2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{b^2} \end{pmatrix} \cdot \begin{pmatrix} 1-t \\ 0 \\ \vdots \\ 0 \end{pmatrix} = 1$$

- This gives $(1-t)^2 = a^2$.

The Easy Case

- For $i \neq 1$ the equation $(e_i - \hat{c}')^t \hat{Q}'^{-1}(e_i - \hat{c}') = 1$ gives

$$\begin{pmatrix} -t \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}^t \cdot \begin{pmatrix} \frac{1}{a^2} & 0 & \dots & 0 \\ 0 & \frac{1}{b^2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{b^2} \end{pmatrix} \cdot \begin{pmatrix} -t \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = 1$$

- This gives $\frac{t^2}{a^2} + \frac{1}{b^2} = 1$, and hence

$$\frac{1}{b^2} = 1 - \frac{t^2}{a^2}$$

The Easy Case

- For $i \neq 1$ the equation $(e_i - \hat{c}')^t \hat{Q}'^{-1}(e_i - \hat{c}') = 1$ gives

$$\begin{pmatrix} -t \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}^t \cdot \begin{pmatrix} \frac{1}{a^2} & 0 & \dots & 0 \\ 0 & \frac{1}{b^2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{b^2} \end{pmatrix} \cdot \begin{pmatrix} -t \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = 1$$

- This gives $\frac{t^2}{a^2} + \frac{1}{b^2} = 1$, and hence

$$\frac{1}{b^2} = 1 - \frac{t^2}{a^2} = 1 - \frac{t^2}{(1-t)^2}$$

The Easy Case

- For $i \neq 1$ the equation $(e_i - \hat{c}')^t \hat{Q}'^{-1}(e_i - \hat{c}') = 1$ gives

$$\begin{pmatrix} -t \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}^t \cdot \begin{pmatrix} \frac{1}{a^2} & 0 & \dots & 0 \\ 0 & \frac{1}{b^2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{b^2} \end{pmatrix} \cdot \begin{pmatrix} -t \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = 1$$

- This gives $\frac{t^2}{a^2} + \frac{1}{b^2} = 1$, and hence

$$\frac{1}{b^2} = 1 - \frac{t^2}{a^2} = 1 - \frac{t^2}{(1-t)^2} = \frac{1-2t}{(1-t)^2}$$

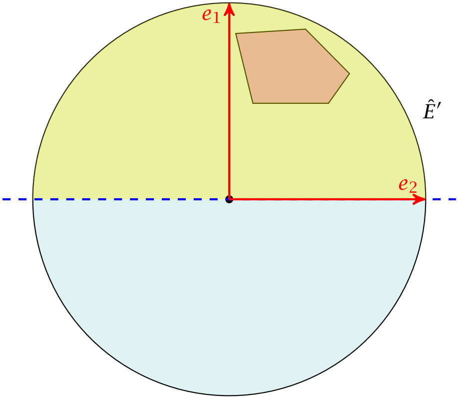
Summary

So far we have

$$a = 1 - t \quad \text{and} \quad b = \frac{1-t}{\sqrt{1-2t}}$$

The Easy Case

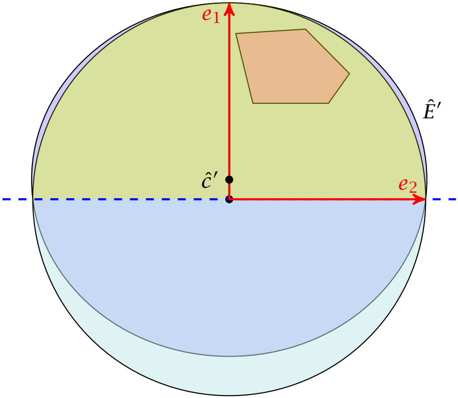
We still have many choices for t:



Choose t such that the volume of $\hat{E}'$ is minimal!!!

The Easy Case

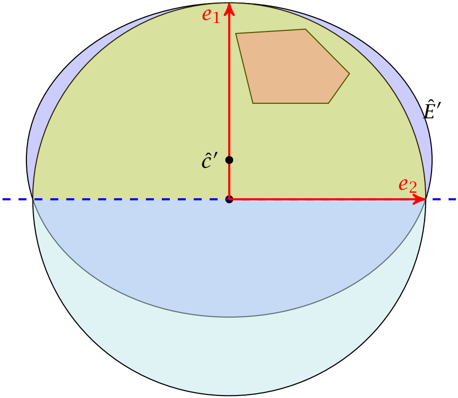
We still have many choices for t:



Choose t such that the volume of $\hat{E}'$ is minimal!!!

The Easy Case

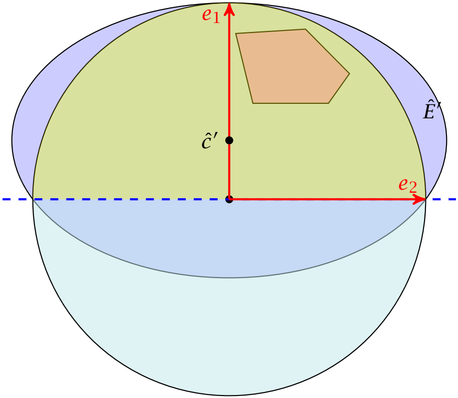
We still have many choices for t:



Choose t such that the volume of $\hat{E}'$ is minimal!!!

The Easy Case

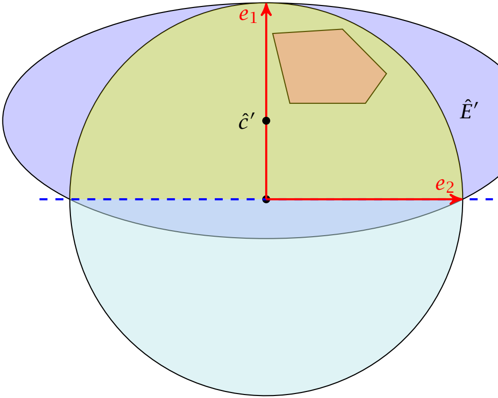
We still have many choices for t:



Choose t such that the volume of $\hat{E}'$ is minimal!!!

The Easy Case

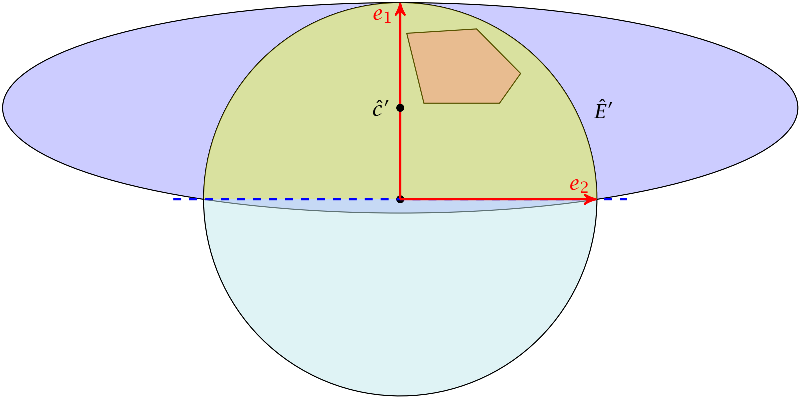
We still have many choices for t:



Choose t such that the volume of $\hat{E}'$ is minimal!!!

The Easy Case

We still have many choices for t:



Choose t such that the volume of $\hat{E}'$ is minimal!!!

The Easy Case

We want to choose t such that the volume of $\hat{E}'$ is minimal.

Lemma 36

Let L be an affine transformation and $K \subseteq \mathbb{R}^n$. Then

$$\text{vol}(L(K)) = |\det(L)| \cdot \text{vol}(K) \ .$$

The Easy Case

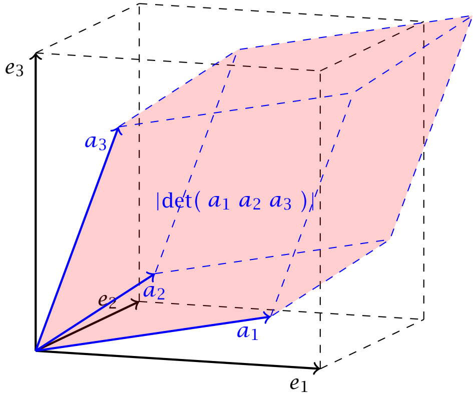
We want to choose t such that the volume of $\hat{E}'$ is minimal.

Lemma 36

Let L be an affine transformation and $K \subseteq \mathbb{R}^n$. Then

$$\mathrm{vol}(L(K)) = |\det(L)| \cdot \mathrm{vol}(K) \ .$$

n-dimensional volume



The Easy Case

- We want to choose t such that the volume of $\hat{E}'$ is minimal.

$$\mathrm{vol}(\hat{E}') = \mathrm{vol}(B(0,1)) \cdot |\det(\hat{L}')| \enspace ,$$

where $\hat{Q}' = \hat{L}'\hat{L}'^t$.

- We have

$$\hat{L}'^{-1} = \begin{pmatrix} \frac{1}{a} & 0 & \dots & 0 \\ 0 & \frac{1}{b} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{b} \end{pmatrix} \text{ and } \hat{L}' = \begin{pmatrix} a & 0 & \dots & 0 \\ 0 & b & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b \end{pmatrix}$$

- Note that a and b in the above equations depend on t, by the previous equations.

The Easy Case

- We want to choose t such that the volume of $\hat{E}'$ is minimal.

$$\mathrm{vol}(\hat{E}') = \mathrm{vol}(B(0,1)) \cdot |\det(\hat{L}')| \enspace ,$$

  where $\hat{Q}' = \hat{L}'\hat{L}'^t$.

- We have

$$\hat{L}'^{-1} = \begin{pmatrix} \frac{1}{a} & 0 & \dots & 0 \\ 0 & \frac{1}{b} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{b} \end{pmatrix} \text{ and } \hat{L}' = \begin{pmatrix} a & 0 & \dots & 0 \\ 0 & b & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b \end{pmatrix}$$

- Note that a and b in the above equations depend on t, by the previous equations.

The Easy Case

- We want to choose t such that the volume of $\hat{E}'$ is minimal.

$$\operatorname{vol}(\hat{E}') = \operatorname{vol}(B(0,1)) \cdot |\det(\hat{L}')| \ ,$$

where $\hat{Q}' = \hat{L}'\hat{L}'^t$.

- We have

$$\hat{L}'^{-1} = \begin{pmatrix} \frac{1}{a} & 0 & \dots & 0 \\ 0 & \frac{1}{b} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{b} \end{pmatrix} \text{ and } \hat{L}' = \begin{pmatrix} a & 0 & \dots & 0 \\ 0 & b & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b \end{pmatrix}$$

- Note that a and b in the above equations depend on t, by the previous equations.

The Easy Case

$$\mathrm{vol}(\hat{E}')$$

The Easy Case

$$\mathrm{vol}(\hat{E}') = \mathrm{vol}(B(0,1)) \cdot |\det(\hat{L}')|$$

The Easy Case

$$\begin{aligned}\mathrm{vol}(\hat{E}') &= \mathrm{vol}(B(0,1)) \cdot |\det(\hat{L}')| \\ &= \mathrm{vol}(B(0,1)) \cdot ab^{n-1}\end{aligned}$$

The Easy Case

$$\begin{aligned}\mathrm{vol}(\hat{E}') &= \mathrm{vol}(B(0,1)) \cdot |\det(\hat{L}')| \\ &= \mathrm{vol}(B(0,1)) \cdot ab^{n-1} \\ &= \mathrm{vol}(B(0,1)) \cdot (1-t) \cdot \left(\frac{1-t}{\sqrt{1-2t}}\right)^{n-1}\end{aligned}$$

The Easy Case

$$\begin{aligned}
\mathrm{vol}(\hat{E}') &= \mathrm{vol}(B(0,1)) \cdot |\det(\hat{L}')| \\
&= \mathrm{vol}(B(0,1)) \cdot ab^{n-1} \\
&= \mathrm{vol}(B(0,1)) \cdot (1-t) \cdot \left(\frac{1-t}{\sqrt{1-2t}}\right)^{n-1} \\
&= \mathrm{vol}(B(0,1)) \cdot \frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}}
\end{aligned}$$

The Easy Case

$$\frac{\mathrm{d}\,\mathrm{vol}(\hat{E}')}{\mathrm{d}\,t}$$

The Easy Case

$$\frac{\mathrm{d}\,\mathrm{vol}(\hat{E}')}{\mathrm{d}\,t} = \frac{\mathrm{d}}{\mathrm{d}\,t}\left(\frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}}\right)$$

The Easy Case

$$\frac{\mathrm{d}\,\mathrm{vol}(\hat{E}')}{\mathrm{d}\,t} = \frac{\mathrm{d}}{\mathrm{d}\,t}\left(\frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}}\right)$$

$$= \frac{1}{N^2}$$

N = denominator

The Easy Case

$$\frac{\mathrm{d}\,\mathrm{vol}(\hat{E}')}{\mathrm{d}\,t} = \frac{\mathrm{d}}{\mathrm{d}\,t}\left(\frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}}\right)$$

$$= \frac{1}{N^2}\cdot\Big((-1)\cdot n(1-t)^{n-1}$$

derivative of numerator

The Easy Case

$$\frac{\mathrm{d}\,\mathrm{vol}(\hat{E}')}{\mathrm{d}\,t} = \frac{\mathrm{d}}{\mathrm{d}\,t}\left(\frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}}\right)$$

$$= \frac{1}{N^2}\cdot\Bigg((-1)\cdot n(1-t)^{n-1}\cdot(\sqrt{1-2t})^{n-1}$$

denominator

The Easy Case

$$\frac{\mathrm{d}\,\mathrm{vol}(\hat{E}')}{\mathrm{d}\,t} = \frac{\mathrm{d}}{\mathrm{d}\,t}\left(\frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}}\right)$$

$$= \frac{1}{N^2}\cdot\Bigg((-1)\cdot n(1-t)^{n-1}\cdot(\sqrt{1-2t})^{n-1}$$

$$-(n-1)(\sqrt{1-2t})^{n-2}$$

outer derivative

The Easy Case

$$\frac{\mathrm{d}\,\mathrm{vol}(\hat{E}')}{\mathrm{d}\,t} = \frac{\mathrm{d}}{\mathrm{d}\,t}\left(\frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}}\right)$$

$$= \frac{1}{N^2}\cdot\Bigg((-1)\cdot n(1-t)^{n-1}\cdot(\sqrt{1-2t})^{n-1}$$

$$-(n-1)(\sqrt{1-2t})^{n-2}\cdot\frac{1}{2\sqrt{1-2t}}\cdot(-2)$$

inner derivative

The Easy Case

$$
\begin{aligned}
\frac{\mathrm{d}\,\mathrm{vol}(\hat{E}')}{\mathrm{d}\,t} &= \frac{\mathrm{d}}{\mathrm{d}\,t}\left(\frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}}\right) \\
&= \frac{1}{N^2}\cdot\Bigg((-1)\cdot n(1-t)^{n-1}\cdot(\sqrt{1-2t})^{n-1} \\
&\quad -(n-1)(\sqrt{1-2t})^{n-2}\cdot\frac{1}{2\sqrt{1-2t}}\cdot(-2)\cdot(1-t)^n\Bigg)
\end{aligned}
$$

numerator

The Easy Case

$$
\begin{aligned}
\frac{\mathrm{d}\,\mathrm{vol}(\hat{E}')}{\mathrm{d}\,t} &= \frac{\mathrm{d}}{\mathrm{d}\,t}\left(\frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}}\right) \\
&= \frac{1}{N^2}\cdot\Bigg((-1)\cdot n(1-t)^{n-1}\cdot(\sqrt{1-2t})^{n-1} \\
&\qquad -(n-1)(\sqrt{1-2t})^{n-2}\cdot\frac{1}{2\sqrt{1-2t}}\cdot(-2)\cdot(1-t)^n\Bigg) \\
&= \frac{1}{N^2}\cdot(\sqrt{1-2t})^{n-3}\cdot(1-t)^{n-1}
\end{aligned}
$$

The Easy Case

$$
\begin{aligned}
\frac{\mathrm{d}\,\mathrm{vol}(\hat{E}')}{\mathrm{d}\,t} &= \frac{\mathrm{d}}{\mathrm{d}\,t}\left(\frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}}\right) \\
&= \frac{1}{N^2}\cdot\Bigg((-1)\cdot n(1-t)^{n-1}\cdot \overbrace{\cancel{(\sqrt{1-2t})^{n-1}}}^{1-2t} \\
&\qquad -(n-1)(\sqrt{1-2t})^{n-2}\cdot\frac{1}{2\sqrt{1-2t}}\cdot(-2)\cdot(1-t)^n\Bigg) \\
&= \frac{1}{N^2}\cdot(\sqrt{1-2t})^{n-3}\cdot(1-t)^{n-1}
\end{aligned}
$$

The Easy Case

$$
\begin{aligned}
\frac{\mathrm{d}\,\mathrm{vol}(\hat{E}')}{\mathrm{d}\,t} &= \frac{\mathrm{d}}{\mathrm{d}\,t}\left(\frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}}\right) \\
&= \frac{1}{N^2}\cdot\Bigg((-1)\cdot n(1-t)^{n-1}\cdot \overbrace{\cancel{(\sqrt{1-2t})^{n-1}}}^{1-2t} \\
&\qquad -(n-1)\cancel{(\sqrt{1-2t})^{n-2}}\cdot\frac{1}{2\cancel{\sqrt{1-2t}}}\cdot(-2)\cdot(1-t)^n\Bigg) \\
&= \frac{1}{N^2}\cdot(\sqrt{1-2t})^{n-3}\cdot(1-t)^{n-1}
\end{aligned}
$$

The Easy Case

$$
\begin{aligned}
\frac{\mathrm{d}\,\mathrm{vol}(\hat{E}')}{\mathrm{d}\,t} &= \frac{\mathrm{d}}{\mathrm{d}\,t}\left(\frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}}\right) \\
&= \frac{1}{N^2}\cdot\Bigg((-1)\cdot n\cancel{(1-t)^{n-1}}\cdot \overset{1-2t}{\cancel{(\sqrt{1-2t})^{n-1}}} \\
&\quad -(n-1)\cancel{(\sqrt{1-2t})^{n-2}}\cdot\frac{1}{2\cancel{\sqrt{1-2t}}}\cdot(-2)\cdot(1-t)^n\Bigg) \\
&= \frac{1}{N^2}\cdot(\sqrt{1-2t})^{n-3}\cdot(1-t)^{n-1}
\end{aligned}
$$

The Easy Case

$$
\begin{aligned}
\frac{\mathrm{d}\,\mathrm{vol}(\hat{E}')}{\mathrm{d}\,t} &= \frac{\mathrm{d}}{\mathrm{d}\,t}\left(\frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}}\right) \\
&= \frac{1}{N^2}\cdot\Bigg((-1)\cdot n\cancel{(1-t)^{n-1}}\cdot \overbrace{\cancel{(\sqrt{1-2t})^{n-1}}}^{1-2t} \\
&\quad -(n-1)\cancel{(\sqrt{1-2t})^{n-2}}\cdot\frac{1}{2\cancel{\sqrt{1-2t}}}\cdot(-2)\cdot \overbrace{\cancel{(1-t)^{n}}}^{1-t}\Bigg) \\
&= \frac{1}{N^2}\cdot(\sqrt{1-2t})^{n-3}\cdot(1-t)^{n-1}
\end{aligned}
$$

The Easy Case

$$\frac{\mathrm{d}\,\mathrm{vol}(\hat{E}')}{\mathrm{d}\,t} = \frac{\mathrm{d}}{\mathrm{d}\,t}\left(\frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}}\right)$$

$$= \frac{1}{N^2}\cdot\Bigg((-1)\cdot n(1-t)^{n-1}\cdot(\sqrt{1-2t})^{n-1}$$

$$-(n-1)(\sqrt{1-2t})^{n-2}\cdot\frac{1}{2\sqrt{1-2t}}\cdot(-2)\cdot(1-t)^n\Bigg)$$

(Handwritten annotations: $(1-t)^{n-1}$ and $(\sqrt{1-2t})^{n-1}$ crossed out with $1-2t$ written above; $(\sqrt{1-2t})^{n-2}$, $\sqrt{1-2t}$ and $(1-t)^n$ crossed out with $1-t$ written above; the minus sign, the 2 and the (-2) crossed out.)

$$= \frac{1}{N^2}\cdot(\sqrt{1-2t})^{n-3}\cdot(1-t)^{n-1}$$

The Easy Case

$$\frac{\mathrm{d}\,\mathrm{vol}(\hat{E}')}{\mathrm{d}\,t} = \frac{\mathrm{d}}{\mathrm{d}\,t}\left(\frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}}\right)$$

$$= \frac{1}{N^2}\cdot\Bigg((-1)\cdot n\cancel{(1-t)^{n-1}}\cdot\overset{1-2t}{\cancel{(\sqrt{1-2t})^{n-1}}}$$

$$\cancel{-}(n-1)\cancel{(\sqrt{1-2t})^{n-2}}\cdot\frac{1}{\cancel{2}\cancel{\sqrt{1-2t}}}\cdot(\cancel{-2})\cdot\overset{1-t}{\cancel{(1-t)^n}}\Bigg)$$

$$= \frac{1}{N^2}\cdot(\sqrt{1-2t})^{n-3}\cdot(1-t)^{n-1}$$

$$\cdot\Big((n-1)(1-t)-n(1-2t)\Big)$$

The Easy Case

$$
\begin{aligned}
\frac{\mathrm{d}\,\mathrm{vol}(\hat{E}')}{\mathrm{d}\,t} &= \frac{\mathrm{d}}{\mathrm{d}\,t}\left(\frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}}\right)\\
&= \frac{1}{N^2}\cdot\Bigg((-1)\cdot n\cancel{(1-t)^{n-1}}\cdot \overset{1-2t}{\cancel{(\sqrt{1-2t})^{n-1}}}\\
&\qquad \cancel{-}(n-1)\cancel{(\sqrt{1-2t})^{n-2}}\cdot\frac{1}{\cancel{2}\cancel{\sqrt{1-2t}}}\cdot(\cancel{-2})\cdot\overset{1-t}{\cancel{(1-t)^n}}\Bigg)\\
&= \frac{1}{N^2}\cdot(\sqrt{1-2t})^{n-3}\cdot(1-t)^{n-1}\\
&\qquad\cdot\Big((n-1)(1-t)-n(1-2t)\Big)\\
&= \frac{1}{N^2}\cdot(\sqrt{1-2t})^{n-3}\cdot(1-t)^{n-1}\cdot\Big((n+1)t-1\Big)
\end{aligned}
$$

The Easy Case

- We obtain the minimum for $t = \frac{1}{n+1}$.
- For this value we obtain

$$a$$

The Easy Case

- We obtain the minimum for $t = \frac{1}{n+1}$.
- For this value we obtain

$$a = 1 - t$$

The Easy Case

- We obtain the minimum for $t = \frac{1}{n+1}$.
- For this value we obtain

$$a = 1 - t = \frac{n}{n+1}$$

The Easy Case

- We obtain the minimum for $t = \frac{1}{n+1}$.
- For this value we obtain

$$a = 1 - t = \frac{n}{n+1} \text{ and } b =$$

The Easy Case

- We obtain the minimum for $t = \frac{1}{n+1}$.
- For this value we obtain

$$a = 1 - t = \frac{n}{n+1} \text{ and } b = \frac{1-t}{\sqrt{1-2t}}$$

The Easy Case

- We obtain the minimum for $t = \frac{1}{n+1}$.
- For this value we obtain

$$a = 1 - t = \frac{n}{n+1} \text{ and } b = \frac{1-t}{\sqrt{1-2t}} = \frac{n}{\sqrt{n^2-1}}$$

The Easy Case

- We obtain the minimum for $t = \frac{1}{n+1}$.
- For this value we obtain

$$a = 1 - t = \frac{n}{n+1} \text{ and } b = \frac{1-t}{\sqrt{1-2t}} = \frac{n}{\sqrt{n^2-1}}$$

To see the equation for b, observe that

$$b^2$$

The Easy Case

- We obtain the minimum for $t = \frac{1}{n+1}$.
- For this value we obtain

$$a = 1 - t = \frac{n}{n+1} \text{ and } b = \frac{1-t}{\sqrt{1-2t}} = \frac{n}{\sqrt{n^2-1}}$$

To see the equation for b, observe that

$$b^2 = \frac{(1-t)^2}{1-2t}$$

The Easy Case

- We obtain the minimum for $t = \frac{1}{n+1}$.
- For this value we obtain

$$a = 1 - t = \frac{n}{n+1} \text{ and } b = \frac{1-t}{\sqrt{1-2t}} = \frac{n}{\sqrt{n^2-1}}$$

To see the equation for b, observe that

$$b^2 = \frac{(1-t)^2}{1-2t} = \frac{(1-\frac{1}{n+1})^2}{1-\frac{2}{n+1}}$$

The Easy Case

- We obtain the minimum for $t = \frac{1}{n+1}$.
- For this value we obtain

$$a = 1 - t = \frac{n}{n+1} \text{ and } b = \frac{1-t}{\sqrt{1-2t}} = \frac{n}{\sqrt{n^2-1}}$$

To see the equation for b, observe that

$$b^2 = \frac{(1-t)^2}{1-2t} = \frac{(1-\frac{1}{n+1})^2}{1-\frac{2}{n+1}} = \frac{(\frac{n}{n+1})^2}{\frac{n-1}{n+1}}$$

The Easy Case

- We obtain the minimum for $t = \frac{1}{n+1}$.
- For this value we obtain

$$a = 1 - t = \frac{n}{n+1} \text{ and } b = \frac{1-t}{\sqrt{1-2t}} = \frac{n}{\sqrt{n^2-1}}$$

To see the equation for b, observe that

$$b^2 = \frac{(1-t)^2}{1-2t} = \frac{(1-\frac{1}{n+1})^2}{1-\frac{2}{n+1}} = \frac{(\frac{n}{n+1})^2}{\frac{n-1}{n+1}} = \frac{n^2}{n^2-1}$$

The Easy Case

Let $\gamma_n = \frac{\mathrm{vol}(\hat{E}')}{\mathrm{vol}(B(0,1))} = ab^{n-1}$ be the ratio by which the volume changes:

$$\gamma_n^2$$

The Easy Case

Let $\gamma_n = \frac{\mathrm{vol}(\hat{E}')}{\mathrm{vol}(B(0,1))} = ab^{n-1}$ be the ratio by which the volume changes:

$$\gamma_n^2 = \left(\frac{n}{n+1}\right)^2 \left(\frac{n^2}{n^2-1}\right)^{n-1}$$

The Easy Case

Let $\gamma_n = \frac{\mathrm{vol}(\hat{E}')}{\mathrm{vol}(B(0,1))} = ab^{n-1}$ be the ratio by which the volume changes:

$$\begin{aligned}\gamma_n^2 &= \Big(\frac{n}{n+1}\Big)^2\Big(\frac{n^2}{n^2-1}\Big)^{n-1} \\ &= \Big(1-\frac{1}{n+1}\Big)^2\Big(1+\frac{1}{(n-1)(n+1)}\Big)^{n-1}\end{aligned}$$

The Easy Case

Let $\gamma_n = \frac{\mathrm{vol}(\hat{E}')}{\mathrm{vol}(B(0,1))} = ab^{n-1}$ be the ratio by which the volume changes:

$$\begin{aligned}\gamma_n^2 &= \Big(\frac{n}{n+1}\Big)^2\Big(\frac{n^2}{n^2-1}\Big)^{n-1}\\ &= \Big(1-\frac{1}{n+1}\Big)^2\Big(1+\frac{1}{(n-1)(n+1)}\Big)^{n-1}\\ &\le e^{-2\frac{1}{n+1}}\cdot e^{\frac{1}{n+1}}\end{aligned}$$

The Easy Case

Let $\gamma_n = \frac{\mathrm{vol}(\hat{E}')}{\mathrm{vol}(B(0,1))} = ab^{n-1}$ be the ratio by which the volume changes:

$$\begin{aligned}
\gamma_n^2 &= \Big(\frac{n}{n+1}\Big)^2 \Big(\frac{n^2}{n^2-1}\Big)^{n-1} \\
&= \Big(1 - \frac{1}{n+1}\Big)^2 \Big(1 + \frac{1}{(n-1)(n+1)}\Big)^{n-1} \\
&\le e^{-2\frac{1}{n+1}} \cdot e^{\frac{1}{n+1}} \\
&= e^{-\frac{1}{n+1}}
\end{aligned}$$

The Easy Case

Let $\gamma_n = \frac{\text{vol}(\hat{E}')}{\text{vol}(B(0,1))} = ab^{n-1}$ be the ratio by which the volume changes:

$$\begin{aligned}
\gamma_n^2 &= \Big(\frac{n}{n+1}\Big)^2 \Big(\frac{n^2}{n^2-1}\Big)^{n-1} \\
&= \Big(1 - \frac{1}{n+1}\Big)^2 \Big(1 + \frac{1}{(n-1)(n+1)}\Big)^{n-1} \\
&\le e^{-2\frac{1}{n+1}} \cdot e^{\frac{1}{n+1}} \\
&= e^{-\frac{1}{n+1}}
\end{aligned}$$

where we used $(1+x)^a \le e^{ax}$ for $x \in \mathbb{R}$ and $a > 0$.

The Easy Case

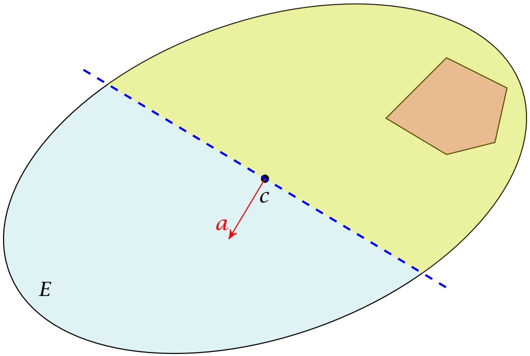
Let $\gamma_n = \frac{\mathrm{vol}(\hat{E}')}{\mathrm{vol}(B(0,1))} = ab^{n-1}$ be the ratio by which the volume changes:

$$
\begin{aligned}
\gamma_n^2 &= \Big(\frac{n}{n+1}\Big)^2\Big(\frac{n^2}{n^2-1}\Big)^{n-1} \\
&= \Big(1-\frac{1}{n+1}\Big)^2\Big(1+\frac{1}{(n-1)(n+1)}\Big)^{n-1} \\
&\le e^{-2\frac{1}{n+1}}\cdot e^{\frac{1}{n+1}} \\
&= e^{-\frac{1}{n+1}}
\end{aligned}
$$

where we used $(1+x)^a \le e^{ax}$ for $x \in \mathbb{R}$ and $a > 0$.

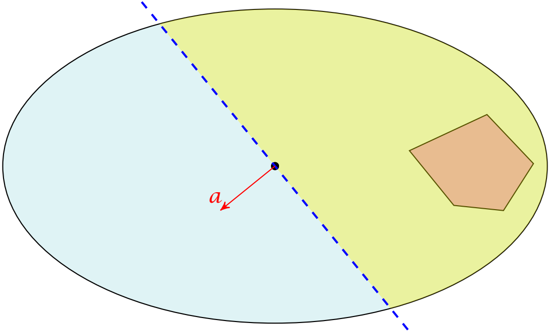
This gives $\gamma_n \le e^{-\frac{1}{2(n+1)}}$.

How to Compute the New Ellipsoid



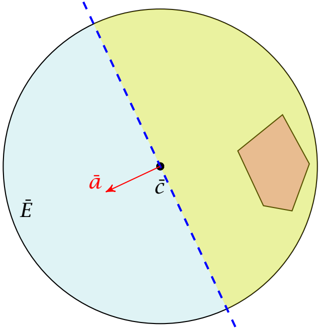
How to Compute the New Ellipsoid

- Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to rotate/distort the ellipsoid (back) into the unit ball.



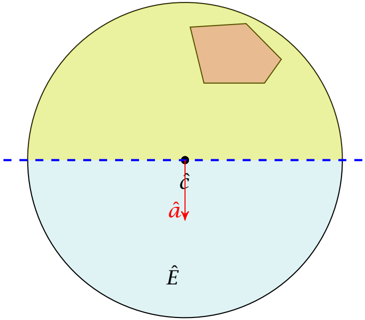
How to Compute the New Ellipsoid

- Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to rotate/distort the ellipsoid (back) into the unit ball.



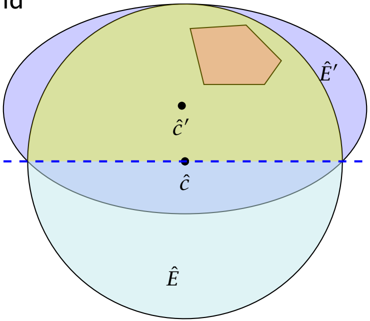
How to Compute the New Ellipsoid

- Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to rotate/distort the ellipsoid (back) into the unit ball.
- Use a rotation R^{-1} to rotate the unit ball such that the normal vector of the halfspace is parallel to e_1.



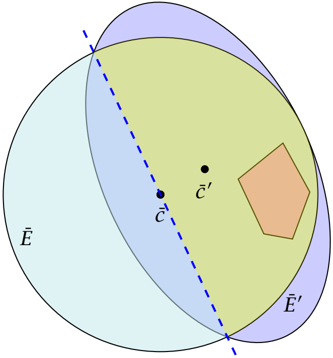
How to Compute the New Ellipsoid

- Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to rotate/distort the ellipsoid (back) into the unit ball.
- Use a rotation R^{-1} to rotate the unit ball such that the normal vector of the halfspace is parallel to e_1.
- Compute the new center $\hat{c}'$ and the new matrix $\hat{Q}'$ for this simplified setting.



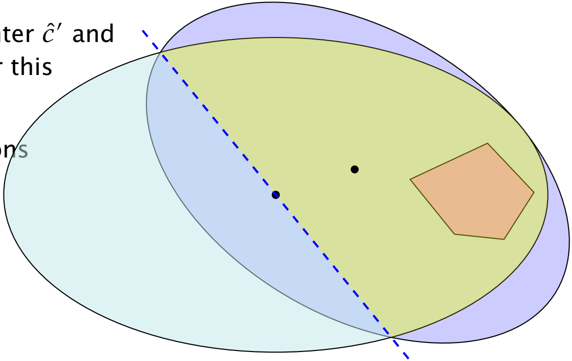
How to Compute the New Ellipsoid

- Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to rotate/distort the ellipsoid (back) into the unit ball.
- Use a rotation R^{-1} to rotate the unit ball such that the normal vector of the halfspace is parallel to e_1.
- Compute the new center $\hat{c}'$ and the new matrix $\hat{Q}'$ for this simplified setting.
- Use the transformations R and f to get the new center c' and the new matrix Q' for the original ellipsoid E.



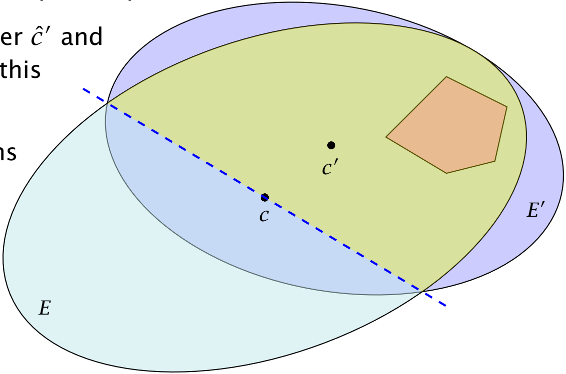
How to Compute the New Ellipsoid

- Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to rotate/distort the ellipsoid (back) into the unit ball.
- Use a rotation R^{-1} to rotate the unit ball such that the normal vector of the halfspace is parallel to e_1.
- Compute the new center $\hat{c}'$ and the new matrix $\hat{Q}'$ for this simplified setting.
- Use the transformations R and f to get the new center c' and the new matrix Q' for the original ellipsoid E.



How to Compute the New Ellipsoid

- Use f^{-1} (recall that $f = Lx + t$ is the affine transformation of the unit ball) to rotate/distort the ellipsoid (back) into the unit ball.
- Use a rotation R^{-1} to rotate the unit ball such that the normal vector of the halfspace is parallel to e_1.
- Compute the new center $\hat{c}'$ and the new matrix $\hat{Q}'$ for this simplified setting.
- Use the transformations R and f to get the new center c' and the new matrix Q' for the original ellipsoid E.



Our progress is the same:

$$e^{-\frac{1}{2(n+1)}}$$

Our progress is the same:

$$e^{-\frac{1}{2(n+1)}} \geq \frac{\mathrm{vol}(\hat{E}')}{\mathrm{vol}(B(0,1))}$$

Our progress is the same:

$$e^{-\frac{1}{2(n+1)}} \geq \frac{\mathrm{vol}(\hat{E}')}{\mathrm{vol}(B(0,1))} = \frac{\mathrm{vol}(\hat{E}')}{\mathrm{vol}(\hat{E})}$$

Our progress is the same:

$$e^{-\frac{1}{2(n+1)}} \geq \frac{\mathrm{vol}(\hat{E}')}{\mathrm{vol}(B(0,1))} = \frac{\mathrm{vol}(\hat{E}')}{\mathrm{vol}(\hat{E})} = \frac{\mathrm{vol}(R(\hat{E}'))}{\mathrm{vol}(R(\hat{E}))}$$

Our progress is the same:

$$e^{-\frac{1}{2(n+1)}} \geq \frac{\mathrm{vol}(\hat{E}')}{\mathrm{vol}(B(0,1))} = \frac{\mathrm{vol}(\hat{E}')}{\mathrm{vol}(\hat{E})} = \frac{\mathrm{vol}(R(\hat{E}'))}{\mathrm{vol}(R(\hat{E}))}$$

$$= \frac{\mathrm{vol}(\bar{E}')}{\mathrm{vol}(\bar{E})}$$

Our progress is the same:

$$e^{-\frac{1}{2(n+1)}} \geq \frac{\mathrm{vol}(\hat{E}')}{\mathrm{vol}(B(0,1))} = \frac{\mathrm{vol}(\hat{E}')}{\mathrm{vol}(\hat{E})} = \frac{\mathrm{vol}(R(\hat{E}'))}{\mathrm{vol}(R(\hat{E}))}$$

$$= \frac{\mathrm{vol}(\bar{E}')}{\mathrm{vol}(\bar{E})} = \frac{\mathrm{vol}(f(\bar{E}'))}{\mathrm{vol}(f(\bar{E}))}$$

Our progress is the same:

$$
\begin{aligned}
e^{-\frac{1}{2(n+1)}} \geq \frac{\mathrm{vol}(\hat{E}')}{\mathrm{vol}(B(0,1))} = \frac{\mathrm{vol}(\hat{E}')}{\mathrm{vol}(\hat{E})} = \frac{\mathrm{vol}(R(\hat{E}'))}{\mathrm{vol}(R(\hat{E}))} \\
= \frac{\mathrm{vol}(\bar{E}')}{\mathrm{vol}(\bar{E})} = \frac{\mathrm{vol}(f(\bar{E}'))}{\mathrm{vol}(f(\bar{E}))} = \frac{\mathrm{vol}(E')}{\mathrm{vol}(E)}
\end{aligned}
$$

Our progress is the same:

$$e^{-\frac{1}{2(n+1)}} \geq \frac{\mathrm{vol}(\hat{E}')}{\mathrm{vol}(B(0,1))} = \frac{\mathrm{vol}(\hat{E}')}{\mathrm{vol}(\hat{E})} = \frac{\mathrm{vol}(R(\hat{E}'))}{\mathrm{vol}(R(\hat{E}))}$$
$$= \frac{\mathrm{vol}(\bar{E}')}{\mathrm{vol}(\bar{E})} = \frac{\mathrm{vol}(f(\bar{E}'))}{\mathrm{vol}(f(\bar{E}))} = \frac{\mathrm{vol}(E')}{\mathrm{vol}(E)}$$

Here it is important that mapping a set with affine function $f(x) = Lx + t$ changes the volume by factor $\det(L)$.

The Ellipsoid Algorithm

How to Compute The New Parameters?

The Ellipsoid Algorithm

How to Compute The New Parameters?

The transformation function of the (old) ellipsoid: $f(x) = Lx + c$;

The Ellipsoid Algorithm

How to Compute The New Parameters?

The transformation function of the (old) ellipsoid: $f(x) = Lx + c$;

The halfspace to be intersected: $H = \{x \mid a^t(x - c) \le 0\}$;

The Ellipsoid Algorithm

How to Compute The New Parameters?

The transformation function of the (old) ellipsoid: $f(x) = Lx + c$;

The halfspace to be intersected: $H = \{x \mid a^t(x - c) \leq 0\}$;

$$f^{-1}(H) = \{f^{-1}(x) \mid a^t(x - c) \leq 0\}$$

The Ellipsoid Algorithm

How to Compute The New Parameters?

The transformation function of the (old) ellipsoid: $f(x) = Lx + c$;

The halfspace to be intersected: $H = \{x \mid a^t(x - c) \leq 0\}$;

$$\begin{aligned} f^{-1}(H) &= \{f^{-1}(x) \mid a^t(x-c) \leq 0\} \\ &= \{f^{-1}(f(y)) \mid a^t(f(y)-c) \leq 0\} \end{aligned}$$

The Ellipsoid Algorithm

How to Compute The New Parameters?

The transformation function of the (old) ellipsoid: $f(x) = Lx + c$;

The halfspace to be intersected: $H = \{x \mid a^t(x - c) \le 0\}$;

$$\begin{aligned} f^{-1}(H) &= \{f^{-1}(x) \mid a^t(x - c) \le 0\} \\ &= \{f^{-1}(f(y)) \mid a^t(f(y) - c) \le 0\} \\ &= \{y \mid a^t(f(y) - c) \le 0\} \end{aligned}$$

How to Compute The New Parameters?

The transformation function of the (old) ellipsoid: $f(x) = Lx + c$;

The halfspace to be intersected: $H = \{x \mid a^t(x-c) \leq 0\}$;

$$\begin{aligned} f^{-1}(H) &= \{f^{-1}(x) \mid a^t(x-c) \leq 0\} \\ &= \{f^{-1}(f(y)) \mid a^t(f(y)-c) \leq 0\} \\ &= \{y \mid a^t(f(y)-c) \leq 0\} \\ &= \{y \mid a^t(Ly+c-c) \leq 0\} \end{aligned}$$

The Ellipsoid Algorithm

How to Compute The New Parameters?

The transformation function of the (old) ellipsoid: $f(x) = Lx + c$;

The halfspace to be intersected: $H = \{x \mid a^t(x - c) \le 0\}$;

$$\begin{aligned} f^{-1}(H) &= \{f^{-1}(x) \mid a^t(x-c) \le 0\} \\ &= \{f^{-1}(f(y)) \mid a^t(f(y)-c) \le 0\} \\ &= \{y \mid a^t(f(y)-c) \le 0\} \\ &= \{y \mid a^t(Ly+c-c) \le 0\} \\ &= \{y \mid (a^tL)y \le 0\} \end{aligned}$$

The Ellipsoid Algorithm

How to Compute The New Parameters?

The transformation function of the (old) ellipsoid: $f(x) = Lx + c$;

The halfspace to be intersected: $H = \{x \mid a^t(x - c) \le 0\}$;

$$\begin{aligned} f^{-1}(H) &= \{f^{-1}(x) \mid a^t(x-c) \le 0\} \\ &= \{f^{-1}(f(y)) \mid a^t(f(y)-c) \le 0\} \\ &= \{y \mid a^t(f(y)-c) \le 0\} \\ &= \{y \mid a^t(Ly+c-c) \le 0\} \\ &= \{y \mid (a^tL)y \le 0\} \end{aligned}$$

This means $\bar{a} = L^t a$.

The Ellipsoid Algorithm

After rotating back (applying R^{-1}) the normal vector of the halfspace points in negative x_1-direction. Hence,

$$R^{-1}\Big(\frac{L^t a}{\|L^t a\|}\Big) = -e_1 \quad \Rightarrow \quad -\frac{L^t a}{\|L^t a\|} = R \cdot e_1$$

The Ellipsoid Algorithm

After rotating back (applying R^{-1}) the normal vector of the halfspace points in negative x_1-direction. Hence,

$$R^{-1}\Big(\frac{L^t a}{\|L^t a\|}\Big) = -e_1 \quad \Rightarrow \quad -\frac{L^t a}{\|L^t a\|} = R \cdot e_1$$

Hence,

$$\bar{c}'$$

The Ellipsoid Algorithm

After rotating back (applying R^{-1}) the normal vector of the halfspace points in negative x_1-direction. Hence,

$$R^{-1}\Big(\frac{L^t a}{\|L^t a\|}\Big) = -e_1 \quad \Rightarrow \quad -\frac{L^t a}{\|L^t a\|} = R \cdot e_1$$

Hence,

$$\bar{c}' = R \cdot \hat{c}'$$

The Ellipsoid Algorithm

After rotating back (applying R^{-1}) the normal vector of the halfspace points in negative x_1-direction. Hence,

$$R^{-1}\Big(\frac{L^t a}{\|L^t a\|}\Big) = -e_1 \quad \Rightarrow \quad -\frac{L^t a}{\|L^t a\|} = R \cdot e_1$$

Hence,

$$\bar{c}' = R \cdot \hat{c}' = R \cdot \frac{1}{n+1} e_1$$

The Ellipsoid Algorithm

After rotating back (applying R^{-1}) the normal vector of the halfspace points in negative x_1-direction. Hence,

$$R^{-1}\Big(\frac{L^t a}{\|L^t a\|}\Big) = -e_1 \quad \Rightarrow \quad -\frac{L^t a}{\|L^t a\|} = R \cdot e_1$$

Hence,

$$\bar{c}' = R \cdot \hat{c}' = R \cdot \frac{1}{n+1} e_1 = -\frac{1}{n+1} \frac{L^t a}{\|L^t a\|}$$

The Ellipsoid Algorithm

After rotating back (applying R^{-1}) the normal vector of the halfspace points in negative x_1-direction. Hence,

$$R^{-1}\Big(\frac{L^t a}{\|L^t a\|}\Big) = -e_1 \quad \Rightarrow \quad -\frac{L^t a}{\|L^t a\|} = R \cdot e_1$$

Hence,

$$\bar{c}' = R \cdot \hat{c}' = R \cdot \frac{1}{n+1} e_1 = -\frac{1}{n+1} \frac{L^t a}{\|L^t a\|}$$

c'

The Ellipsoid Algorithm

After rotating back (applying R^{-1}) the normal vector of the halfspace points in negative x_1-direction. Hence,

$$R^{-1}\Big(\frac{L^t a}{\|L^t a\|}\Big) = -e_1 \quad \Rightarrow \quad -\frac{L^t a}{\|L^t a\|} = R \cdot e_1$$

Hence,

$$\bar{c}' = R \cdot \hat{c}' = R \cdot \frac{1}{n+1} e_1 = -\frac{1}{n+1} \frac{L^t a}{\|L^t a\|}$$

$$c' = f(\bar{c}')$$

The Ellipsoid Algorithm

After rotating back (applying R^{-1}) the normal vector of the halfspace points in negative x_1-direction. Hence,

$$R^{-1}\Big(\frac{L^t a}{\|L^t a\|}\Big) = -e_1 \quad \Rightarrow \quad -\frac{L^t a}{\|L^t a\|} = R \cdot e_1$$

Hence,

$$\bar{c}' = R \cdot \hat{c}' = R \cdot \frac{1}{n+1} e_1 = -\frac{1}{n+1} \frac{L^t a}{\|L^t a\|}$$

$$c' = f(\bar{c}') = L \cdot \bar{c}' + c$$

The Ellipsoid Algorithm

After rotating back (applying R^{-1}) the normal vector of the halfspace points in negative x_1-direction. Hence,

$$R^{-1}\Big(\frac{L^t a}{\|L^t a\|}\Big) = -e_1 \quad \Rightarrow \quad -\frac{L^t a}{\|L^t a\|} = R \cdot e_1$$

Hence,

$$\bar{c}' = R \cdot \hat{c}' = R \cdot \frac{1}{n+1} e_1 = -\frac{1}{n+1} \frac{L^t a}{\|L^t a\|}$$

$$\begin{aligned} c' &= f(\bar{c}') = L \cdot \bar{c}' + c \\ &= -\frac{1}{n+1} L \frac{L^t a}{\|L^t a\|} + c \end{aligned}$$

The Ellipsoid Algorithm

After rotating back (applying R^{-1}) the normal vector of the halfspace points in negative x_1-direction. Hence,

$$R^{-1}\Big(\frac{L^t a}{\|L^t a\|}\Big) = -e_1 \quad \Rightarrow \quad -\frac{L^t a}{\|L^t a\|} = R \cdot e_1$$

Hence,

$$\bar{c}' = R \cdot \hat{c}' = R \cdot \frac{1}{n+1} e_1 = -\frac{1}{n+1}\frac{L^t a}{\|L^t a\|}$$

$$\begin{aligned} c' &= f(\bar{c}') = L \cdot \bar{c}' + c \\ &= -\frac{1}{n+1} L \frac{L^t a}{\|L^t a\|} + c \\ &= c - \frac{1}{n+1}\frac{Qa}{\sqrt{a^t Q a}} \end{aligned}$$

For computing the matrix Q' of the new ellipsoid we assume in the following that $\hat{E}', \bar{E}'$ and E' refer to the ellispoids centered in the origin.

Recall that

$$\hat{Q}' = \begin{pmatrix} a^2 & 0 & \dots & 0 \\ 0 & b^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b^2 \end{pmatrix}$$

This gives

$$\hat{Q}' = \frac{n^2}{n^2-1}\Big(I - \frac{2}{n+1}e_1 e_1^t\Big)$$

because for $a = n/n+1$ and $b = n/\sqrt{n^2-1}$

$$b^2 - b^2\frac{2}{n+1} = \frac{n^2}{n^2-1} - \frac{2n^2}{(n-1)(n+1)^2}$$

$$= \frac{n^2(n+1) - 2n^2}{(n-1)(n+1)^2} = \frac{n^2(n-1)}{(n-1)(n+1)^2} = a^2$$

Recall that

$$\hat{Q}' = \begin{pmatrix} a^2 & 0 & \dots & 0 \\ 0 & b^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b^2 \end{pmatrix}$$

This gives

$$\hat{Q}' = \frac{n^2}{n^2-1}\Big(I - \frac{2}{n+1}e_1e_1^t\Big)$$

because for $a = n/n+1$ and $b = n/\sqrt{n^2-1}$

$$b^2 - b^2\frac{2}{n+1} = \frac{n^2}{n^2-1} - \frac{2n^2}{(n-1)(n+1)^2}$$

$$= \frac{n^2(n+1) - 2n^2}{(n-1)(n+1)^2} = \frac{n^2(n-1)}{(n-1)(n+1)^2} = a^2$$

Recall that

$$\hat{Q}' = \begin{pmatrix} a^2 & 0 & \dots & 0 \\ 0 & b^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b^2 \end{pmatrix}$$

This gives

$$\hat{Q}' = \frac{n^2}{n^2-1}\Big(I - \frac{2}{n+1} e_1 e_1^t\Big)$$

because for $a = n/n+1$ and $b = n/\sqrt{n^2-1}$

Recall that

$$\hat{Q}' = \begin{pmatrix} a^2 & 0 & \dots & 0 \\ 0 & b^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b^2 \end{pmatrix}$$

This gives

$$\hat{Q}' = \frac{n^2}{n^2-1}\Big(I - \frac{2}{n+1} e_1 e_1^t\Big)$$

because for $a = n/_{n+1}$ and $b = n/_{\sqrt{n^2-1}}$

$$b^2 - b^2 \frac{2}{n+1} = \frac{n^2}{n^2-1} - \frac{2n^2}{(n-1)(n+1)^2}$$

$$= \frac{n^2(n+1) - 2n^2}{(n-1)(n+1)^2} = \frac{n^2(n-1)}{(n-1)(n+1)^2} = a^2$$

Recall that

$$\hat{Q}' = \begin{pmatrix} a^2 & 0 & \dots & 0 \\ 0 & b^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b^2 \end{pmatrix}$$

This gives

$$\hat{Q}' = \frac{n^2}{n^2-1}\Big(I - \frac{2}{n+1} e_1 e_1^t\Big)$$

because for $a = {}^n/_{n+1}$ and $b = {}^n/_{\sqrt{n^2-1}}$

$$b^2 - b^2 \frac{2}{n+1} = \frac{n^2}{n^2-1} - \frac{2n^2}{(n-1)(n+1)^2}$$

$$= \frac{n^2(n+1) - 2n^2}{(n-1)(n+1)^2} = \frac{n^2(n-1)}{(n-1)(n+1)^2} = a^2$$

Recall that

$$\hat{Q}' = \begin{pmatrix} a^2 & 0 & \dots & 0 \\ 0 & b^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b^2 \end{pmatrix}$$

This gives

$$\hat{Q}' = \frac{n^2}{n^2 - 1}\Big(I - \frac{2}{n+1} e_1 e_1^t\Big)$$

because for $a = n/n+1$ and $b = n/\sqrt{n^2-1}$

$$\begin{aligned} b^2 - b^2 \frac{2}{n+1} &= \frac{n^2}{n^2-1} - \frac{2n^2}{(n-1)(n+1)^2} \\ &= \frac{n^2(n+1) - 2n^2}{(n-1)(n+1)^2} = \frac{n^2(n-1)}{(n-1)(n+1)^2} = a^2 \end{aligned}$$

Recall that

$$\hat{Q}' = \begin{pmatrix} a^2 & 0 & \dots & 0 \\ 0 & b^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b^2 \end{pmatrix}$$

This gives

$$\hat{Q}' = \frac{n^2}{n^2-1}\Big(I - \frac{2}{n+1} e_1 e_1^t\Big)$$

because for $a = {}^{n}/_{n+1}$ and $b = {}^{n}/_{\sqrt{n^2-1}}$

$$\begin{aligned} b^2 - b^2 \frac{2}{n+1} &= \frac{n^2}{n^2-1} - \frac{2n^2}{(n-1)(n+1)^2} \\ &= \frac{n^2(n+1) - 2n^2}{(n-1)(n+1)^2} = \frac{n^2(n-1)}{(n-1)(n+1)^2} = a^2 \end{aligned}$$

Recall that

$$\hat{Q}' = \begin{pmatrix} a^2 & 0 & \dots & 0 \\ 0 & b^2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b^2 \end{pmatrix}$$

This gives

$$\hat{Q}' = \frac{n^2}{n^2-1}\Big(I - \frac{2}{n+1}e_1 e_1^t\Big)$$

because for $a = {}^{n}/_{n+1}$ and $b = {}^{n}/_{\sqrt{n^2-1}}$

$$\begin{aligned} b^2 - b^2\frac{2}{n+1} &= \frac{n^2}{n^2-1} - \frac{2n^2}{(n-1)(n+1)^2} \\ &= \frac{n^2(n+1) - 2n^2}{(n-1)(n+1)^2} = \frac{n^2(n-1)}{(n-1)(n+1)^2} = a^2 \end{aligned}$$

9 The Ellipsoid Algorithm

$\bar{E}'$

9 The Ellipsoid Algorithm

$$\bar{E}' = R(\hat{E}')$$

9 The Ellipsoid Algorithm

$$
\begin{aligned}
\bar{E}' &= R(\hat{E}') \\
&= \{R(x) \mid x^t {\hat{Q}'}^{-1} x \le 1\}
\end{aligned}
$$

9 The Ellipsoid Algorithm

$$\begin{aligned}\bar{E}' &= R(\hat{E}') \\ &= \{R(x) \mid x^t {\hat{Q}'}^{-1} x \le 1\} \\ &= \{y \mid (R^{-1}y)^t {\hat{Q}'}^{-1} R^{-1} y \le 1\}\end{aligned}$$

9 The Ellipsoid Algorithm

$$
\begin{aligned}
\bar{E}' &= R(\hat{E}') \\
&= \{R(x) \mid x^t \hat{Q}'^{-1} x \le 1\} \\
&= \{y \mid (R^{-1}y)^t \hat{Q}'^{-1} R^{-1} y \le 1\} \\
&= \{y \mid y^t (R^t)^{-1} \hat{Q}'^{-1} R^{-1} y \le 1\}
\end{aligned}
$$

9 The Ellipsoid Algorithm

$$
\begin{aligned}
\bar{E}' &= R(\hat{E}') \\
&= \{R(x) \mid x^t \hat{Q}'^{-1} x \le 1\} \\
&= \{y \mid (R^{-1}y)^t \hat{Q}'^{-1} R^{-1} y \le 1\} \\
&= \{y \mid y^t (R^t)^{-1} \hat{Q}'^{-1} R^{-1} y \le 1\} \\
&= \{y \mid y^t (\underbrace{R\hat{Q}'R^t}_{\bar{Q}'})^{-1} y \le 1\}
\end{aligned}
$$

9 The Ellipsoid Algorithm

Hence,

$$\bar{Q}'$$

9 The Ellipsoid Algorithm

Hence,

$$\bar{Q}' = R\hat{Q}'R^t$$

9 The Ellipsoid Algorithm

Hence,

$$\begin{aligned}\bar{Q}' &= R\hat{Q}'R^t \\ &= R \cdot \frac{n^2}{n^2-1}\Big(I - \frac{2}{n+1}e_1e_1^t\Big) \cdot R^t\end{aligned}$$

9 The Ellipsoid Algorithm

Hence,

$$
\begin{aligned}
\bar{Q}' &= R\hat{Q}'R^t \\
&= R \cdot \frac{n^2}{n^2-1}\Big(I - \frac{2}{n+1}e_1e_1^t\Big) \cdot R^t \\
&= \frac{n^2}{n^2-1}\Big(R \cdot R^t - \frac{2}{n+1}(Re_1)(Re_1)^t\Big)
\end{aligned}
$$

9 The Ellipsoid Algorithm

Hence,

$$\begin{aligned}\bar{Q}' &= R\hat{Q}'R^t \\ &= R \cdot \frac{n^2}{n^2-1}\Big(I - \frac{2}{n+1}e_1e_1^t\Big) \cdot R^t \\ &= \frac{n^2}{n^2-1}\Big(R \cdot R^t - \frac{2}{n+1}(Re_1)(Re_1)^t\Big) \\ &= \frac{n^2}{n^2-1}\Big(I - \frac{2}{n+1}\frac{L^taa^tL}{\|L^ta\|^2}\Big)\end{aligned}$$

9 The Ellipsoid Algorithm

E'

9 The Ellipsoid Algorithm

$$E' = L(\bar{E}')$$

9 The Ellipsoid Algorithm

$$
\begin{aligned}
E' &= L(\bar{E}') \\
&= \{L(x) \mid x^t \bar{Q}'^{-1} x \le 1\}
\end{aligned}
$$

9 The Ellipsoid Algorithm

$$
\begin{aligned}
E' &= L(\bar{E}') \\
&= \{L(x) \mid x^t \bar{Q}'^{-1} x \le 1\} \\
&= \{y \mid (L^{-1}y)^t \bar{Q}'^{-1} L^{-1} y \le 1\}
\end{aligned}
$$

9 The Ellipsoid Algorithm

$$
\begin{aligned}
E' &= L(\bar{E}') \\
&= \{L(x) \mid x^t \bar{Q}'^{-1} x \le 1\} \\
&= \{y \mid (L^{-1}y)^t \bar{Q}'^{-1} L^{-1} y \le 1\} \\
&= \{y \mid y^t (L^t)^{-1} \bar{Q}'^{-1} L^{-1} y \le 1\}
\end{aligned}
$$

9 The Ellipsoid Algorithm

$$
\begin{aligned}
E' &= L(\bar{E}') \\
&= \{L(x) \mid x^t \bar{Q}'^{-1} x \le 1\} \\
&= \{y \mid (L^{-1}y)^t \bar{Q}'^{-1} L^{-1} y \le 1\} \\
&= \{y \mid y^t (L^t)^{-1} \bar{Q}'^{-1} L^{-1} y \le 1\} \\
&= \{y \mid y^t (\underbrace{L\bar{Q}'L^t}_{Q'})^{-1} y \le 1\}
\end{aligned}
$$

9 The Ellipsoid Algorithm

Hence,

$$Q'$$

9 The Ellipsoid Algorithm

Hence,

$$Q' = L\bar{Q}'L^t$$

9 The Ellipsoid Algorithm

Hence,

$$\begin{aligned} Q' &= L\bar{Q}'L^t \\ &= L \cdot \frac{n^2}{n^2-1}\Big(I - \frac{2}{n+1}\frac{L^t a a^t L}{a^t Q a}\Big) \cdot L^t \end{aligned}$$

9 The Ellipsoid Algorithm

Hence,

$$
\begin{aligned}
Q' &= L\bar{Q}'L^t \\
&= L \cdot \frac{n^2}{n^2-1}\Big(I - \frac{2}{n+1}\frac{L^t a a^t L}{a^t Q a}\Big) \cdot L^t \\
&= \frac{n^2}{n^2-1}\Big(Q - \frac{2}{n+1}\frac{Q a a^t Q}{a^t Q a}\Big)
\end{aligned}
$$

Incomplete Algorithm

Algorithm 1 ellipsoid-algorithm

input: point $c \in \mathbb{R}^n$, convex set $K \subseteq \mathbb{R}^n$

output: point $x \in K$ or "K is empty"

$Q \leftarrow$???

repeat

  if $c \in K$ **then return** c

  else

    choose a violated hyperplane a

$$c \leftarrow c - \frac{1}{n+1}\frac{Qa}{\sqrt{a^tQa}}$$

$$Q \leftarrow \frac{n^2}{n^2-1}\Big(Q - \frac{2}{n+1}\frac{Qaa^tQ}{a^tQa}\Big)$$

  endif

until ???

return "K is empty"

Repeat: Size of basic solutions

Lemma 37

Let $P = \{x \in \mathbb{R}^n \mid Ax \le b\}$ be a bounded polytop. Let $\langle a_{\max} \rangle$ be the maximum encoding length of an entry in A, b. Then every entry x_j in a basic solution fulfills $|x_j| = \frac{D_j}{D}$ with $D_j, D \le 2^{2n\langle a_{\max}\rangle + 2n\log_2 n}$.

In the following we use $\delta := 2^{2n\langle a_{\max}\rangle + 2n\log_2 n}$.

Note that here we have $P = \{x \mid Ax \le b\}$. The previous lemmas we had about the size of feasible solutions were slightly different as they were for different polytopes.

Repeat: Size of basic solutions

Lemma 37

Let $P = \{x \in \mathbb{R}^n \mid Ax \le b\}$ be a bounded polytop. Let $\langle a_{\max}\rangle$ be the maximum encoding length of an entry in A, b. Then every entry x_j in a basic solution fulfills $|x_j| = \frac{D_j}{D}$ with $D_j, D \le 2^{2n\langle a_{\max}\rangle + 2n\log_2 n}$.

In the following we use $\delta := 2^{2n\langle a_{\max}\rangle + 2n\log_2 n}$.

Note that here we have $P = \{x \mid Ax \le b\}$. The previous lemmas we had about the size of feasible solutions were slightly different as they were for different polytopes.

Repeat: Size of basic solutions

Proof:

Let $\bar{A} = \begin{bmatrix} A & -A & \\ -A & A & I_m \end{bmatrix}$, $\bar{b} = \begin{pmatrix} b \\ -b \end{pmatrix}$, be the matrix and right-hand vector after transforming the system to standard form.

The determinant of the matrices $\bar{A}_B$ and $\bar{M}_j$ (matrix obt. when replacing the j-th column of $\bar{A}_B$ by $\bar{b}$) can become at most

$$\begin{aligned} \det(\bar{A}_B), \det(\bar{M}_j) &\leq \|\vec{\ell}_{\max}\|^{2n} \\ &\leq (\sqrt{2n} \cdot 2^{\langle a_{\max} \rangle})^{2n} \leq 2^{2n\langle a_{\max}\rangle + 2n \log_2 n} \end{aligned} ,$$

where $\vec{\ell}_{\max}$ is the longest column-vector that can be obtained after deleting all but $2n$ rows and columns from $\bar{A}$.

This holds because columns from I_m selected when going from $\bar{A}$ to $\bar{A}_B$ do not increase the determinant. Only the at most $2n$ columns from matrices A and $-A$ that $\bar{A}$ consists of contribute.

How do we find the first ellipsoid?

For feasibility checking we can assume that the polytop P is bounded; it is sufficient to consider basic solutions.

Every entry x_i in a basic solution fulfills $|x_i| \leq \delta$.

Hence, P is contained in the cube $-\delta \leq x_i \leq \delta$.

A vector in this cube has at most distance $R := \sqrt{n}\delta$ from the origin.

Starting with the ball $E_0 := B(0, R)$ ensures that P is completely contained in the initial ellipsoid. This ellipsoid has volume at most $R^n B(0,1) \leq (n\delta)^n B(0,1)$.

How do we find the first ellipsoid?

For feasibility checking we can assume that the polytop P is bounded; it is sufficient to consider basic solutions.

Every entry x_i in a basic solution fulfills $|x_i| \leq \delta$.

Hence, P is contained in the cube $-\delta \leq x_i \leq \delta$.

A vector in this cube has at most distance $R := \sqrt{n}\delta$ from the origin.

Starting with the ball $E_0 := B(0, R)$ ensures that P is completely contained in the initial ellipsoid. This ellipsoid has volume at most $R^n B(0,1) \leq (n\delta)^n B(0,1)$.

How do we find the first ellipsoid?

For feasibility checking we can assume that the polytop P is bounded; it is sufficient to consider basic solutions.

Every entry x_i in a basic solution fulfills $|x_i| \leq \delta$.

Hence, P is contained in the cube $-\delta \leq x_i \leq \delta$.

A vector in this cube has at most distance $R := \sqrt{n}\delta$ from the origin.

Starting with the ball $E_0 := B(0, R)$ ensures that P is completely contained in the initial ellipsoid. This ellipsoid has volume at most $R^n B(0,1) \leq (n\delta)^n B(0,1)$.

How do we find the first ellipsoid?

For feasibility checking we can assume that the polytop P is bounded; it is sufficient to consider basic solutions.

Every entry x_i in a basic solution fulfills $|x_i| \leq \delta$.

Hence, P is contained in the cube $-\delta \leq x_i \leq \delta$.

A vector in this cube has at most distance $R := \sqrt{n}\delta$ from the origin.

Starting with the ball $E_0 := B(0, R)$ ensures that P is completely contained in the initial ellipsoid. This ellipsoid has volume at most $R^n B(0,1) \leq (n\delta)^n B(0,1)$.

How do we find the first ellipsoid?

For feasibility checking we can assume that the polytop P is bounded; it is sufficient to consider basic solutions.

Every entry x_i in a basic solution fulfills $|x_i| \leq \delta$.

Hence, P is contained in the cube $-\delta \leq x_i \leq \delta$.

A vector in this cube has at most distance $R := \sqrt{n}\delta$ from the origin.

Starting with the ball $E_0 := B(0, R)$ ensures that P is completely contained in the initial ellipsoid. This ellipsoid has volume at most $R^n B(0,1) \leq (n\delta)^n B(0,1)$.

How do we find the first ellipsoid?

For feasibility checking we can assume that the polytop P is bounded; it is sufficient to consider basic solutions.

Every entry x_i in a basic solution fulfills $|x_i| \leq \delta$.

Hence, P is contained in the cube $-\delta \leq x_i \leq \delta$.

A vector in this cube has at most distance $R := \sqrt{n}\delta$ from the origin.

Starting with the ball $E_0 := B(0, R)$ ensures that P is completely contained in the initial ellipsoid. This ellipsoid has volume at most $R^n B(0,1) \leq (n\delta)^n B(0,1)$.

When can we terminate?

Let $P := \{x \mid Ax \le b\}$ with $A \in \mathbb{Z}$ and $b \in \mathbb{Z}$ be a bounded polytop. Let $\langle a_{\max}\rangle$ be the encoding length of the largest entry in A or b.

Consider the following polytope

$$P_\lambda := \left\{x \mid Ax \le b + \frac{1}{\lambda}\begin{pmatrix}1\\ \vdots\\ 1\end{pmatrix}\right\} ,$$

where $\lambda = \delta^2 + 1$.

When can we terminate?

Let $P := \{x \mid Ax \leq b\}$ with $A \in \mathbb{Z}$ and $b \in \mathbb{Z}$ be a bounded polytop. Let $\langle a_{\max} \rangle$ be the encoding length of the largest entry in A or b.

Consider the following polytope

$$P_\lambda := \left\{ x \mid Ax \leq b + \frac{1}{\lambda} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \right\} ,$$

where $\lambda = \delta^2 + 1$.

When can we terminate?

Let $P := \{x \mid Ax \le b\}$ with $A \in \mathbb{Z}$ and $b \in \mathbb{Z}$ be a bounded polytop. Let $\langle a_{\max} \rangle$ be the encoding length of the largest entry in A or b.

Consider the following polytope

$$P_\lambda := \left\{ x \mid Ax \le b + \frac{1}{\lambda} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \right\} ,$$

where $\lambda = \delta^2 + 1$.

Lemma 38

P_λ is feasible if and only if P is feasible.

$\Longleftarrow$: obvious!

Lemma 38
P_λ is feasible if and only if P is feasible.

$\Leftarrow$: obvious!

⟹:

Consider the polytops

$$\bar{P} = \left\{ x \mid \begin{bmatrix} A & -A & \\ -A & A & I_m \end{bmatrix} x = \begin{pmatrix} b \\ -b \end{pmatrix} ; x \geq 0 \right\}$$

and

$$\bar{P}_\lambda = \left\{ x \mid \begin{bmatrix} A & -A & \\ -A & A & I_m \end{bmatrix} x = \begin{pmatrix} b \\ -b \end{pmatrix} + \frac{1}{\lambda} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} ; x \geq 0 \right\} .$$

P is feasible if and only if $\bar{P}$ is feasible, and P_λ feasible if and only if $\bar{P}_\lambda$ feasible.

$\bar{P}_\lambda$ is bounded since P_λ and P are bounded.

$\Longrightarrow$:

Consider the polytops

$$\bar{P} = \left\{ x \mid \begin{bmatrix} A & -A & I_m \\ -A & A & \end{bmatrix} x = \begin{pmatrix} b \\ -b \end{pmatrix} ; x \geq 0 \right\}$$

and

$$\bar{P}_\lambda = \left\{ x \mid \begin{bmatrix} A & -A & I_m \\ -A & A & \end{bmatrix} x = \begin{pmatrix} b \\ -b \end{pmatrix} + \frac{1}{\lambda} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} ; x \geq 0 \right\} .$$

P is feasible if and only if $\bar{P}$ is feasible, and P_λ feasible if and only if $\bar{P}_\lambda$ feasible.

$\bar{P}_\lambda$ is bounded since P_λ and P are bounded.

$\Rightarrow$:

Consider the polytops

$$\bar{P} = \left\{ x \mid \begin{bmatrix} A & -A & I_m \\ -A & A & \end{bmatrix} x = \begin{pmatrix} b \\ -b \end{pmatrix}; x \geq 0 \right\}$$

and

$$\bar{P}_\lambda = \left\{ x \mid \begin{bmatrix} A & -A & I_m \\ -A & A & \end{bmatrix} x = \begin{pmatrix} b \\ -b \end{pmatrix} + \frac{1}{\lambda} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}; x \geq 0 \right\} .$$

P is feasible if and only if $\bar{P}$ is feasible, and P_λ feasible if and only if $\bar{P}_\lambda$ feasible.

$\bar{P}_\lambda$ is bounded since P_λ and P are bounded.

$\Longrightarrow$:

Consider the polytops

$$\bar{P} = \left\{x \mid \begin{bmatrix} A & -A & I_m \\ -A & A & \end{bmatrix} x = \begin{pmatrix} b \\ -b \end{pmatrix}; x \geq 0\right\}$$

and

$$\bar{P}_\lambda = \left\{x \mid \begin{bmatrix} A & -A & I_m \\ -A & A & \end{bmatrix} x = \begin{pmatrix} b \\ -b \end{pmatrix} + \frac{1}{\lambda}\begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}; x \geq 0\right\} .$$

P is feasible if and only if $\bar{P}$ is feasible, and P_λ feasible if and only if $\bar{P}_\lambda$ feasible.

$\bar{P}_\lambda$ is bounded since P_λ and P are bounded.

Let $\bar{A} = \begin{bmatrix} A & -A & \\ -A & A & I_m \end{bmatrix}$, and $\bar{b} = \begin{pmatrix} b \\ -b \end{pmatrix}$.

$\bar{P}_\lambda$ feasible implies that there is a basic feasible solution represented by

$$x_B = \bar{A}_B^{-1}\bar{b} + \frac{1}{\lambda}\bar{A}_B^{-1}\begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

(The other x-values are zero)

The only reason that this basic feasible solution is not feasible for $\bar{P}$ is that one of the basic variables becomes negative.

Hence, there exists i with

$$(\bar{A}_B^{-1}\bar{b})_i < 0 \le (\bar{A}_B^{-1}\bar{b})_i + \frac{1}{\lambda}(\bar{A}_B^{-1}\vec{1})_i$$

Let $\bar{A} = \begin{bmatrix} A & -A & \\ -A & A & I_m \end{bmatrix}$, and $\bar{b} = \begin{pmatrix} b \\ -b \end{pmatrix}$.

$\bar{P}_\lambda$ feasible implies that there is a basic feasible solution represented by

$$x_B = \bar{A}_B^{-1}\bar{b} + \frac{1}{\lambda}\bar{A}_B^{-1}\begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

(The other x-values are zero)

The only reason that this basic feasible solution is not feasible for $\bar{P}$ is that one of the basic variables becomes negative.

Hence, there exists i with

$$(\bar{A}_B^{-1}\bar{b})_i < 0 \le (\bar{A}_B^{-1}\bar{b})_i + \frac{1}{\lambda}(\bar{A}_B^{-1}\vec{1})_i$$

Let $\bar{A} = \begin{bmatrix} A & -A & I_m \\ -A & A & \end{bmatrix}$, and $\bar{b} = \begin{pmatrix} b \\ -b \end{pmatrix}$.

$\bar{P}_\lambda$ feasible implies that there is a basic feasible solution represented by

$$x_B = \bar{A}_B^{-1}\bar{b} + \frac{1}{\lambda}\bar{A}_B^{-1}\begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

(The other x-values are zero)

The only reason that this basic feasible solution is not feasible for $\bar{P}$ is that one of the basic variables becomes negative.

Hence, there exists i with

$$(\bar{A}_B^{-1}\bar{b})_i < 0 \le (\bar{A}_B^{-1}\bar{b})_i + \frac{1}{\lambda}(\bar{A}_B^{-1}\vec{1})_i$$

By Cramers rule we get

$$(\bar{A}_B^{-1}\bar{b})_i < 0 \quad \Longrightarrow \quad (\bar{A}_B^{-1}\bar{b})_i \le -\frac{1}{\det(\bar{A}_B)}$$

and

$$(\bar{A}_B^{-1}\vec{1})_i \le \det(\bar{M}_j) \ ,$$

where $\bar{M}_j$ is obtained by replacing the j-th column of $\bar{A}_B$ by $\vec{1}$.

However, we showed that the determinants of $\bar{A}_B$ and $\bar{M}_j$ can become at most δ.

Since, we chose $\lambda = \delta^2 + 1$ this gives a contradiction.

By Cramers rule we get

$$(\bar{A}_B^{-1}\bar{b})_i < 0 \quad \Longrightarrow \quad (\bar{A}_B^{-1}\bar{b})_i \le -\frac{1}{\det(\bar{A}_B)}$$

and

$$(\bar{A}_B^{-1}\vec{1})_i \le \det(\bar{M}_j) \ ,$$

where $\bar{M}_j$ is obtained by replacing the j-th column of $\bar{A}_B$ by $\vec{1}$.

However, we showed that the determinants of $\bar{A}_B$ and $\bar{M}_j$ can become at most δ.

Since, we chose $\lambda = \delta^2 + 1$ this gives a contradiction.

By Cramers rule we get

$$(\bar{A}_B^{-1}\bar{b})_i < 0 \quad \Longrightarrow \quad (\bar{A}_B^{-1}\bar{b})_i \le -\frac{1}{\det(\bar{A}_B)}$$

and

$$(\bar{A}_B^{-1}\vec{1})_i \le \det(\bar{M}_j) \ ,$$

where $\bar{M}_j$ is obtained by replacing the j-th column of $\bar{A}_B$ by $\vec{1}$.

However, we showed that the determinants of $\bar{A}_B$ and $\bar{M}_j$ can become at most δ.

Since, we chose $\lambda = \delta^2 + 1$ this gives a contradiction.

Lemma 39

If P_λ is feasible then it contains a ball of radius $r := 1/\delta^3$. This has a volume of at least $r^n \mathrm{vol}(B(0,1)) = \frac{1}{\delta^{3n}} \mathrm{vol}(B(0,1))$.

Lemma 39

If P_λ is feasible then it contains a ball of radius $r := 1/\delta^3$. This has a volume of at least $r^n \mathrm{vol}(B(0,1)) = \frac{1}{\delta^{3n}} \mathrm{vol}(B(0,1))$.

Proof:

If P_λ feasible then also P. Let x be feasible for P.

Lemma 39

If P_λ is feasible then it contains a ball of radius $r := 1/\delta^3$. This has a volume of at least $r^n \mathrm{vol}(B(0,1)) = \frac{1}{\delta^{3n}} \mathrm{vol}(B(0,1))$.

Proof:

If P_λ feasible then also P. Let x be feasible for P.

This means $Ax \leq b$.

Lemma 39

If P_λ is feasible then it contains a ball of radius $r := 1/\delta^3$. This has a volume of at least $r^n \mathrm{vol}(B(0,1)) = \frac{1}{\delta^{3n}} \mathrm{vol}(B(0,1))$.

Proof:

If P_λ feasible then also P. Let x be feasible for P.
This means $Ax \le b$.

Let $\vec{\ell}$ with $\|\vec{\ell}\| \le r$. Then

$$(A(x + \vec{\ell}))_i$$

Lemma 39

If P_λ is feasible then it contains a ball of radius $r := 1/\delta^3$. This has a volume of at least $r^n \mathrm{vol}(B(0,1)) = \frac{1}{\delta^{3n}} \mathrm{vol}(B(0,1))$.

Proof:

If P_λ feasible then also P. Let x be feasible for P.
This means $Ax \leq b$.

Let $\vec{\ell}$ with $\|\vec{\ell}\| \leq r$. Then

$$(A(x + \vec{\ell}))_i = (Ax)_i + (A\vec{\ell})_i$$

Lemma 39

If P_λ is feasible then it contains a ball of radius $r := 1/\delta^3$. This has a volume of at least $r^n \mathrm{vol}(B(0,1)) = \frac{1}{\delta^{3n}} \mathrm{vol}(B(0,1))$.

Proof:

If P_λ feasible then also P. Let x be feasible for P.
This means $Ax \le b$.

Let $\vec{\ell}$ with $\|\vec{\ell}\| \le r$. Then

$$(A(x + \vec{\ell}))_i = (Ax)_i + (A\vec{\ell})_i \le b_i + A_i\vec{\ell}$$

Lemma 39

If P_λ is feasible then it contains a ball of radius $r := 1/\delta^3$. This has a volume of at least $r^n \mathrm{vol}(B(0,1)) = \frac{1}{\delta^{3n}} \mathrm{vol}(B(0,1))$.

Proof:

If P_λ feasible then also P. Let x be feasible for P.
This means $Ax \le b$.

Let $\vec{\ell}$ with $\|\vec{\ell}\| \le r$. Then

$$\begin{aligned}(A(x + \vec{\ell}))_i &= (Ax)_i + (A\vec{\ell})_i \le b_i + A_i\vec{\ell} \\ &\le b_i + \|A_i\| \cdot \|\vec{\ell}\|\end{aligned}$$

Lemma 39

If P_λ is feasible then it contains a ball of radius $r := 1/\delta^3$. This has a volume of at least $r^n \mathrm{vol}(B(0,1)) = \frac{1}{\delta^{3n}} \mathrm{vol}(B(0,1))$.

Proof:

If P_λ feasible then also P. Let x be feasible for P.
This means $Ax \le b$.

Let $\vec{\ell}$ with $\|\vec{\ell}\| \le r$. Then

$$
\begin{aligned}
(A(x+\vec{\ell}))_i &= (Ax)_i + (A\vec{\ell})_i \le b_i + A_i\vec{\ell} \\
&\le b_i + \|A_i\| \cdot \|\vec{\ell}\| \le b_i + \sqrt{n} \cdot 2^{\langle a_{\max}\rangle} \cdot r
\end{aligned}
$$

Lemma 39

If P_λ is feasible then it contains a ball of radius $r := 1/\delta^3$. This has a volume of at least $r^n \mathrm{vol}(B(0,1)) = \frac{1}{\delta^{3n}}\mathrm{vol}(B(0,1))$.

Proof:

If P_λ feasible then also P. Let x be feasible for P.

This means $Ax \le b$.

Let $\vec{\ell}$ with $\|\vec{\ell}\| \le r$. Then

$$\begin{aligned}(A(x+\vec{\ell}))_i &= (Ax)_i + (A\vec{\ell})_i \le b_i + A_i\vec{\ell}\\ &\le b_i + \|A_i\| \cdot \|\vec{\ell}\| \le b_i + \sqrt{n}\cdot 2^{\langle a_{\max}\rangle}\cdot r\\ &\le b_i + \frac{\sqrt{n}\cdot 2^{\langle a_{\max}\rangle}}{\delta^3}\end{aligned}$$

Lemma 39

If P_λ is feasible then it contains a ball of radius $r := 1/\delta^3$. This has a volume of at least $r^n \mathrm{vol}(B(0,1)) = \frac{1}{\delta^{3n}} \mathrm{vol}(B(0,1))$.

Proof:

If P_λ feasible then also P. Let x be feasible for P.

This means $Ax \le b$.

Let $\vec{\ell}$ with $\|\vec{\ell}\| \le r$. Then

$$\begin{aligned}(A(x+\vec{\ell}))_i &= (Ax)_i + (A\vec{\ell})_i \le b_i + A_i\vec{\ell}\\ &\le b_i + \|A_i\| \cdot \|\vec{\ell}\| \le b_i + \sqrt{n} \cdot 2^{\langle a_{\max}\rangle} \cdot r\\ &\le b_i + \frac{\sqrt{n} \cdot 2^{\langle a_{\max}\rangle}}{\delta^3} \le b_i + \frac{1}{\delta^2+1} \le b_i + \frac{1}{\lambda}\end{aligned}$$

Lemma 39

If P_λ is feasible then it contains a ball of radius $r := 1/\delta^3$. This has a volume of at least $r^n \mathrm{vol}(B(0,1)) = \frac{1}{\delta^{3n}}\mathrm{vol}(B(0,1))$.

Proof:

If P_λ feasible then also P. Let x be feasible for P.
This means $Ax \le b$.

Let $\vec{\ell}$ with $\|\vec{\ell}\| \le r$. Then

$$
\begin{aligned}
(A(x+\vec{\ell}))_i &= (Ax)_i + (A\vec{\ell})_i \le b_i + A_i\vec{\ell} \\
&\le b_i + \|A_i\| \cdot \|\vec{\ell}\| \le b_i + \sqrt{n} \cdot 2^{\langle a_{\max}\rangle} \cdot r \\
&\le b_i + \frac{\sqrt{n} \cdot 2^{\langle a_{\max}\rangle}}{\delta^3} \le b_i + \frac{1}{\delta^2+1} \le b_i + \frac{1}{\lambda}
\end{aligned}
$$

Hence, $x + \vec{\ell}$ is feasible for P_λ which proves the lemma.

How many iterations do we need until the volume becomes too small?

How many iterations do we need until the volume becomes too small?

$$e^{-\frac{i}{2(n+1)}} \cdot \mathrm{vol}(B(0,R)) < \mathrm{vol}(B(0,r))$$

How many iterations do we need until the volume becomes too small?

$$e^{-\frac{i}{2(n+1)}} \cdot \mathrm{vol}(B(0,R)) < \mathrm{vol}(B(0,r))$$

Hence,

$$i$$

How many iterations do we need until the volume becomes too small?

$$e^{-\frac{i}{2(n+1)}} \cdot \mathrm{vol}(B(0,R)) < \mathrm{vol}(B(0,r))$$

Hence,

$$i > 2(n+1)\ln\left(\frac{\mathrm{vol}(B(0,R))}{\mathrm{vol}(B(0,r))}\right)$$

How many iterations do we need until the volume becomes too small?

$$e^{-\frac{i}{2(n+1)}} \cdot \mathrm{vol}(B(0,R)) < \mathrm{vol}(B(0,r))$$

Hence,

$$\begin{aligned} i &> 2(n+1)\ln\left(\frac{\mathrm{vol}(B(0,R))}{\mathrm{vol}(B(0,r))}\right) \\ &= 2(n+1)\ln\left(n^n\delta^n \cdot \delta^{3n}\right) \end{aligned}$$

How many iterations do we need until the volume becomes too small?

$$e^{-\frac{i}{2(n+1)}} \cdot \mathrm{vol}(B(0,R)) < \mathrm{vol}(B(0,r))$$

Hence,

$$\begin{aligned} i &> 2(n+1)\ln\left(\frac{\mathrm{vol}(B(0,R))}{\mathrm{vol}(B(0,r))}\right) \\ &= 2(n+1)\ln\left(n^n\delta^n \cdot \delta^{3n}\right) \\ &= 8n(n+1)\ln(\delta) + 2(n+1)n\ln(n) \end{aligned}$$

How many iterations do we need until the volume becomes too small?

$$e^{-\frac{i}{2(n+1)}} \cdot \mathrm{vol}(B(0,R)) < \mathrm{vol}(B(0,r))$$

Hence,

$$\begin{aligned} i &> 2(n+1)\ln\left(\frac{\mathrm{vol}(B(0,R))}{\mathrm{vol}(B(0,r))}\right) \\ &= 2(n+1)\ln\left(n^n\delta^n \cdot \delta^{3n}\right) \\ &= 8n(n+1)\ln(\delta) + 2(n+1)n\ln(n) \\ &= \mathcal{O}(\mathrm{poly}(n, \langle a_{\max}\rangle)) \end{aligned}$$

Algorithm 1 ellipsoid-algorithm

input: point $c \in \mathbb{R}^n$, convex set $K \subseteq \mathbb{R}^n$, radii R and r
    with $K \subseteq B(c, R)$, and $B(x, r) \subseteq K$ for some x
output: point $x \in K$ or "K is empty"
$Q \leftarrow \mathrm{diag}(R^2, \ldots, R^2)$ // i.e., $L = \mathrm{diag}(R, \ldots, R)$
repeat
- **if** $c \in K$ **then return** c
- **else**
  - choose a violated hyperplane a
  - $c \leftarrow c - \frac{1}{n+1} \frac{Qa}{\sqrt{a^t Q a}}$
  - $Q \leftarrow \frac{n^2}{n^2-1}\Big(Q - \frac{2}{n+1}\frac{Qaa^tQ}{a^tQa}\Big)$
- **endif**

until $\det(Q) \le r^{2n}$ // i.e., $\det(L) \le r^n$
return "K is empty"

Separation Oracle:

Let $K \subseteq \mathbb{R}^n$ be a convex set. A separation oracle for K is an algorithm A that gets as input a point $x \in \mathbb{R}^n$ and either

- certifies that $x \in K$,
- or finds a hyperplane separating x from K.

We will usually assume that A is a polynomial-time algorithm.

In order to find a point in K we need

- a guarantee that a ball of radius r is contained in K,
- an initial ball $B(c, R)$ with radius R that contains K,
- a separation oracle for K.

The Ellipsoid algorithm requires $\mathcal{O}(\text{poly}(n) \cdot \log(R/r))$ iterations. Each iteration is polytime for a polynomial-time Separation oracle.

Separation Oracle:

Let $K \subseteq \mathbb{R}^n$ be a convex set. A separation oracle for K is an algorithm A that gets as input a point $x \in \mathbb{R}^n$ and either

- certifies that $x \in K$,
- or finds a hyperplane separating x from K.

We will usually assume that A is a polynomial-time algorithm.

In order to find a point in K we need

- a guarantee that a ball of radius r is contained in K,
- an initial ball $B(c, R)$ with radius R that contains K,
- a separation oracle for K.

The Ellipsoid algorithm requires $\mathcal{O}(\text{poly}(n) \cdot \log(R/r))$ iterations. Each iteration is polytime for a polynomial-time Separation oracle.

Separation Oracle:

Let $K \subseteq \mathbb{R}^n$ be a convex set. A separation oracle for K is an algorithm A that gets as input a point $x \in \mathbb{R}^n$ and either

- certifies that $x \in K$,
- or finds a hyperplane separating x from K.

We will usually assume that A is a polynomial-time algorithm.

In order to find a point in K we need

- a guarantee that a ball of radius r is contained in K,
- an initial ball $B(c,R)$ with radius R that contains K,
- a separation oracle for K.

The Ellipsoid algorithm requires $\mathcal{O}(\text{poly}(n) \cdot \log(R/r))$ iterations. Each iteration is polytime for a polynomial-time Separation oracle.

Separation Oracle:

Let $K \subseteq \mathbb{R}^n$ be a convex set. A separation oracle for K is an algorithm A that gets as input a point $x \in \mathbb{R}^n$ and either

- certifies that $x \in K$,
- or finds a hyperplane separating x from K.

We will usually assume that A is a polynomial-time algorithm.

In order to find a point in K we need

- a guarantee that a ball of radius r is contained in K,
- an initial ball $B(c, R)$ with radius R that contains K,
- a separation oracle for K.

The Ellipsoid algorithm requires $\mathcal{O}(\text{poly}(n) \cdot \log(R/r))$ iterations. Each iteration is polytime for a polynomial-time Separation oracle.

Separation Oracle:

Let $K \subseteq \mathbb{R}^n$ be a convex set. A separation oracle for K is an algorithm A that gets as input a point $x \in \mathbb{R}^n$ and either

- certifies that $x \in K$,
- or finds a hyperplane separating x from K.

We will usually assume that A is a polynomial-time algorithm.

In order to find a point in K we need

- a guarantee that a ball of radius r is contained in K,
- an initial ball $B(c, R)$ with radius R that contains K,
- a separation oracle for K.

The Ellipsoid algorithm requires $\mathcal{O}(\text{poly}(n) \cdot \log(R/r))$ iterations. Each iteration is polytime for a polynomial-time Separation oracle.

Separation Oracle:

Let $K \subseteq \mathbb{R}^n$ be a convex set. A separation oracle for K is an algorithm A that gets as input a point $x \in \mathbb{R}^n$ and either

- certifies that $x \in K$,
- or finds a hyperplane separating x from K.

We will usually assume that A is a polynomial-time algorithm.

In order to find a point in K we need

- a guarantee that a ball of radius r is contained in K,
- an initial ball $B(c, R)$ with radius R that contains K,
- a separation oracle for K.

The Ellipsoid algorithm requires $\mathcal{O}(\text{poly}(n) \cdot \log(R/r))$ iterations. Each iteration is polytime for a polynomial-time Separation oracle.

Separation Oracle:

Let $K \subseteq \mathbb{R}^n$ be a convex set. A separation oracle for K is an algorithm A that gets as input a point $x \in \mathbb{R}^n$ and either

- certifies that $x \in K$,
- or finds a hyperplane separating x from K.

We will usually assume that A is a polynomial-time algorithm.

In order to find a point in K we need

- a guarantee that a ball of radius r is contained in K,
- an initial ball $B(c, R)$ with radius R that contains K,
- a separation oracle for K.

The Ellipsoid algorithm requires $\mathcal{O}(\text{poly}(n) \cdot \log(R/r))$ iterations. Each iteration is polytime for a polynomial-time Separation oracle.

10 Karmarkars Algorithm

We want to solve the following linear program:

- $\min v = c^t x$ subject to $Ax = 0$ and $x \in \Delta$.
- Here $\Delta = \{x \in \mathbb{R}^n \mid e^t x = 1, x \geq 0\}$ with $e^t = (1, \dots, 1)$ denotes the standard simplex in $\mathbb{R}^n$.

Further assumptions:

1. A is an $m \times n$-matrix with rank m.
2. $Ae = 0$, i.e., the center of the simplex is feasible.
3. The optimum solution is 0.

10 Karmarkars Algorithm

We want to solve the following linear program:

- $\min v = c^t x$ subject to $Ax = 0$ and $x \in \Delta$.
- Here $\Delta = \{x \in \mathbb{R}^n \mid e^t x = 1, x \geq 0\}$ with $e^t = (1, \dots, 1)$ denotes the standard simplex in $\mathbb{R}^n$.

Further assumptions:

1. A is an $m \times n$-matrix with rank m.
2. $Ae = 0$, i.e., the center of the simplex is feasible.
3. The optimum solution is 0.

10 Karmarkars Algorithm

We want to solve the following linear program:

- $\min v = c^t x$ subject to $Ax = 0$ and $x \in \Delta$.
- Here $\Delta = \{x \in \mathbb{R}^n \mid e^t x = 1, x \geq 0\}$ with $e^t = (1, \ldots, 1)$ denotes the standard simplex in $\mathbb{R}^n$.

Further assumptions:

1. A is an $m \times n$-matrix with rank m.
2. $Ae = 0$, i.e., the center of the simplex is feasible.
3. The optimum solution is 0.

10 Karmarkars Algorithm

We want to solve the following linear program:

- $\min v = c^t x$ subject to $Ax = 0$ and $x \in \Delta$.
- Here $\Delta = \{x \in \mathbb{R}^n \mid e^t x = 1, x \geq 0\}$ with $e^t = (1, \ldots, 1)$ denotes the standard simplex in $\mathbb{R}^n$.

Further assumptions:

1. A is an $m \times n$-matrix with rank m.
2. $Ae = 0$, i.e., the center of the simplex is feasible.
3. The optimum solution is 0.

10 Karmarkars Algorithm

We want to solve the following linear program:

- $\min v = c^t x$ subject to $Ax = 0$ and $x \in \Delta$.
- Here $\Delta = \{x \in \mathbb{R}^n \mid e^t x = 1, x \geq 0\}$ with $e^t = (1, \ldots, 1)$ denotes the standard simplex in $\mathbb{R}^n$.

Further assumptions:

1. A is an $m \times n$-matrix with rank m.
2. $Ae = 0$, i.e., the center of the simplex is feasible.
3. The optimum solution is 0.

10 Karmarkars Algorithm

Suppose you start with $\max\{c^t x \mid Ax = b; x \geq 0\}$.

- Multiply c by -1 and do a minimization. $\Rightarrow$ minimization problem
- We can check for feasibility by using the two phase algorithm. $\Rightarrow$ can assume that LP is feasible.
- Compute the dual; pack primal and dual into one LP and minimize the duality gap. $\Rightarrow$ optimum is 0
- Add a new variable pair x_ℓ, x'_ℓ (both restricted to be positive) and the constraint $\sum_i x_i = 1$. $\Rightarrow$ solution in simplex
- Add $-(\sum_i x_i) b_i = -b_i$ to every constraint. $\Rightarrow$ vector b is 0
- If A does not have full row rank we can delete constraints (or conclude that the LP is infeasible).
  $\Rightarrow A$ has full row rank

We still need to make e/n feasible.

10 Karmarkars Algorithm

Suppose you start with $\max\{c^t x \mid Ax = b; x \geq 0\}$.

- Multiply c by -1 and do a minimization. $\Rightarrow$ minimization problem
- We can check for feasibility by using the two phase algorithm. $\Rightarrow$ can assume that LP is feasible.
- Compute the dual; pack primal and dual into one LP and minimize the duality gap. $\Rightarrow$ optimum is 0
- Add a new variable pair x_ℓ, x'_ℓ (both restricted to be positive) and the constraint $\sum_i x_i = 1$. $\Rightarrow$ solution in simplex
- Add $-(\sum_i x_i) b_i = -b_i$ to every constraint. $\Rightarrow$ vector b is 0
- If A does not have full row rank we can delete constraints (or conclude that the LP is infeasible).
  $\Rightarrow$ A has full row rank

We still need to make e/n feasible.

10 Karmarkars Algorithm

Suppose you start with $\max\{c^t x \mid Ax = b; x \geq 0\}$.

- Multiply c by -1 and do a minimization. ⇒ minimization problem
- We can check for feasibility by using the two phase algorithm. ⇒ can assume that LP is feasible.
- Compute the dual; pack primal and dual into one LP and minimize the duality gap. ⇒ optimum is 0
- Add a new variable pair x_ℓ, x'_ℓ (both restricted to be positive) and the constraint $\sum_i x_i = 1$. ⇒ solution in simplex
- Add $-(\sum_i x_i)b_i = -b_i$ to every constraint. ⇒ vector b is 0
- If A does not have full row rank we can delete constraints (or conclude that the LP is infeasible).
  ⇒ A has full row rank

We still need to make e/n feasible.

10 Karmarkars Algorithm

Suppose you start with $\max\{c^t x \mid Ax = b; x \geq 0\}$.

- Multiply c by -1 and do a minimization. $\Rightarrow$ minimization problem
- We can check for feasibility by using the two phase algorithm. $\Rightarrow$ can assume that LP is feasible.
- Compute the dual; pack primal and dual into one LP and minimize the duality gap. $\Rightarrow$ optimum is 0
- Add a new variable pair x_ℓ, x'_ℓ (both restricted to be positive) and the constraint $\sum_i x_i = 1$. $\Rightarrow$ solution in simplex
- Add $-(\sum_i x_i)b_i = -b_i$ to every constraint. $\Rightarrow$ vector b is 0
- If A does not have full row rank we can delete constraints (or conclude that the LP is infeasible).
  $\Rightarrow$ A has full row rank

We still need to make e/n feasible.

10 Karmarkars Algorithm

Suppose you start with $\max\{c^t x \mid Ax = b; x \geq 0\}$.

- Multiply c by -1 and do a minimization. $\Rightarrow$ minimization problem
- We can check for feasibility by using the two phase algorithm. $\Rightarrow$ can assume that LP is feasible.
- Compute the dual; pack primal and dual into one LP and minimize the duality gap. $\Rightarrow$ optimum is 0
- Add a new variable pair x_ℓ, x'_ℓ (both restricted to be positive) and the constraint $\sum_i x_i = 1$. $\Rightarrow$ solution in simplex
- Add $-(\sum_i x_i) b_i = -b_i$ to every constraint. $\Rightarrow$ vector b is 0
- If A does not have full row rank we can delete constraints (or conclude that the LP is infeasible).
  $\Rightarrow$ A has full row rank

We still need to make e/n feasible.

10 Karmarkars Algorithm

Suppose you start with $\max\{c^t x \mid Ax = b; x \geq 0\}$.

- Multiply c by -1 and do a minimization. $\Rightarrow$ minimization problem
- We can check for feasibility by using the two phase algorithm. $\Rightarrow$ can assume that LP is feasible.
- Compute the dual; pack primal and dual into one LP and minimize the duality gap. $\Rightarrow$ optimum is 0
- Add a new variable pair x_ℓ, x'_ℓ (both restricted to be positive) and the constraint $\sum_i x_i = 1$. $\Rightarrow$ solution in simplex
- Add $-(\sum_i x_i) b_i = -b_i$ to every constraint. $\Rightarrow$ vector b is 0
- If A does not have full row rank we can delete constraints (or conclude that the LP is infeasible).
  $\Rightarrow$ A has full row rank

We still need to make e/n feasible.

10 Karmarkars Algorithm

Suppose you start with $\max\{c^t x \mid Ax = b; x \geq 0\}$.

- Multiply c by -1 and do a minimization. $\Rightarrow$ minimization problem
- We can check for feasibility by using the two phase algorithm. $\Rightarrow$ can assume that LP is feasible.
- Compute the dual; pack primal and dual into one LP and minimize the duality gap. $\Rightarrow$ optimum is 0
- Add a new variable pair x_ℓ, x'_ℓ (both restricted to be positive) and the constraint $\sum_i x_i = 1$. $\Rightarrow$ solution in simplex
- Add $-(\sum_i x_i)b_i = -b_i$ to every constraint. $\Rightarrow$ vector b is 0
- If A does not have full row rank we can delete constraints (or conclude that the LP is infeasible).
  $\Rightarrow$ A has full row rank

We still need to make e/n feasible.

10 Karmarkars Algorithm

Suppose you start with $\max\{c^t x \mid Ax = b; x \geq 0\}$.

- Multiply c by -1 and do a minimization. $\Rightarrow$ minimization problem
- We can check for feasibility by using the two phase algorithm. $\Rightarrow$ can assume that LP is feasible.
- Compute the dual; pack primal and dual into one LP and minimize the duality gap. $\Rightarrow$ optimum is 0
- Add a new variable pair x_ℓ, x'_ℓ (both restricted to be positive) and the constraint $\sum_i x_i = 1$. $\Rightarrow$ solution in simplex
- Add $-(\sum_i x_i) b_i = -b_i$ to every constraint. $\Rightarrow$ vector b is 0
- If A does not have full row rank we can delete constraints (or conclude that the LP is infeasible).
  $\Rightarrow$ A has full row rank

We still need to make e/n feasible.

10 Karmarkars Algorithm

The algorithm computes strictly feasible interior points $x^{(0)} = \frac{e}{n}, x^{(1)}, x^{(2)}, \ldots$ with

$$c^t x^{(k)} \leq 2^{-\Theta(L)} c^t x^{(0)}$$

For $k = \Theta(L)$. A point x is strictly feasible if $x > 0$.

If my objective value is close enough to 0 (the optimum!!) I can "snap" to an optimum vertex.

10 Karmarkars Algorithm

The algorithm computes strictly feasible interior points $x^{(0)} = \frac{e}{n}, x^{(1)}, x^{(2)}, \dots$ with

$$c^t x^{(k)} \leq 2^{-\Theta(L)} c^t x^{(0)}$$

For $k = \Theta(L)$. A point x is strictly feasible if $x > 0$.

If my objective value is close enough to 0 (the optimum!!) I can "snap" to an optimum vertex.

10 Karmarkars Algorithm

Iteration:

1. Distort the problem by mapping the simplex onto itself so that the current point $\bar{x}$ moves to the center.
2. Project the optimization direction c onto the feasible region. Determine a distance to travel along this direction such that you do not leave the simplex (and you do not touch the border). $\hat{x}_{\text{new}}$ is the point you reached.
3. Do a backtransformation to transform $\hat{x}$ into your new point $\bar{x}_{\text{new}}$.

10 Karmarkars Algorithm

Iteration:

1. Distort the problem by mapping the simplex onto itself so that the current point $\bar{x}$ moves to the center.
2. Project the optimization direction c onto the feasible region. Determine a distance to travel along this direction such that you do not leave the simplex (and you do not touch the border). $\hat{x}_{\text{new}}$ is the point you reached.
3. Do a backtransformation to transform $\hat{x}$ into your new point $\bar{x}_{\text{new}}$.

10 Karmarkars Algorithm

Iteration:

1. Distort the problem by mapping the simplex onto itself so that the current point $\bar{x}$ moves to the center.
2. Project the optimization direction c onto the feasible region. Determine a distance to travel along this direction such that you do not leave the simplex (and you do not touch the border). $\hat{x}_{\text{new}}$ is the point you reached.
3. Do a backtransformation to transform $\hat{x}$ into your new point $\bar{x}_{\text{new}}$.

The Transformation

Let $\bar{Y} = \mathrm{diag}(\bar{x})$ the diagonal matrix with entries $\bar{x}$ on the diagonal.

Define

$$F_{\bar{x}} : x \mapsto \frac{\bar{Y}^{-1}x}{e^t\bar{Y}^{-1}x} \enspace .$$

The inverse function is

$$F_{\bar{x}}^{-1} : \hat{x} \mapsto \frac{\bar{Y}\hat{x}}{e^t\bar{Y}\hat{x}} \enspace .$$

Note that $\bar{x} > 0$ in every coordinate. Therefore the above is well defined.

The Transformation

Let $\bar{Y} = \operatorname{diag}(\bar{x})$ the diagonal matrix with entries $\bar{x}$ on the diagonal.

Define

$$F_{\bar{x}} : x \mapsto \frac{\bar{Y}^{-1}x}{e^t \bar{Y}^{-1}x} \enspace .$$

The inverse function is

$$F_{\bar{x}}^{-1} : \hat{x} \mapsto \frac{\bar{Y}\hat{x}}{e^t \bar{Y}\hat{x}} \enspace .$$

Note that $\bar{x} > 0$ in every coordinate. Therefore the above is well defined.

The Transformation

Let $\bar{Y} = \mathrm{diag}(\bar{x})$ the diagonal matrix with entries $\bar{x}$ on the diagonal.

Define

$$F_{\bar{x}} : x \mapsto \frac{\bar{Y}^{-1}x}{e^t\bar{Y}^{-1}x} \enspace .$$

The inverse function is

$$F_{\bar{x}}^{-1} : \hat{x} \mapsto \frac{\bar{Y}\hat{x}}{e^t\bar{Y}\hat{x}} \enspace .$$

Note that $\bar{x} > 0$ in every coordinate. Therefore the above is well defined.

The Transformation

Let $\bar{Y} = \mathrm{diag}(\bar{x})$ the diagonal matrix with entries $\bar{x}$ on the diagonal.

Define

$$F_{\bar{x}} : x \mapsto \frac{\bar{Y}^{-1}x}{e^t\bar{Y}^{-1}x} \enspace .$$

The inverse function is

$$F_{\bar{x}}^{-1} : \hat{x} \mapsto \frac{\bar{Y}\hat{x}}{e^t\bar{Y}\hat{x}} \enspace .$$

Note that $\bar{x} > 0$ in every coordinate. Therefore the above is well defined.

Properties

$F_{\bar{x}}^{-1}$ really is the inverse of $F_{\bar{x}}$:

$$F_{\bar{x}}(F_{\bar{x}}^{-1}(\hat{x})) = \frac{\bar{Y}^{-1}\frac{\bar{Y}\hat{x}}{e^t\bar{Y}\hat{x}}}{e^t\bar{Y}^{-1}\frac{\bar{Y}\hat{x}}{e^t\bar{Y}\hat{x}}} = \frac{\hat{x}}{e^t\hat{x}} = \hat{x}$$

because $\hat{x} \in \Delta$.

Note that in particular every $\hat{x} \in \Delta$ has a preimage (Urbild) under $F_{\bar{x}}$.

Properties

$\bar{x}$ is mapped to e/n

$$F_{\bar{x}}(\bar{x}) = \frac{\bar{Y}^{-1}\bar{x}}{e^t\bar{Y}^{-1}\bar{x}} = \frac{e}{e^t e} = \frac{e}{n}$$

Properties

A unit vectors e_i is mapped to itself:

$$F_{\bar{x}}(e_i) = \frac{\bar{Y}^{-1}e_i}{e^t\bar{Y}^{-1}e_i} = \frac{(0,\dots,0,1/\bar{x}_i,0,\dots,0)^t}{e^t(0,\dots,0,1/\bar{x}_i,0,\dots,0)^t} = e_i$$

Properties

All nodes of the simplex are mapped to the simplex:

$$F_{\bar{x}}(x) = \frac{\bar{Y}^{-1}x}{e^t\bar{Y}^{-1}x} = \frac{\left(\frac{x_1}{\bar{x}_1},\dots,\frac{x_n}{\bar{x}_n}\right)^t}{e^t\left(\frac{x_1}{\bar{x}_1},\dots,\frac{x_n}{\bar{x}_n}\right)^t} = \frac{\left(\frac{x_1}{\bar{x}_1},\dots,\frac{x_n}{\bar{x}_n}\right)^t}{\sum_i \frac{x_i}{\bar{x}_i}} \in \Delta$$

The Transformation

Easy to check:

- $F_{\bar{x}}^{-1}$ really is the inverse of $F_{\bar{x}}$.
- $\bar{x}$ is mapped to e/n.
- A unit vectors e_i is mapped to itself.
- All nodes of the simplex are mapped to the simplex.

The Transformation

Easy to check:

- $F_{\bar{x}}^{-1}$ really is the inverse of $F_{\bar{x}}$.
- $\bar{x}$ is mapped to e/n.
- A unit vectors e_i is mapped to itself.
- All nodes of the simplex are mapped to the simplex.

The Transformation

Easy to check:

- $F_{\bar{x}}^{-1}$ really is the inverse of $F_{\bar{x}}$.
- $\bar{x}$ is mapped to e/n.
- A unit vectors e_i is mapped to itself.
- All nodes of the simplex are mapped to the simplex.

The Transformation

Easy to check:

- $F_{\bar{x}}^{-1}$ really is the inverse of $F_{\bar{x}}$.
- $\bar{x}$ is mapped to e/n.
- A unit vectors e_i is mapped to itself.
- All nodes of the simplex are mapped to the simplex.

10 Karmarkars Algorithm

We have the problem

$$\min\{c^t x \mid Ax = 0;\, x \in \Delta\}$$

10 Karmarkars Algorithm

We have the problem

$$\begin{aligned}&\min\{c^t x \mid Ax = 0;\, x \in \Delta\}\\&\quad = \min\{c^t F_{\bar{x}}^{-1}(\hat{x}) \mid AF_{\bar{x}}^{-1}(\hat{x}) = 0;\, F_{\bar{x}}^{-1}(\hat{x}) \in \Delta\}\end{aligned}$$

10 Karmarkars Algorithm

We have the problem

$$
\begin{aligned}
&\min\{c^t x \mid Ax = 0;\, x \in \Delta\} \\
&\quad = \min\{c^t F_{\bar{x}}^{-1}(\hat{x}) \mid AF_{\bar{x}}^{-1}(\hat{x}) = 0;\, F_{\bar{x}}^{-1}(\hat{x}) \in \Delta\} \\
&\quad = \min\{c^t F_{\bar{x}}^{-1}(\hat{x}) \mid AF_{\bar{x}}^{-1}(\hat{x}) = 0;\, \hat{x} \in \Delta\}
\end{aligned}
$$

10 Karmarkars Algorithm

We have the problem

$$\begin{aligned}
&\min\{c^t x \mid Ax = 0;\, x \in \Delta\}\\
&\quad = \min\{c^t F_{\bar{x}}^{-1}(\hat{x}) \mid AF_{\bar{x}}^{-1}(\hat{x}) = 0;\, F_{\bar{x}}^{-1}(\hat{x}) \in \Delta\}\\
&\quad = \min\{c^t F_{\bar{x}}^{-1}(\hat{x}) \mid AF_{\bar{x}}^{-1}(\hat{x}) = 0;\, \hat{x} \in \Delta\}\\
&\quad = \min\left\{\frac{c^t \bar{Y}\hat{x}}{e^t \bar{Y}\hat{x}} \mid \frac{A\bar{Y}\hat{x}}{e^t \bar{Y}\hat{x}} = 0;\, \hat{x} \in \Delta\right\}
\end{aligned}$$

10 Karmarkars Algorithm

We have the problem

$$
\begin{aligned}
\min\{c^t x \mid Ax = 0;\, x \in \Delta\} \\
&= \min\{c^t F_{\bar{x}}^{-1}(\hat{x}) \mid AF_{\bar{x}}^{-1}(\hat{x}) = 0;\, F_{\bar{x}}^{-1}(\hat{x}) \in \Delta\} \\
&= \min\{c^t F_{\bar{x}}^{-1}(\hat{x}) \mid AF_{\bar{x}}^{-1}(\hat{x}) = 0;\, \hat{x} \in \Delta\} \\
&= \min\left\{\frac{c^t \bar{Y}\hat{x}}{e^t \bar{Y}\hat{x}} \mid \frac{A\bar{Y}\hat{x}}{e^t \bar{Y}\hat{x}} = 0;\, \hat{x} \in \Delta\right\}
\end{aligned}
$$

Since the optimum solution is 0 this problem is the same as

$$
\min\{\hat{c}^t \hat{x} \mid \hat{A}\hat{x} = 0, \hat{x} \in \Delta\}
$$

with $\hat{c} = \bar{Y}^t c = \bar{Y}c$ and $\hat{A} = A\bar{Y}$.

We still need to make e/n feasible.

- We know that our LP is feasible. Let $\bar{x}$ be a feasible point.
- Apply $F_{\bar{x}}$, and solve

$$\min\{\hat{c}^t x \mid \hat{A}x = 0; x \in \Delta\}$$

- The feasible point is moved to the center.

10 Karmarkars Algorithm

When computing $\hat{x}_{\text{new}}$ we do not want to leave the simplex or touch its boundary (why?).

For this we compute the radius of a ball that completely lies in the simplex.

$$B\left(\frac{e}{n},\rho\right) = \left\{x \in \mathbb{R}^n \mid \left\|x - \frac{e}{n}\right\| \leq \rho\right\} \ .$$

We are looking for the largest radius r such that

$$B\left(\frac{e}{n},r\right) \cap \left\{x \mid e^t x = 1\right\} \subseteq \Delta.$$

10 Karmarkars Algorithm

When computing $\hat{x}_{\text{new}}$ we do not want to leave the simplex or touch its boundary (why?).

For this we compute the radius of a ball that completely lies in the simplex.

$$B\left(\frac{e}{n},\rho\right) = \left\{x \in \mathbb{R}^n \mid \left\|x - \frac{e}{n}\right\| \le \rho\right\} \;.$$

We are looking for the largest radius r such that

$$B\left(\frac{e}{n},r\right) \cap \left\{x \mid e^t x = 1\right\} \subseteq \Delta.$$

10 Karmarkars Algorithm

When computing $\hat{x}_{\text{new}}$ we do not want to leave the simplex or touch its boundary (why?).

For this we compute the radius of a ball that completely lies in the simplex.

$$B\left(\frac{e}{n}, \rho\right) = \left\{x \in \mathbb{R}^n \mid \left\|x - \frac{e}{n}\right\| \le \rho\right\} .$$

We are looking for the largest radius r such that

$$B\left(\frac{e}{n}, r\right) \cap \left\{x \mid e^t x = 1\right\} \subseteq \Delta.$$

10 Karmarkars Algorithm

When computing $\hat{x}_{\text{new}}$ we do not want to leave the simplex or touch its boundary (why?).

For this we compute the radius of a ball that completely lies in the simplex.

$$B\left(\frac{e}{n}, \rho\right) = \left\{x \in \mathbb{R}^n \mid \left\|x - \frac{e}{n}\right\| \le \rho\right\} \quad .$$

We are looking for the largest radius r such that

$$B\left(\frac{e}{n}, r\right) \cap \left\{x \mid e^t x = 1\right\} \subseteq \Delta.$$

10 Karmarkars Algorithm

This holds for $r = \|\frac{e}{n} - (e - e_1)\frac{1}{n-1}\|$. ($r$ is the distance between the center e/n and the center of the $(n-1)$-dimensional simplex obtained by intersecting a side ($x_i = 0$) of the unit cube with Δ.)

This gives $r = \frac{1}{\sqrt{n(n-1)}}$.

Now we consider the problem

$$\min\{\hat{c}^t x \mid \hat{A}x = 0, x \in B(e/n, r) \cap \Delta\}$$

This problem is easy to solve!!!

10 Karmarkars Algorithm

This holds for $r = \|\frac{e}{n} - (e - e_1)\frac{1}{n-1}\|$. ($r$ is the distance between the center e/n and the center of the $(n-1)$-dimensional simplex obtained by intersecting a side ($x_i = 0$) of the unit cube with Δ.)

This gives $r = \frac{1}{\sqrt{n(n-1)}}$.

Now we consider the problem

$$\min\{\hat{c}^t x \mid \hat{A}x = 0, x \in B(e/n, r) \cap \Delta\}$$

This problem is easy to solve!!!

10 Karmarkars Algorithm

This holds for $r = \|\frac{e}{n} - (e - e_1)\frac{1}{n-1}\|$. ($r$ is the distance between the center e/n and the center of the $(n-1)$-dimensional simplex obtained by intersecting a side ($x_i = 0$) of the unit cube with Δ.)

This gives $r = \frac{1}{\sqrt{n(n-1)}}$.

Now we consider the problem

$$\min\{\hat{c}^t x \mid \hat{A}x = 0, x \in B(e/n, r) \cap \Delta\}$$

This problem is easy to solve!!!

10 Karmarkars Algorithm

This holds for $r = \|\frac{e}{n} - (e - e_1)\frac{1}{n-1}\|$. ($r$ is the distance between the center e/n and the center of the $(n-1)$-dimensional simplex obtained by intersecting a side ($x_i = 0$) of the unit cube with Δ.)

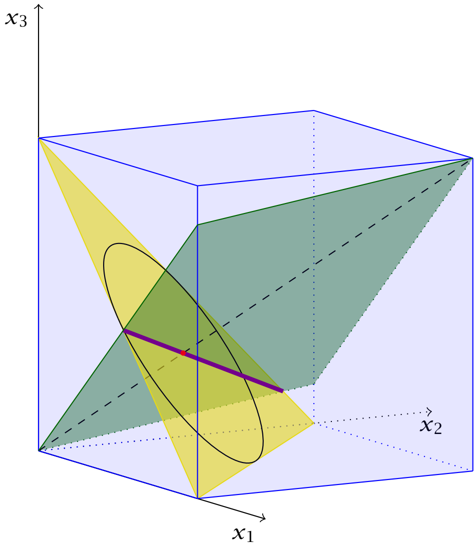
This gives $r = \frac{1}{\sqrt{n(n-1)}}$.

Now we consider the problem

$$\min\{\hat{c}^t x \mid \hat{A}x = 0, x \in B(e/n, r) \cap \Delta\}$$

This problem is easy to solve!!!

The Simplex



10 Karmarkars Algorithm

Ideally we would like to go in direction of $-\hat{c}$ (starting from the center of the simplex).

However, doing this may violate constraints $\hat{A}\hat{x} = 0$ or the constraint $\hat{x} \in \Delta$.

Therefore we first project $\hat{c}$ on the nullspace of

$$B = \begin{pmatrix} \hat{A} \\ e^t \end{pmatrix}$$

We use

$$P = I - B^t(BB^t)^{-1}B$$

Then

$$\hat{d} = P\hat{c}$$

is the required projection.

10 Karmarkars Algorithm

Ideally we would like to go in direction of $-\hat{c}$ (starting from the center of the simplex).

However, doing this may violate constraints $\hat{A}\hat{x} = 0$ or the constraint $\hat{x} \in \Delta$.

Therefore we first project $\hat{c}$ on the nullspace of

$$B = \begin{pmatrix} \hat{A} \\ e^t \end{pmatrix}$$

We use

$$P = I - B^t (BB^t)^{-1} B$$

Then

$$\hat{d} = P\hat{c}$$

is the required projection.

10 Karmarkars Algorithm

Ideally we would like to go in direction of $-\hat{c}$ (starting from the center of the simplex).

However, doing this may violate constraints $\hat{A}\hat{x} = 0$ or the constraint $\hat{x} \in \Delta$.

Therefore we first project $\hat{c}$ on the nullspace of

$$B = \begin{pmatrix} \hat{A} \\ e^t \end{pmatrix}$$

We use

$$P = I - B^t(BB^t)^{-1}B$$

Then

$$\hat{d} = P\hat{c}$$

is the required projection.

10 Karmarkars Algorithm

Ideally we would like to go in direction of $-\hat{c}$ (starting from the center of the simplex).

However, doing this may violate constraints $\hat{A}\hat{x} = 0$ or the constraint $\hat{x} \in \Delta$.

Therefore we first project $\hat{c}$ on the nullspace of

$$B = \begin{pmatrix} \hat{A} \\ e^t \end{pmatrix}$$

We use

$$P = I - B^t(BB^t)^{-1}B$$

Then

$$\hat{d} = P\hat{c}$$

is the required projection.

10 Karmarkars Algorithm

Ideally we would like to go in direction of $-\hat{c}$ (starting from the center of the simplex).

However, doing this may violate constraints $\hat{A}\hat{x} = 0$ or the constraint $\hat{x} \in \Delta$.

Therefore we first project $\hat{c}$ on the nullspace of

$$B = \begin{pmatrix} \hat{A} \\ e^t \end{pmatrix}$$

We use

$$P = I - B^t(BB^t)^{-1}B$$

Then

$$\hat{d} = P\hat{c}$$

is the required projection.

10 Karmarkars Algorithm

We get the new point

$$\hat{x}(\rho) = \frac{e}{n} - \rho \frac{\hat{d}}{\|\hat{d}\|}$$

for $\rho < r$.

Choose $\rho = \alpha r$ with $\alpha = 1/4$.

10 Karmarkars Algorithm

We get the new point

$$\hat{x}(\rho) = \frac{e}{n} - \rho \frac{\hat{d}}{\|\hat{d}\|}$$

for $\rho < r$.

Choose $\rho = \alpha r$ with $\alpha = 1/4$.

Iteration of Karmarkars Algorithm

- Current solution $\bar{x}$. $\bar{Y} := \text{diag}(\bar{x}_1, \dots, \bar{x}_n)$.
- Transform problem via $F_{\bar{x}}(x) = \frac{\bar{Y}^{-1}x}{e^t\bar{Y}^{-1}x}$.
  Let $\hat{c} = \bar{Y}c$, and $\hat{A} = A\bar{Y}$.
- Compute
  $$\hat{d} = (I - B^t(BB^t)^{-1}B)\hat{c} \enspace ,$$
  where $B = \begin{pmatrix} \hat{A} \\ e^t \end{pmatrix}$.
- Set
  $$\hat{x}_{\text{new}} = \frac{e}{n} - \rho\frac{\hat{d}}{\|\hat{d}\|} \enspace ,$$
  with $\rho = \alpha r$ with $\alpha = 1/4$ and $r = 1/\sqrt{n(n-1)}$.
- Compute $\bar{x}_{\text{new}} = F_{\bar{x}}^{-1}(\hat{x}_{\text{new}})$.

Lemma 40

The new point $\hat{x}_{\mathrm{new}}$ in the transformed space is the point that minimizes the cost $\hat{c}^t\hat{x}$ among all feasible points in $B(\frac{e}{n},\rho)$.

Proof: Let $\hat{z}$ be another feasible point in $B(\frac{e}{n}, \rho)$.

Proof: Let $\hat{z}$ be another feasible point in $B(\frac{e}{n}, \rho)$.

As $\hat{A}\hat{z} = 0$, $\hat{A}\hat{x}_{\text{new}} = 0$, $e^t\hat{z} = 1$, $e^t\hat{x}_{\text{new}} = 1$

Proof: Let $\hat{z}$ be another feasible point in $B(\frac{e}{n}, \rho)$.

As $\hat{A}\hat{z} = 0$, $\hat{A}\hat{x}_{\text{new}} = 0$, $e^t\hat{z} = 1$, $e^t\hat{x}_{\text{new}} = 1$ we have

$$B(\hat{x}_{\text{new}} - \hat{z}) = 0 \quad .$$

Proof: Let $\hat{z}$ be another feasible point in $B(\frac{e}{n}, \rho)$.

As $\hat{A}\hat{z} = 0$, $\hat{A}\hat{x}_{\text{new}} = 0$, $e^t\hat{z} = 1$, $e^t\hat{x}_{\text{new}} = 1$ we have

$$B(\hat{x}_{\text{new}} - \hat{z}) = 0 \ .$$

Further,

$$(\hat{c} - \hat{d})^t$$

Proof: Let $\hat{z}$ be another feasible point in $B(\frac{e}{n}, \rho)$.

As $\hat{A}\hat{z} = 0$, $\hat{A}\hat{x}_{\text{new}} = 0$, $e^t\hat{z} = 1$, $e^t\hat{x}_{\text{new}} = 1$ we have

$$B(\hat{x}_{\text{new}} - \hat{z}) = 0 \ .$$

Further,

$$(\hat{c} - \hat{d})^t = (\hat{c} - P\hat{c})^t$$

Proof: Let $\hat{z}$ be another feasible point in $B(\frac{e}{n}, \rho)$.

As $\hat{A}\hat{z} = 0$, $\hat{A}\hat{x}_{\text{new}} = 0$, $e^t\hat{z} = 1$, $e^t\hat{x}_{\text{new}} = 1$ we have

$$B(\hat{x}_{\text{new}} - \hat{z}) = 0 \ .$$

Further,

$$\begin{aligned}(\hat{c} - \hat{d})^t &= (\hat{c} - P\hat{c})^t \\ &= (B^t(BB^t)^{-1}B\hat{c})^t\end{aligned}$$

Proof: Let $\hat{z}$ be another feasible point in $B(\frac{e}{n}, \rho)$.

As $\hat{A}\hat{z} = 0$, $\hat{A}\hat{x}_{\text{new}} = 0$, $e^t\hat{z} = 1$, $e^t\hat{x}_{\text{new}} = 1$ we have

$$B(\hat{x}_{\text{new}} - \hat{z}) = 0 \ .$$

Further,

$$\begin{aligned}(\hat{c} - \hat{d})^t &= (\hat{c} - P\hat{c})^t \\ &= (B^t(BB^t)^{-1}B\hat{c})^t \\ &= \hat{c}^t B^t (BB^t)^{-1} B\end{aligned}$$

Proof: Let $\hat{z}$ be another feasible point in $B(\frac{e}{n}, \rho)$.

As $\hat{A}\hat{z} = 0$, $\hat{A}\hat{x}_{\text{new}} = 0$, $e^t\hat{z} = 1$, $e^t\hat{x}_{\text{new}} = 1$ we have

$$B(\hat{x}_{\text{new}} - \hat{z}) = 0 \; .$$

Further,

$$\begin{aligned}(\hat{c} - \hat{d})^t &= (\hat{c} - P\hat{c})^t \\ &= (B^t(BB^t)^{-1}B\hat{c})^t \\ &= \hat{c}^t B^t(BB^t)^{-1}B\end{aligned}$$

Hence, we get

$$(\hat{c} - \hat{d})^t(\hat{x}_{\text{new}} - \hat{z}) = 0$$

Proof: Let $\hat{z}$ be another feasible point in $B(\frac{e}{n}, \rho)$.

As $\hat{A}\hat{z} = 0$, $\hat{A}\hat{x}_{\text{new}} = 0$, $e^t\hat{z} = 1$, $e^t\hat{x}_{\text{new}} = 1$ we have

$$B(\hat{x}_{\text{new}} - \hat{z}) = 0 \ .$$

Further,

$$\begin{aligned}(\hat{c} - \hat{d})^t &= (\hat{c} - P\hat{c})^t \\ &= (B^t(BB^t)^{-1}B\hat{c})^t \\ &= \hat{c}^t B^t(BB^t)^{-1}B\end{aligned}$$

Hence, we get

$$(\hat{c} - \hat{d})^t(\hat{x}_{\text{new}} - \hat{z}) = 0 \text{ or } \hat{c}^t(\hat{x}_{\text{new}} - \hat{z}) = \hat{d}^t(\hat{x}_{\text{new}} - \hat{z})$$

Proof: Let $\hat{z}$ be another feasible point in $B(\frac{e}{n}, \rho)$.

As $\hat{A}\hat{z} = 0$, $\hat{A}\hat{x}_{\text{new}} = 0$, $e^t\hat{z} = 1$, $e^t\hat{x}_{\text{new}} = 1$ we have

$$B(\hat{x}_{\text{new}} - \hat{z}) = 0 \ .$$

Further,

$$\begin{aligned}(\hat{c} - \hat{d})^t &= (\hat{c} - P\hat{c})^t \\ &= (B^t(BB^t)^{-1}B\hat{c})^t \\ &= \hat{c}^t B^t(BB^t)^{-1}B\end{aligned}$$

Hence, we get

$$(\hat{c} - \hat{d})^t(\hat{x}_{\text{new}} - \hat{z}) = 0 \text{ or } \hat{c}^t(\hat{x}_{\text{new}} - \hat{z}) = \hat{d}^t(\hat{x}_{\text{new}} - \hat{z})$$

which means that the cost-difference between $\hat{x}_{\text{new}}$ and $\hat{z}$ is the same measured w.r.t. the cost-vector $\hat{c}$ or the projected cost-vector $\hat{d}$.

But

$$\frac{\hat{d}^t}{\|\hat{d}\|}(\hat{x}_{\text{new}} - \hat{z})$$

But

$$\frac{\hat{d}^t}{\|\hat{d}\|}(\hat{x}_{\text{new}} - \hat{z}) = \frac{\hat{d}^t}{\|\hat{d}\|}\left(\frac{e}{n} - \rho\frac{\hat{d}}{\|\hat{d}\|} - \hat{z}\right)$$

But

$$\frac{\hat{d}^t}{\|\hat{d}\|}(\hat{x}_{\text{new}} - \hat{z}) = \frac{\hat{d}^t}{\|\hat{d}\|}\left(\frac{e}{n} - \rho\frac{\hat{d}}{\|\hat{d}\|} - \hat{z}\right) = \frac{\hat{d}^t}{\|\hat{d}\|}\left(\frac{e}{n} - \hat{z}\right) - \rho$$

But

$$\frac{\hat{d}^t}{\|\hat{d}\|}(\hat{x}_{\text{new}} - \hat{z}) = \frac{\hat{d}^t}{\|\hat{d}\|}\left(\frac{e}{n} - \rho\frac{\hat{d}}{\|\hat{d}\|} - \hat{z}\right) = \frac{\hat{d}^t}{\|\hat{d}\|}\left(\frac{e}{n} - \hat{z}\right) - \rho < 0$$

as $\frac{e}{n} - \hat{z}$ is a vector of length at most ρ.

But

$$\frac{\hat{d}^t}{\|\hat{d}\|}(\hat{x}_{\text{new}} - \hat{z}) = \frac{\hat{d}^t}{\|\hat{d}\|}\left(\frac{e}{n} - \rho\frac{\hat{d}}{\|\hat{d}\|} - \hat{z}\right) = \frac{\hat{d}^t}{\|\hat{d}\|}\left(\frac{e}{n} - \hat{z}\right) - \rho < 0$$

as $\frac{e}{n} - \hat{z}$ is a vector of length at most ρ.

This gives $\hat{d}(\hat{x}_{\text{new}} - \hat{z}) \leq 0$ and therefore $\hat{c}\hat{x}_{\text{new}} \leq \hat{c}\hat{z}$.

In order to measure the progress of the algorithm we introduce a potential function f:

$$f(x)$$

In order to measure the progress of the algorithm we introduce a potential function f:

$$f(x) = \sum_j \ln(\frac{c^t x}{x_j})$$

In order to measure the progress of the algorithm we introduce a potential function f:

$$f(x) = \sum_j \ln(\frac{c^t x}{x_j}) = n \ln(c^t x) - \sum_j \ln(x_j) \enspace .$$

In order to measure the progress of the algorithm we introduce a potential function f:

$$f(x) = \sum_j \ln(\frac{c^t x}{x_j}) = n \ln(c^t x) - \sum_j \ln(x_j) \enspace .$$

- The function f is invariant to scaling (i.e., $f(kx) = f(x)$).

In order to measure the progress of the algorithm we introduce a potential function f:

$$f(x) = \sum_j \ln(\frac{c^t x}{x_j}) = n\ln(c^t x) - \sum_j \ln(x_j) \enspace .$$

- The function f is invariant to scaling (i.e., $f(kx) = f(x)$).
- The potential function essentially measures cost (note the term $n\ln(c^t x)$) but it penalizes us for choosing x_j values very small (by the term $-\sum_j \ln(x_j)$; note that $-\ln(x_j)$ is always positive).

For a point $\hat{z}$ in the transformed space we use the potential function

$$\hat{f}(\hat{z})$$

For a point $\hat{z}$ in the transformed space we use the potential function

$$\hat{f}(\hat{z}) := f(F_{\bar{x}}^{-1}(\hat{z}))$$

For a point $\hat{z}$ in the transformed space we use the potential function

$$\hat{f}(\hat{z}) := f(F_{\bar{x}}^{-1}(\hat{z})) = f(\frac{\bar{Y}\hat{z}}{e^t\bar{Y}\hat{z}}) = f(\bar{Y}\hat{z})$$

For a point $\hat{z}$ in the transformed space we use the potential function

$$\hat{f}(\hat{z}) := f(F_{\bar{x}}^{-1}(\hat{z})) = f(\frac{\bar{Y}\hat{z}}{e^t\bar{Y}\hat{z}}) = f(\bar{Y}\hat{z})$$

$$= \sum_j \ln(\frac{c^t\bar{Y}\hat{z}}{\bar{x}_j\hat{z}_j})$$

For a point $\hat{z}$ in the transformed space we use the potential function

$$\hat{f}(\hat{z}) := f(F_{\bar{x}}^{-1}(\hat{z})) = f(\frac{\bar{Y}\hat{z}}{e^t\bar{Y}\hat{z}}) = f(\bar{Y}\hat{z})$$

$$= \sum_j \ln(\frac{c^t\bar{Y}\hat{z}}{\bar{x}_j\hat{z}_j}) = \sum_j \ln(\frac{\hat{c}^t\hat{z}}{\hat{z}_j}) - \sum_j \ln \bar{x}_j$$

For a point $\hat{z}$ in the transformed space we use the potential function

$$\hat{f}(\hat{z}) := f(F_{\bar{x}}^{-1}(\hat{z})) = f\Big(\frac{\bar{Y}\hat{z}}{e^t\bar{Y}\hat{z}}\Big) = f(\bar{Y}\hat{z})$$

$$= \sum_j \ln\Big(\frac{c^t\bar{Y}\hat{z}}{\bar{x}_j\hat{z}_j}\Big) = \sum_j \ln\Big(\frac{\hat{c}^t\hat{z}}{\hat{z}_j}\Big) - \sum_j \ln \bar{x}_j$$

Observation:

This means the potential of a point in the transformed space is simply the potential of its pre-image under F.

For a point $\hat{z}$ in the transformed space we use the potential function

$$\hat{f}(\hat{z}) := f(F_{\bar{x}}^{-1}(\hat{z})) = f(\frac{\bar{Y}\hat{z}}{e^t\bar{Y}\hat{z}}) = f(\bar{Y}\hat{z})$$
$$= \sum_j \ln(\frac{c^t\bar{Y}\hat{z}}{\bar{x}_j\hat{z}_j}) = \sum_j \ln(\frac{\hat{c}^t\hat{z}}{\hat{z}_j}) - \sum_j \ln \bar{x}_j$$

Observation:

This means the potential of a point in the transformed space is simply the potential of its pre-image under F.

Note that if we are interested in potential-change we can ignore the additive term above. Then f and $\hat{f}$ have the same form; only c is replaced by $\hat{c}$.

The basic idea is to show that one iteration of Karmarkar results in a constant decrease of $\hat{f}$. This means

$$\hat{f}(\hat{x}_{\text{new}}) \leq \hat{f}(\frac{e}{n}) - \delta \enspace ,$$

where δ is a constant.

The basic idea is to show that one iteration of Karmarkar results in a constant decrease of $\hat{f}$. This means

$$\hat{f}(\hat{x}_{\text{new}}) \le \hat{f}(\frac{e}{n}) - \delta \ ,$$

where δ is a constant.

This gives

$$f(\bar{x}_{\text{new}}) \le f(\bar{x}) - \delta \ .$$

Lemma 41

There is a feasible point z (i.e., $\hat{A}z = 0$) in $B(\frac{e}{n}, \rho) \cap \Delta$ that has

$$\hat{f}(z) \leq \hat{f}(\frac{e}{n}) - \delta$$

with $\delta = \ln(1 + \alpha)$.

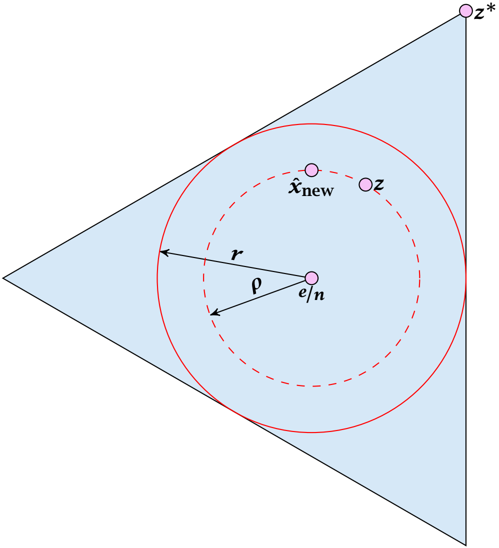
Lemma 41

There is a feasible point z (i.e., $\hat{A}z = 0$) in $B(\frac{e}{n}, \rho) \cap \Delta$ that has

$$\hat{f}(z) \leq \hat{f}(\frac{e}{n}) - \delta$$

with $\delta = \ln(1 + \alpha)$.

Note that this shows the existence of a good point within the ball. In general it will be difficult to find this point.


z^*
$\hat{x}_{\text{new}}$
z
r
ρ
e/n

Let z^* be the feasible point in the transformed space where $\hat{c}^t x$ is minimized. (Note that in contrast $\hat{x}_{\text{new}}$ is the point in the intersection of the feasible region and $B(\frac{e}{n}, \rho)$ that minimizes this function; in general $z^* \neq \hat{x}_{\text{new}}$)

Let z^* be the feasible point in the transformed space where $\hat{c}^t x$ is minimized. (Note that in contrast $\hat{x}_{\text{new}}$ is the point in the intersection of the feasible region and $B(\frac{e}{n}, \rho)$ that minimizes this function; in general $z^* \neq \hat{x}_{\text{new}}$)

z^* must lie at the boundary of the simplex. This means $z^* \notin B(\frac{e}{n}, \rho)$.

Let z^* be the feasible point in the transformed space where $\hat{c}^t x$ is minimized. (Note that in contrast $\hat{x}_{\text{new}}$ is the point in the intersection of the feasible region and $B(\frac{e}{n}, \rho)$ that minimizes this function; in general $z^* \neq \hat{x}_{\text{new}}$)

z^* must lie at the boundary of the simplex. This means $z^* \notin B(\frac{e}{n}, \rho)$.

The point z we want to use lies farthest in the direction from $\frac{e}{n}$ to z^*, namely

Let z^* be the feasible point in the transformed space where $\hat{c}^t x$ is minimized. (Note that in contrast $\hat{x}_{\text{new}}$ is the point in the intersection of the feasible region and $B(\frac{e}{n}, \rho)$ that minimizes this function; in general $z^* \neq \hat{x}_{\text{new}}$)

z^* must lie at the boundary of the simplex. This means $z^* \notin B(\frac{e}{n}, \rho)$.

The point z we want to use lies farthest in the direction from $\frac{e}{n}$ to z^*, namely

$$z = (1 - \lambda)\frac{e}{n} + \lambda z^*$$

for some positive $\lambda < 1$.

Hence,

$$\hat{c}^t z = (1-\lambda)\hat{c}^t \frac{e}{n} + \lambda \hat{c}^t z^*$$

Hence,

$$\hat{c}^t z = (1-\lambda)\hat{c}^t \frac{e}{n} + \lambda \hat{c}^t z^*$$

The optimum cost (at z^*) is zero.

Hence,

$$\hat{c}^t z = (1-\lambda)\hat{c}^t \frac{e}{n} + \lambda \hat{c}^t z^*$$

The optimum cost (at z^*) is zero.

Therefore,

$$\frac{\hat{c}^t \frac{e}{n}}{\hat{c}^t z} = \frac{1}{1-\lambda}$$

The improvement in the potential function is

$$\hat{f}(\frac{e}{n}) - \hat{f}(z)$$

The improvement in the potential function is

$$\hat{f}(\frac{e}{n}) - \hat{f}(z) = \sum_j \ln(\frac{\hat{c}^t \frac{e}{n}}{\frac{1}{n}}) - \sum_j \ln(\frac{\hat{c}^t z}{z_j})$$

The improvement in the potential function is

$$\begin{aligned}\hat{f}(\frac{e}{n}) - \hat{f}(z) &= \sum_j \ln(\frac{\hat{c}^t \frac{e}{n}}{\frac{1}{n}}) - \sum_j \ln(\frac{\hat{c}^t z}{z_j}) \\ &= \sum_j \ln(\frac{\hat{c}^t \frac{e}{n}}{\hat{c}^t z} \cdot \frac{z_j}{\frac{1}{n}})\end{aligned}$$

The improvement in the potential function is

$$\begin{aligned}
\hat{f}(\frac{e}{n}) - \hat{f}(z) &= \sum_j \ln(\frac{\hat{c}^t \frac{e}{n}}{\frac{1}{n}}) - \sum_j \ln(\frac{\hat{c}^t z}{z_j}) \\
&= \sum_j \ln(\frac{\hat{c}^t \frac{e}{n}}{\hat{c}^t z} \cdot \frac{z_j}{\frac{1}{n}}) \\
&= \sum_j \ln(\frac{n}{1-\lambda} z_j)
\end{aligned}$$

The improvement in the potential function is

$$
\begin{aligned}
\hat{f}(\frac{e}{n}) - \hat{f}(z) &= \sum_j \ln(\frac{\hat{c}^t \frac{e}{n}}{\frac{1}{n}}) - \sum_j \ln(\frac{\hat{c}^t z}{z_j}) \\
&= \sum_j \ln(\frac{\hat{c}^t \frac{e}{n}}{\hat{c}^t z} \cdot \frac{z_j}{\frac{1}{n}}) \\
&= \sum_j \ln(\frac{n}{1-\lambda} z_j) \\
&= \sum_j \ln(\frac{n}{1-\lambda}((1-\lambda)\frac{1}{n} + \lambda z_j^*))
\end{aligned}
$$

The improvement in the potential function is

$$\begin{aligned}
\hat{f}(\frac{e}{n}) - \hat{f}(z) &= \sum_j \ln(\frac{\hat{c}^t \frac{e}{n}}{\frac{1}{n}}) - \sum_j \ln(\frac{\hat{c}^t z}{z_j}) \\
&= \sum_j \ln(\frac{\hat{c}^t \frac{e}{n}}{\hat{c}^t z} \cdot \frac{z_j}{\frac{1}{n}}) \\
&= \sum_j \ln(\frac{n}{1-\lambda} z_j) \\
&= \sum_j \ln(\frac{n}{1-\lambda}((1-\lambda)\frac{1}{n} + \lambda z_j^*)) \\
&= \sum_j \ln(1 + \frac{n\lambda}{1-\lambda} z_j^*)
\end{aligned}$$

We can use the fact that for non-negative s_i

$$\sum_i \ln(1 + s_i) \geq \ln(1 + \sum_i s_i)$$

We can use the fact that for non-negative s_i

$$\sum_i \ln(1+s_i) \geq \ln(1+\sum_i s_i)$$

This gives

$$\hat{f}(\frac{e}{n}) - \hat{f}(z)$$

We can use the fact that for non-negative s_i

$$\sum_i \ln(1 + s_i) \geq \ln(1 + \sum_i s_i)$$

This gives

$$\hat{f}(\frac{e}{n}) - \hat{f}(z) = \sum_j \ln(1 + \frac{n\lambda}{1-\lambda} z_j^*)$$

We can use the fact that for non-negative s_i

$$\sum_i \ln(1 + s_i) \geq \ln(1 + \sum_i s_i)$$

This gives

$$\begin{aligned}\hat{f}(\frac{e}{n}) - \hat{f}(z) &= \sum_j \ln(1 + \frac{n\lambda}{1-\lambda} z_j^*) \\ &\geq \ln(1 + \frac{n\lambda}{1-\lambda})\end{aligned}$$

In order to get further we need a bound on λ:

$$\alpha r$$

In order to get further we need a bound on λ:

$$\alpha r = \rho$$

In order to get further we need a bound on λ:

$$\alpha r = \rho = \|z - {}^{e}/_{n}\|$$

In order to get further we need a bound on λ:

$$\alpha r = \rho = \|z - {}^{e}\!/\!{}_{n}\| = \|\lambda(z^* - {}^{e}\!/\!{}_{n})\|$$

In order to get further we need a bound on λ:

$$\alpha r = \rho = \|z - e/n\| = \|\lambda(z^* - e/n)\| \leq \lambda R$$

In order to get further we need a bound on λ:

$$\alpha r = \rho = \|z - e/n\| = \|\lambda(z^* - e/n)\| \leq \lambda R$$

In order to get further we need a bound on λ:

$$\alpha r = \rho = \|z - e/n\| = \|\lambda(z^* - e/n)\| \leq \lambda R$$

Here R is the radius of the ball around $\frac{e}{n}$ that contains the whole simplex.

In order to get further we need a bound on λ:

$$\alpha r = \rho = \|z - e/n\| = \|\lambda(z^* - e/n)\| \le \lambda R$$

Here R is the radius of the ball around $\frac{e}{n}$ that contains the whole simplex.

$R = \sqrt{(n-1)/n}$.

In order to get further we need a bound on λ:

$$\alpha r = \rho = \|z - {}^{e}/_{n}\| = \|\lambda(z^* - {}^{e}/_{n})\| \le \lambda R$$

Here R is the radius of the ball around $\frac{e}{n}$ that contains the whole simplex.

$R = \sqrt{(n-1)/n}$. Since $r = {}^{1}/_{\sqrt{(n-1)n}}$ we have $R/r = n - 1$ and

$$\lambda \ge \alpha \frac{r}{R} \ge \alpha/(n-1)$$

In order to get further we need a bound on λ:

$$\alpha r = \rho = \|z - e/n\| = \|\lambda(z^* - e/n)\| \leq \lambda R$$

Here R is the radius of the ball around $\frac{e}{n}$ that contains the whole simplex.

$R = \sqrt{(n-1)/n}$. Since $r = 1/\sqrt{(n-1)n}$ we have $R/r = n - 1$ and

$$\lambda \geq \alpha \frac{r}{R} \geq \alpha/(n-1)$$

Then

$$1 + n\frac{\lambda}{1-\lambda}$$

In order to get further we need a bound on λ:

$$\alpha r = \rho = \|z - e/n\| = \|\lambda(z^* - e/n)\| \le \lambda R$$

Here R is the radius of the ball around $\frac{e}{n}$ that contains the whole simplex.

$R = \sqrt{(n-1)/n}$. Since $r = 1/\sqrt{(n-1)n}$ we have $R/r = n - 1$ and

$$\lambda \ge \alpha \frac{r}{R} \ge \alpha/(n-1)$$

Then

$$1 + n\frac{\lambda}{1-\lambda} \ge 1 + \frac{n\alpha}{n - \alpha - 1}$$

In order to get further we need a bound on λ:

$$\alpha r = \rho = \|z - {}^{e}/_{n}\| = \|\lambda(z^* - {}^{e}/_{n})\| \leq \lambda R$$

Here R is the radius of the ball around $\frac{e}{n}$ that contains the whole simplex.

$R = \sqrt{(n-1)/n}$. Since $r = {}^{1}/\sqrt{(n-1)n}$ we have $R/r = n - 1$ and

$$\lambda \geq \alpha \frac{r}{R} \geq \alpha/(n-1)$$

Then

$$1 + n\frac{\lambda}{1-\lambda} \geq 1 + \frac{n\alpha}{n - \alpha - 1} \geq 1 + \alpha$$

In order to get further we need a bound on λ:

$$\alpha r = \rho = \|z - e/n\| = \|\lambda(z^* - e/n)\| \le \lambda R$$

Here R is the radius of the ball around $\frac{e}{n}$ that contains the whole simplex.

$R = \sqrt{(n-1)/n}$. Since $r = 1/\sqrt{(n-1)n}$ we have $R/r = n - 1$ and

$$\lambda \ge \alpha \frac{r}{R} \ge \alpha/(n-1)$$

Then

$$1 + n\frac{\lambda}{1-\lambda} \ge 1 + \frac{n\alpha}{n-\alpha-1} \ge 1 + \alpha$$

This gives the lemma.

Lemma 42

If we choose $\alpha = 1/4$ and $n \geq 4$ in Karmarkars algorithm the point $\hat{x}_{\text{new}}$ satisfies

$$\hat{f}(\hat{x}_{\text{new}}) \leq \hat{f}(\frac{e}{n}) - \delta$$

with $\delta = 1/10$.

Proof:

Proof:

Define

$$g(\hat{x}) =$$

Proof:

Define

$$g(\hat{x}) = n \ln \frac{\hat{c}^t \hat{x}}{\hat{c}^t \frac{e}{n}}$$

Proof:

Define

$$g(\hat{x}) = n \ln \frac{\hat{c}^t \hat{x}}{\hat{c}^t \frac{e}{n}} = n(\ln \hat{c}^t \hat{x} - \ln \hat{c}^t \frac{e}{n}) \ .$$

Proof:

Define

$$g(\hat{x}) = n \ln \frac{\hat{c}^t \hat{x}}{\hat{c}^t \frac{e}{n}} = n(\ln \hat{c}^t \hat{x} - \ln \hat{c}^t \frac{e}{n}) \enspace .$$

This is the change in the cost part of the potential function when going from the center $\frac{e}{n}$ to the point $\hat{x}$ in the transformed space.

Similar, the penalty when going from $\frac{e}{n}$ to w increases by

$$h(\hat{x}) = \text{pen}(\hat{x}) - \text{pen}(\frac{e}{n}) = -\sum_j \ln \frac{\hat{x}_j}{\frac{1}{n}}$$

where $\text{pen}(v) = -\sum_j \ln(v_j)$.

We want to derive a lower bound on

$$\hat{f}(\frac{e}{n}) - \hat{f}(\hat{x}_{\text{new}})$$

We want to derive a lower bound on

$$\begin{aligned}\hat{f}(\frac{e}{n}) - \hat{f}(\hat{x}_{\text{new}}) = {} & [\hat{f}(\frac{e}{n}) - \hat{f}(z)] \\ & + h(z) \\ & - h(\hat{x}_{\text{new}}) \\ & + [g(z) - g(\hat{x}_{\text{new}})]\end{aligned}$$

where z is the point in the ball where $\hat{f}$ achieves its minimum.

We want to derive a lower bound on

$$\begin{aligned}\hat{f}(\frac{e}{n}) - \hat{f}(\hat{x}_{\text{new}}) = {} & [\hat{f}(\frac{e}{n}) - \hat{f}(z)] \\ & + h(z) \\ & - h(\hat{x}_{\text{new}}) \\ & + [g(z) - g(\hat{x}_{\text{new}})]\end{aligned}$$

where z is the point in the ball where $\hat{f}$ achieves its minimum.

We have

$$[\hat{f}(\frac{e}{n}) - \hat{f}(z)] \geq \ln(1+\alpha)$$

by the previous lemma.

We have

$$[\hat{f}(\frac{e}{n}) - \hat{f}(z)] \geq \ln(1+\alpha)$$

by the previous lemma.

We have

$$[g(z) - g(\hat{x}_{\text{new}})] \geq 0$$

since $\hat{x}_{\text{new}}$ is the point with minimum cost in the ball, and g is monotonically increasing with cost.

We show that the change $h(w)$ in penalty when going from e/n to w fulfills

$$|h(w)| \leq \frac{\beta^2}{2(1-\beta)}$$

where $\beta = n\alpha r$ and w is some point in the ball $B(\frac{e}{n}, \alpha r)$.

We show that the change $h(w)$ in penalty when going from e/n to w fulfills

$$|h(w)| \le \frac{\beta^2}{2(1-\beta)}$$

where $\beta = n\alpha r$ and w is some point in the ball $B(\frac{e}{n}, \alpha r)$.

Hence,

$$\hat{f}(\frac{e}{n}) - \hat{f}(\hat{x}_{\text{new}}) \ge \ln(1+\alpha) - \frac{\beta^2}{(1-\beta)} \quad .$$

Lemma 43

For $|x| \le \beta < 1$

$$|\ln(1+x) - x| \le \frac{x^2}{2(1-\beta)} \enspace .$$

This gives for $w \in B(\frac{e}{n}, \rho)$

$$|h(w)|$$

This gives for $w \in B(\frac{e}{n}, \rho)$

$$|h(w)| = \left| \sum_j \ln \frac{w_j}{1/n} \right|$$

This gives for $w \in B(\frac{e}{n}, \rho)$

$$
\begin{aligned}
|h(w)| &= \left| \sum_j \ln \frac{w_j}{1/n} \right| \\
&= \left| \sum_j \ln\left(\frac{1/n + (w_j - 1/n)}{1/n} \right) - \sum_j n\Big(w_j - \frac{1}{n}\Big) \right|
\end{aligned}
$$

This gives for $w \in B(\frac{e}{n}, \rho)$

$$|h(w)| = \left| \sum_j \ln \frac{w_j}{1/n} \right|$$

$$= \left| \sum_j \ln \left(\frac{1/n + (w_j - 1/n)}{1/n} \right) - \sum_j n \left(w_j - \frac{1}{n} \right) \right|$$

$= 0$

This gives for $w \in B(\frac{e}{n}, \rho)$

$$\begin{aligned} |h(w)| &= \left| \sum_j \ln \frac{w_j}{1/n} \right| \\ &= \left| \sum_j \ln \left(\frac{1/n + (w_j - 1/n)}{1/n} \right) - \sum_j n \left(w_j - \frac{1}{n} \right) \right| \\ &= \left| \sum_j \left[\ln \Big(1 + n(w_j - 1/n) \Big) - n(w_j - 1/n) \right] \right| \end{aligned}$$

This gives for $w \in B(\frac{e}{n}, \rho)$

$$|h(w)| = \left| \sum_j \ln \frac{w_j}{1/n} \right|$$

$$= \left| \sum_j \ln\left(\frac{1/n + (w_j - 1/n)}{1/n}\right) - \sum_j n\Big(w_j - \frac{1}{n}\Big) \right|$$

$$= \left| \sum_j \left[\ln\Big(1 + n(\underbrace{w_j - 1/n}_{\le \alpha r})\Big) - n(w_j - 1/n) \right] \right|$$

This gives for $w \in B(\frac{e}{n}, \rho)$

$$
\begin{aligned}
|h(w)| &= \left| \sum_j \ln \frac{w_j}{1/n} \right| \\
&= \left| \sum_j \ln\left(\frac{1/n + (w_j - 1/n)}{1/n} \right) - \sum_j n\left(w_j - \frac{1}{n} \right) \right| \\
&= \left| \sum_j \left[\ln\left(1 + \underbrace{n(w_j - 1/n)}_{\leq n\alpha r < 1} \right) - n(w_j - 1/n) \right] \right|
\end{aligned}
$$

This gives for $w \in B(\frac{e}{n}, \rho)$

$$
\begin{aligned}
|h(w)| &= \left| \sum_j \ln \frac{w_j}{1/n} \right| \\
&= \left| \sum_j \ln \left(\frac{1/n + (w_j - 1/n)}{1/n} \right) - \sum_j n\left(w_j - \frac{1}{n} \right) \right| \\
&= \left| \sum_j \left[\ln \left(1 + \underbrace{n(w_j - 1/n)}_{x} \right) - \underbrace{n(w_j - 1/n)}_{x} \right] \right|
\end{aligned}
$$

This gives for $w \in B(\frac{e}{n}, \rho)$

$$\begin{aligned}
|h(w)| &= \left|\sum_j \ln \frac{w_j}{1/n}\right| \\
&= \left|\sum_j \ln\left(\frac{1/n + (w_j - 1/n)}{1/n}\right) - \sum_j n\Big(w_j - \frac{1}{n}\Big)\right| \\
&= \left|\sum_j \Big[\ln\Big(1 + n(w_j - 1/n)\Big) - n(w_j - 1/n)\Big]\right| \\
&\le \sum_j \frac{n^2(w_j - 1/n)^2}{2(1 - \alpha n r)}
\end{aligned}$$

This gives for $w \in B(\frac{e}{n}, \rho)$

$$
\begin{aligned}
|h(w)| &= \left| \sum_j \ln \frac{w_j}{1/n} \right| \\
&= \left| \sum_j \ln\left(\frac{1/n + (w_j - 1/n)}{1/n}\right) - \sum_j n\left(w_j - \frac{1}{n}\right) \right| \\
&= \left| \sum_j \left[\ln\Big(1 + n(w_j - 1/n)\Big) - n(w_j - 1/n) \right] \right| \\
&\le \sum_j \frac{n^2(w_j - 1/n)^2}{2(1 - \alpha n r)} \\
&\le \frac{(\alpha n r)^2}{2(1 - \alpha n r)}
\end{aligned}
$$

The decrease in potential is therefore at least

$$\ln(1+\alpha) - \frac{\beta^2}{1-\beta}$$

with $\beta = n\alpha r = \alpha\sqrt{\frac{n}{n-1}}$.

It can be shown that this is at least $\frac{1}{10}$ for $n \geq 4$ and $\alpha = 1/4$.

The decrease in potential is therefore at least

$$\ln(1+\alpha) - \frac{\beta^2}{1-\beta}$$

with $\beta = n\alpha r = \alpha\sqrt{\frac{n}{n-1}}$.

It can be shown that this is at least $\frac{1}{10}$ for $n \geq 4$ and $\alpha = 1/4$.

Let $\bar{x}^{(k)}$ be the current point after the k-th iteration, and let $\bar{x}^{(0)} = \frac{e}{n}$.

Then $f(\bar{x}^{(k)}) \leq f(e/n) - k/10$.
This gives

$$n \ln \frac{c^t \bar{x}^{(k)}}{c^t \frac{e}{n}} \leq \sum_j \ln \bar{x}_j^{(k)} - \sum_j \ln \frac{1}{n} - k/10$$
$$\leq n \ln n - k/10$$

Choosing $k = 10n(\ell + \ln n)$ with $\ell = \Theta(L)$ we get

$$\frac{c^t \bar{x}^{(k)}}{c^t \frac{e}{n}} \leq e^{-\ell} \leq 2^{-\ell} .$$

Hence, $\Theta(nL)$ iterations are sufficient. One iteration can be performed in time $\mathcal{O}(n^3)$.

Let $\bar{x}^{(k)}$ be the current point after the k-th iteration, and let $\bar{x}^{(0)} = \frac{e}{n}$.

Then $f(\bar{x}^{(k)}) \leq f(e/n) - k/10$.

This gives

$$
\begin{aligned}
n \ln \frac{c^t \bar{x}^{(k)}}{c^t \frac{e}{n}} &\leq \sum_j \ln \bar{x}_j^{(k)} - \sum_j \ln \frac{1}{n} - k/10 \\
&\leq n \ln n - k/10
\end{aligned}
$$

Choosing $k = 10n(\ell + \ln n)$ with $\ell = \Theta(L)$ we get

$$
\frac{c^t \bar{x}^{(k)}}{c^t \frac{e}{n}} \leq e^{-\ell} \leq 2^{-\ell} .
$$

Hence, $\Theta(nL)$ iterations are sufficient. One iteration can be performed in time $\mathcal{O}(n^3)$.

Let $\bar{x}^{(k)}$ be the current point after the k-th iteration, and let $\bar{x}^{(0)} = \frac{e}{n}$.

Then $f(\bar{x}^{(k)}) \leq f(e/n) - k/10$.
This gives

$$n \ln \frac{c^t \bar{x}^{(k)}}{c^t \frac{e}{n}} \leq \sum_j \ln \bar{x}_j^{(k)} - \sum_j \ln \frac{1}{n} - k/10$$
$$\leq n \ln n - k/10$$

Choosing $k = 10n(\ell + \ln n)$ with $\ell = \Theta(L)$ we get

$$\frac{c^t \bar{x}^{(k)}}{c^t \frac{e}{n}} \leq e^{-\ell} \leq 2^{-\ell} \ .$$

Hence, $\Theta(nL)$ iterations are sufficient. One iteration can be performed in time $\mathcal{O}(n^3)$.

Let $\bar{x}^{(k)}$ be the current point after the k-th iteration, and let $\bar{x}^{(0)} = \frac{e}{n}$.

Then $f(\bar{x}^{(k)}) \le f(e/n) - k/10$.
This gives

$$n \ln \frac{c^t \bar{x}^{(k)}}{c^t \frac{e}{n}} \le \sum_j \ln \bar{x}_j^{(k)} - \sum_j \ln \frac{1}{n} - k/10$$

$$\le n \ln n - k/10$$

Choosing $k = 10n(\ell + \ln n)$ with $\ell = \Theta(L)$ we get

$$\frac{c^t \bar{x}^{(k)}}{c^t \frac{e}{n}} \le e^{-\ell} \le 2^{-\ell} \ .$$

Hence, $\Theta(nL)$ iterations are sufficient. One iteration can be performed in time $\mathcal{O}(n^3)$.

Let $\bar{x}^{(k)}$ be the current point after the k-th iteration, and let $\bar{x}^{(0)} = \frac{e}{n}$.

Then $f(\bar{x}^{(k)}) \leq f(e/n) - k/10$.
This gives

$$\begin{aligned} n \ln \frac{c^t \bar{x}^{(k)}}{c^t \frac{e}{n}} &\leq \sum_j \ln \bar{x}_j^{(k)} - \sum_j \ln \frac{1}{n} - k/10 \\ &\leq n \ln n - k/10 \end{aligned}$$

Choosing $k = 10n(\ell + \ln n)$ with $\ell = \Theta(L)$ we get

$$\frac{c^t \bar{x}^{(k)}}{c^t \frac{e}{n}} \leq e^{-\ell} \leq 2^{-\ell} \ .$$

Hence, $\Theta(nL)$ iterations are sufficient. One iteration can be performed in time $\mathcal{O}(n^3)$.

Let $\bar{x}^{(k)}$ be the current point after the k-th iteration, and let $\bar{x}^{(0)} = \frac{e}{n}$.

Then $f(\bar{x}^{(k)}) \leq f(e/n) - k/10$.
This gives

$$n \ln \frac{c^t \bar{x}^{(k)}}{c^t \frac{e}{n}} \leq \sum_j \ln \bar{x}_j^{(k)} - \sum_j \ln \frac{1}{n} - k/10$$
$$\leq n \ln n - k/10$$

Choosing $k = 10n(\ell + \ln n)$ with $\ell = \Theta(L)$ we get

$$\frac{c^t \bar{x}^{(k)}}{c^t \frac{e}{n}} \leq e^{-\ell} \leq 2^{-\ell} \;.$$

Hence, $\Theta(nL)$ iterations are sufficient. One iteration can be performed in time $\mathcal{O}(n^3)$.

Let $\bar{x}^{(k)}$ be the current point after the k-th iteration, and let $\bar{x}^{(0)} = \frac{e}{n}$.

Then $f(\bar{x}^{(k)}) \leq f(e/n) - k/10$.
This gives

$$\begin{aligned} n \ln \frac{c^t \bar{x}^{(k)}}{c^t \frac{e}{n}} &\leq \sum_j \ln \bar{x}_j^{(k)} - \sum_j \ln \frac{1}{n} - k/10 \\ &\leq n \ln n - k/10 \end{aligned}$$

Choosing $k = 10n(\ell + \ln n)$ with $\ell = \Theta(L)$ we get

$$\frac{c^t \bar{x}^{(k)}}{c^t \frac{e}{n}} \leq e^{-\ell} \leq 2^{-\ell} \quad .$$

Hence, $\Theta(nL)$ iterations are sufficient. One iteration can be performed in time $\mathcal{O}(n^3)$.

Part III

Approximation Algorithms

There are many practically important optimization problems that are NP-hard.

What can we do?

- Heuristics.
- Exploit special structure of instances occurring in practice.
- Consider algorithms that do not compute the optimal solution but provide solutions that are close to optimum.

There are many practically important optimization problems that are NP-hard.

What can we do?

- Heuristics.
- Exploit special structure of instances occurring in practise.
- Consider algorithms that do not compute the optimal solution but provide solutions that are close to optimum.

There are many practically important optimization problems that are NP-hard.

What can we do?

- Heuristics.
- Exploit special structure of instances occurring in practise.
- Consider algorithms that do not compute the optimal solution but provide solutions that are close to optimum.

There are many practically important optimization problems that are NP-hard.

What can we do?

- Heuristics.
- Exploit special structure of instances occurring in practise.
- Consider algorithms that do not compute the optimal solution but provide solutions that are close to optimum.

There are many practically important optimization problems that are NP-hard.

What can we do?

- Heuristics.
- Exploit special structure of instances occurring in practise.
- Consider algorithms that do not compute the optimal solution but provide solutions that are close to optimum.

Definition 44

An α-approximation for an optimization problem is a polynomial-time algorithm that for all instances of the problem produces a solution whose value is within a factor of α of the value of an optimal solution.

Why approximation algorithms?

- We need algorithms for hard problems.
- It gives a rigorous mathematical base for studying heuristics.
- It provides a metric to compare the difficulty of various optimization problems.
- Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

Why not?

- Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

Why approximation algorithms?

- We need algorithms for hard problems.
- It gives a rigorous mathematical base for studying heuristics.
- It provides a metric to compare the difficulty of various optimization problems.
- Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

Why not?

- Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

Why approximation algorithms?

- We need algorithms for hard problems.
- It gives a rigorous mathematical base for studying heuristics.
- It provides a metric to compare the difficulty of various optimization problems.
- Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

Why not?

- Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

Why approximation algorithms?

- We need algorithms for hard problems.
- It gives a rigorous mathematical base for studying heuristics.
- It provides a metric to compare the difficulty of various optimization problems.
- Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

Why not?

- Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

Why approximation algorithms?

- We need algorithms for hard problems.
- It gives a rigorous mathematical base for studying heuristics.
- It provides a metric to compare the difficulty of various optimization problems.
- Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

Why not?

- Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

Why approximation algorithms?

- We need algorithms for hard problems.
- It gives a rigorous mathematical base for studying heuristics.
- It provides a metric to compare the difficulty of various optimization problems.
- Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

Why not?

- Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

Why approximation algorithms?

- We need algorithms for hard problems.
- It gives a rigorous mathematical base for studying heuristics.
- It provides a metric to compare the difficulty of various optimization problems.
- Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

Why not?

- Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

Definition 45

An optimization problem $P = (\mathcal{I}, \mathrm{sol}, m, \mathrm{goal})$ is in **NPO** if

- $x \in \mathcal{I}$ can be decided in polynomial time
- $y \in \mathrm{sol}(\mathcal{I})$ can be verified in polynomial time
- m can be computed in polynomial time
- $\mathrm{goal} \in \{\min, \max\}$

In other words: the decision problem is there a solution y with $m(x, y)$ at most/at least z is in NP.

- x is problem instance
- y is candidate solution
- $m^*(x)$ cost/profit of an optimal solution

Definition 46 (Performance Ratio)

$$R(x,y) := \max\left\{\frac{m(x,y)}{m^*(x)}, \frac{m^*(x)}{m(x,y)}\right\}$$

Definition 47 (r-approximation)

An algorithm A is an r-approximation algorithm iff

$$\forall x \in \mathcal{I} : R(x, A(x)) \leq r \ ,$$

and A runs in polynomial time.

Definition 48 (PTAS)

A PTAS for a problem P from NPO is an algorithm that takes as input $x \in \mathcal{I}$ and $\epsilon > 0$ and produces a solution y for x with

$$R(x, y) \leq 1 + \epsilon \ .$$

The running time is polynomial in $|x|$.

approximation with arbitrary good factor... fast?

Problems that have a PTAS

Scheduling. Given m jobs with known processing times; schedule the jobs on n machines such that the MAKESPAN is minimized.

Definition 49 (FPTAS)
An FPTAS for a problem P from NPO is an algorithm that takes as input $x \in \mathcal{I}$ and $\epsilon > 0$ and produces a solution y for x with

$$R(x, y) \leq 1 + \epsilon \ .$$

The running time is polynomial in $|x|$ and $1/\epsilon$.

approximation with arbitrary good factor... fast!

Problems that have an FPTAS

KNAPSACK. Given a set of items with profits and weights choose a subset of total weight at most W s.t. the profit is maximized.

Definition 50 (APX – approximable)

A problem P from NPO is in APX if there exist a constant $r \geq 1$ and an r-approximation algorithm for P.

constant factor approximation...

Problems that are in APX

MAXCUT. Given a graph $G = (V, E)$; partition V into two disjoint pieces A and B s. t. the number of edges between both pieces is maximized.

MAX-3SAT. Given a 3CNF-formula. Find an assignment to the variables that satisfies the maximum number of clauses.

Problems with polylogarithmic approximation guarantees

- Set Cover
- Minimum Multicut
- Sparsest Cut
- Minimum Bisection

There is an r-approximation with $r \leq \mathcal{O}(\log^c(|x|))$ for some constant c.

Note that only for some of the above problem a matching lower bound is known.

There are really difficult problems!

Theorem 51

For any constant $\epsilon > 0$ there does not exist an $\Omega(n^{1-\epsilon})$-approximation algorithm for the maximum clique problem on a given graph G with n nodes unless $\mathrm{P} = \mathrm{NP}$.

Note that an n-approximation is trivial.

There are really difficult problems!

Theorem 51
For any constant $\epsilon > 0$ there does not exist an $\Omega(n^{1-\epsilon})$-approximation algorithm for the maximum clique problem on a given graph G with n nodes unless $\mathrm{P} = \mathrm{NP}$.

Note that an n-approximation is trivial.

There are really difficult problems!

Theorem 51

For any constant $\epsilon > 0$ there does not exist an $\Omega(n^{1-\epsilon})$-approximation algorithm for the maximum clique problem on a given graph G with n nodes unless $\mathrm{P} = \mathrm{NP}$.

Note that an n-approximation is trivial.

There are weird problems!

Asymmetric k-Center admits an $\mathcal{O}(\log^* n)$-approximation.

There is no $o(\log^* n)$-approximation to Asymmetric k-Center unless $NP \subseteq DTIME(n^{\log\log\log n})$.

Class APX not important in practise.

Instead of saying problem P is in APX one says problem P admits a 4-approximation.

One only says that a problem is APX-hard.

A crucial ingredient for the design and analysis of approximation algorithms is a technique to obtain an upper bound (for maximization problems) or a lower bound (for minimization problems).

Therefore Linear Programs or Integer Linear Programs play a vital role in the design of many approximation algorithms.

A crucial ingredient for the design and analysis of approximation algorithms is a technique to obtain an upper bound (for maximization problems) or a lower bound (for minimization problems).

Therefore Linear Programs or Integer Linear Programs play a vital role in the design of many approximation algorithms.

Definition 52
An Integer Linear Program or Integer Program is a Linear Program in which all variables are required to be integral.

Definition 53
A Mixed Integer Program is a Linear Program in which a subset of the variables are required to be integral.

Definition 52

An Integer Linear Program or Integer Program is a Linear Program in which all variables are required to be integral.

Definition 53

A Mixed Integer Program is a Linear Program in which a subset of the variables are required to be integral.

Many important combinatorial optimization problems can be formulated in the form of an Integer Program.

Note that solving Integer Programs in general is NP-complete!

Many important combinatorial optimization problems can be formulated in the form of an Integer Program.

Note that solving Integer Programs in general is NP-complete!

Set Cover

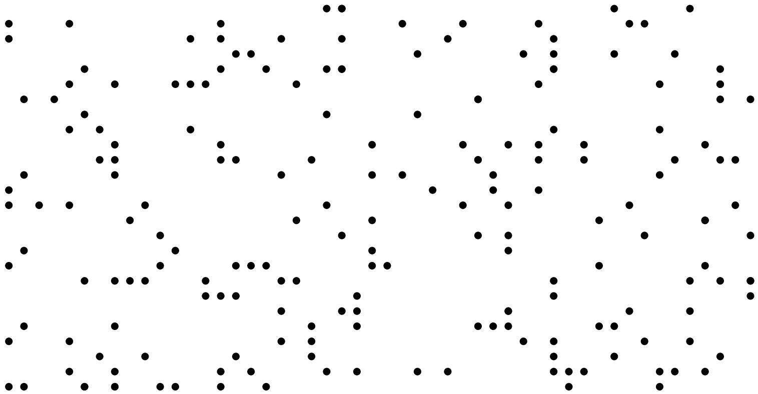
Given a ground set U, a collection of subsets $S_1, \dots, S_k \subseteq U$, where the i-th subset S_i has weight/cost w_i. Find a collection $I \subseteq \{1, \dots, k\}$ such that

$$\forall u \in U \exists i \in I : \; u \in S_i \quad \text{(every element is covered)}$$

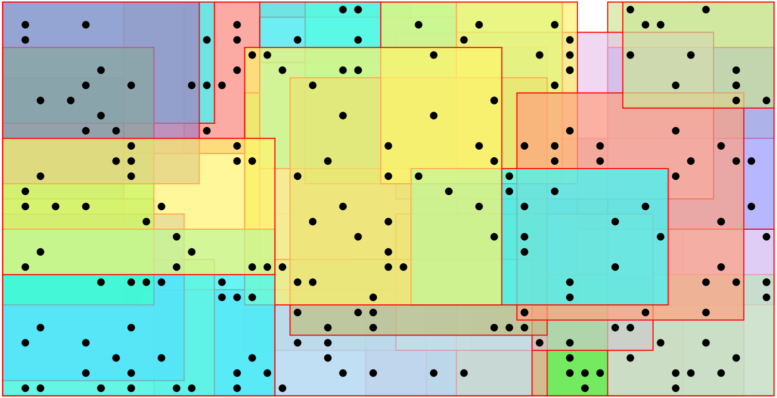
and

$$\sum_{i \in I} w_i \quad \text{is minimized.}$$

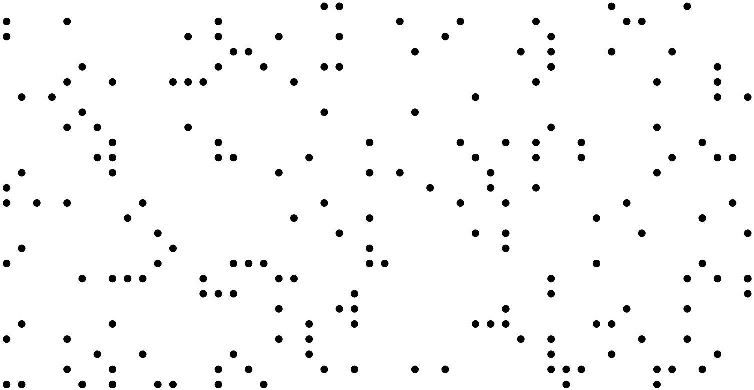
Set Cover



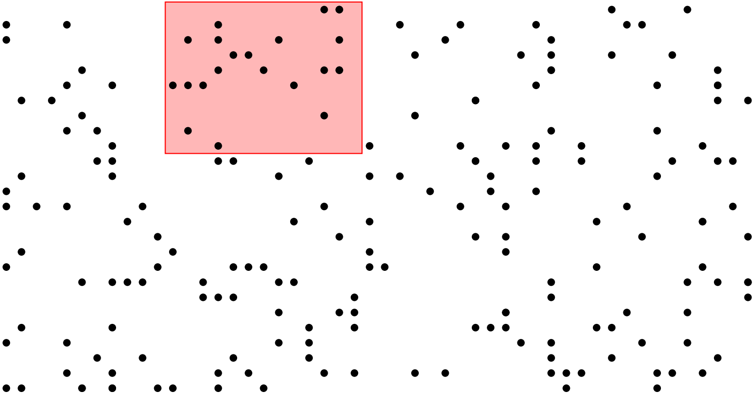
Set Cover



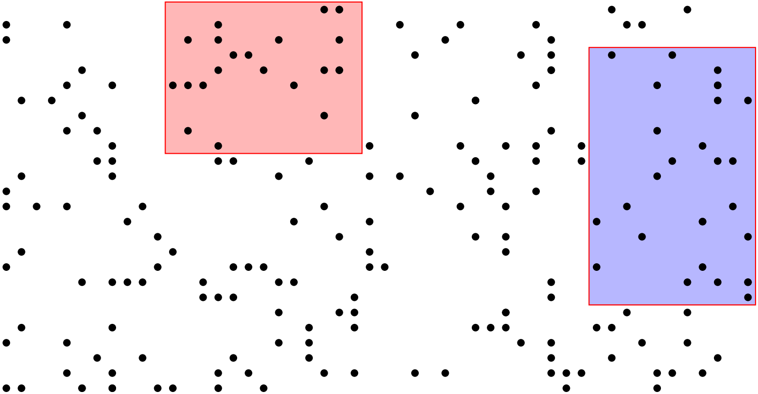
Set Cover



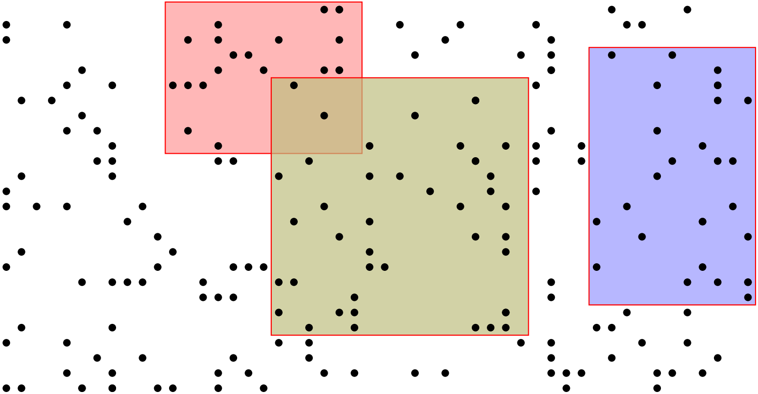
Set Cover



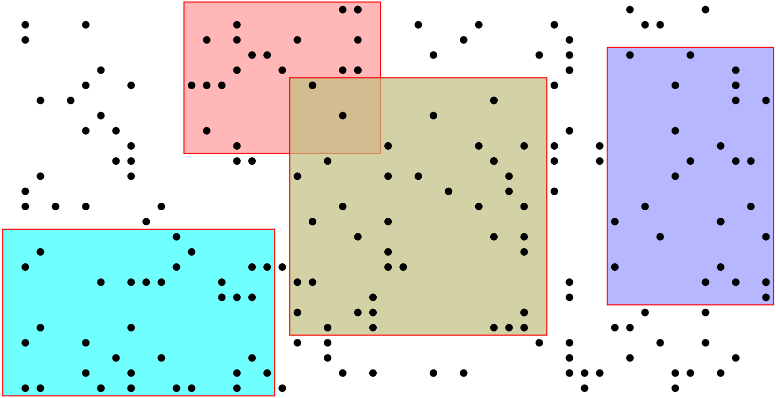
Set Cover



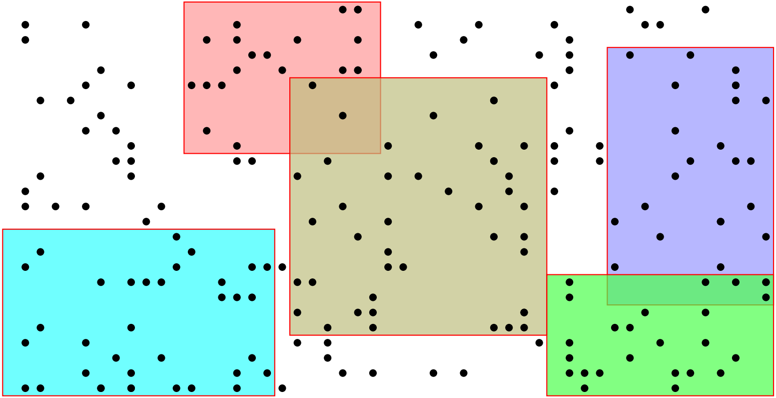
Set Cover



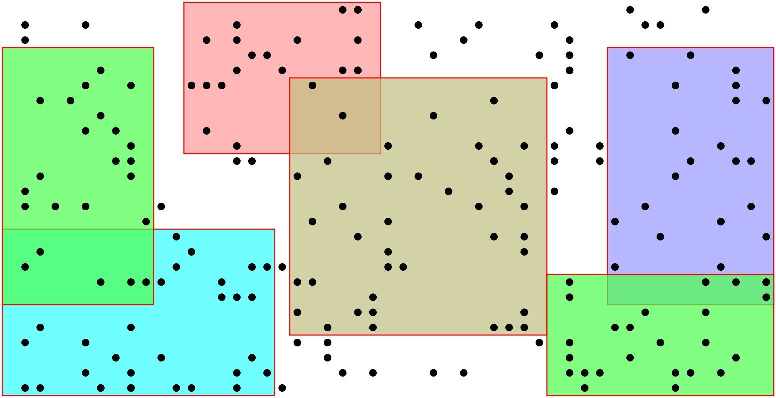
Set Cover



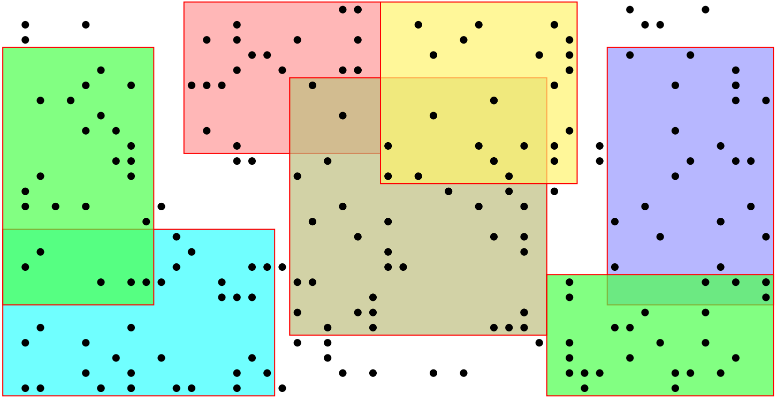
Set Cover



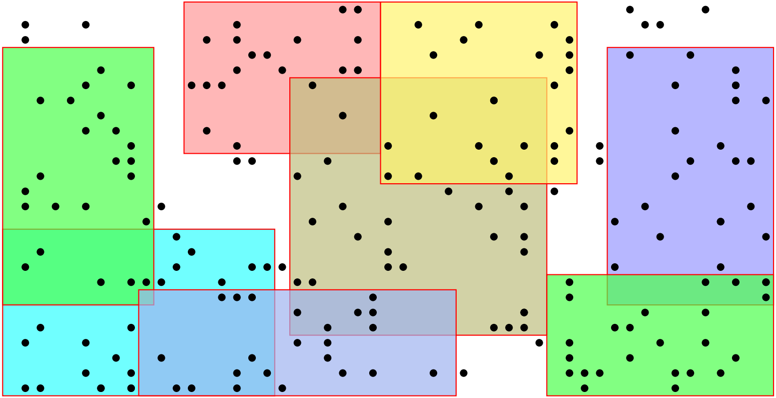
Set Cover



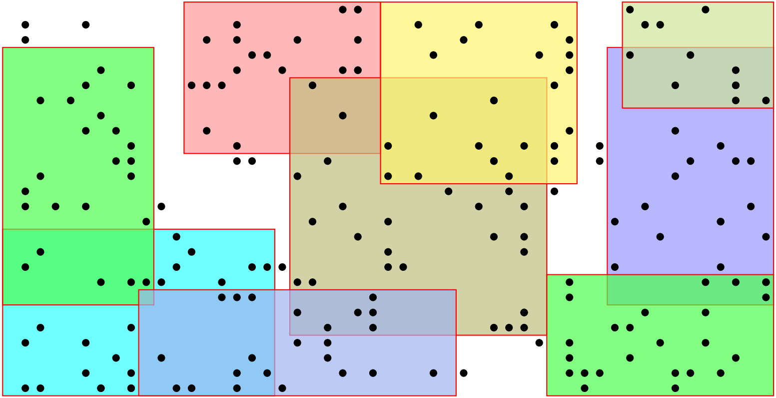
Set Cover



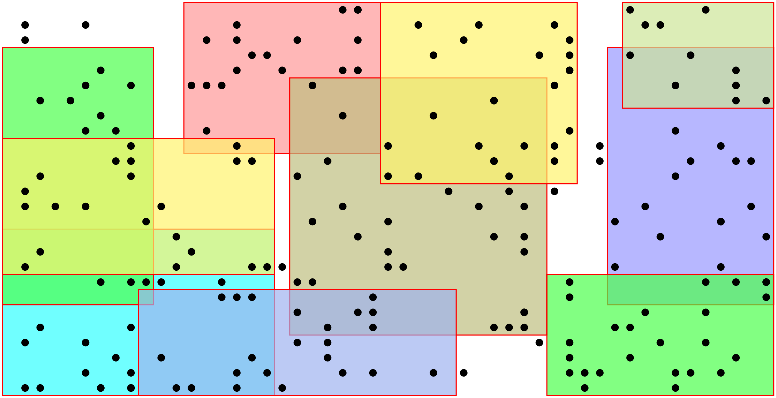
Set Cover



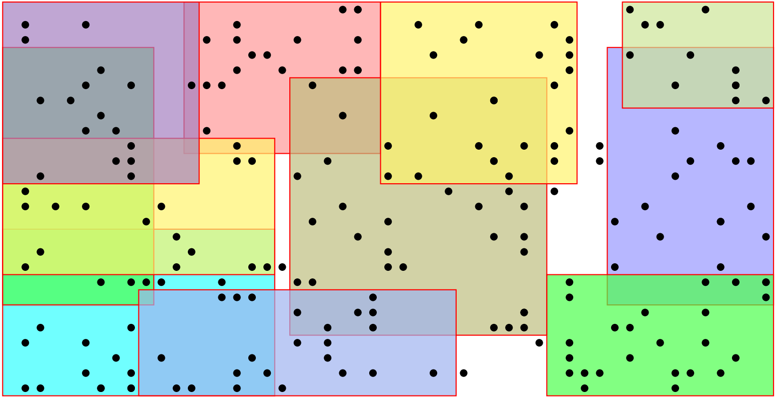
Set Cover



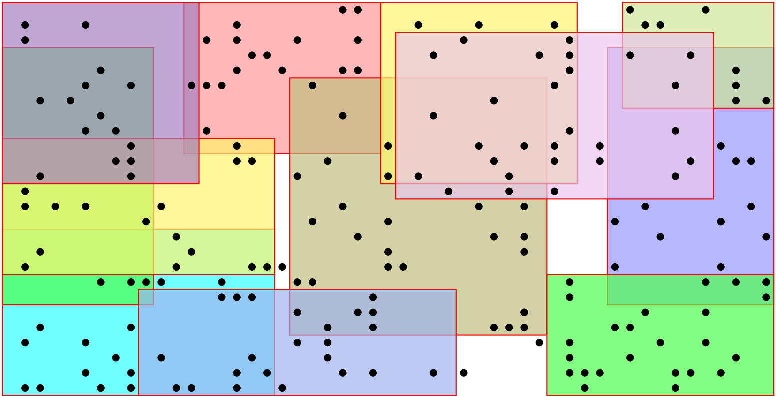
Set Cover



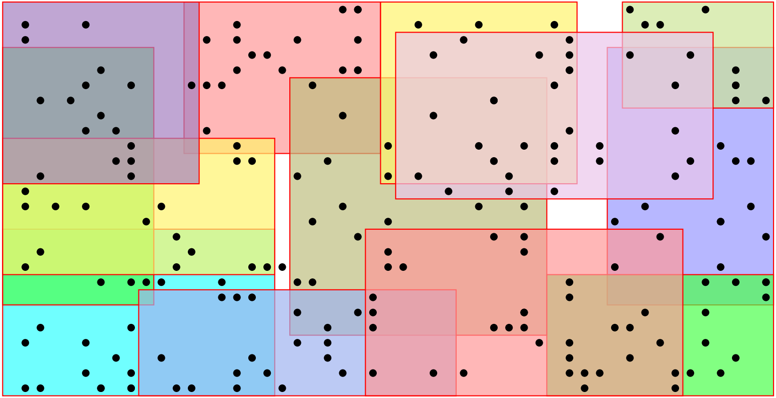
Set Cover



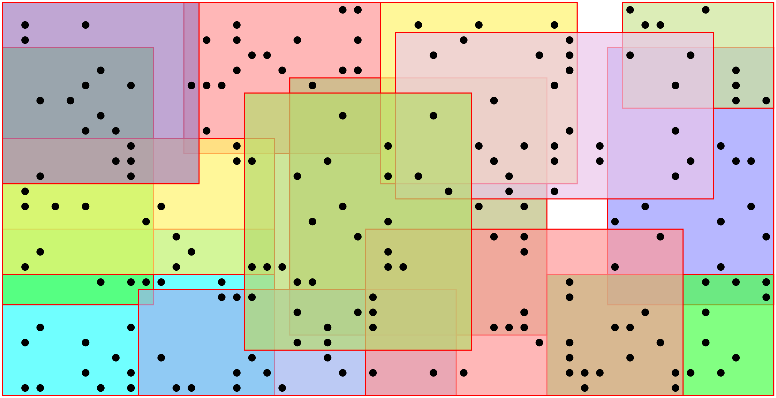
Set Cover



Set Cover



Set Cover



IP-Formulation of Set Cover

$$
\begin{array}{llcll}
\min & & \sum_i w_i x_i & & \\
\text{s.t.} & \forall u \in U & \sum_{i:u\in S_i} x_i & \geq & 1 \\
& \forall i \in \{1,\dots,k\} & x_i & \geq & 0 \\
& \forall i \in \{1,\dots,k\} & x_i & \text{integral} &
\end{array}
$$

Vertex Cover

Given a graph $G = (V, E)$ and a weight w_v for every node. Find a vertex subset $S \subseteq V$ of minimum weight such that every edge is incident to at least one vertex in S.

IP-Formulation of Vertex Cover

$$\begin{array}{llrcl} \min & & \sum_{v\in V} w_v x_v & & \\ \text{s.t.} & \forall e=(i,j)\in E & x_i + x_j & \geq & 1 \\ & \forall v\in V & x_v & \in & \{0,1\} \end{array}$$

Maximum Weighted Matching

Given a graph $G = (V, E)$, and a weight w_e for every edge $e \in E$. Find a subset of edges of maximum weight such that no vertex is incident to more than one edge.

$$\begin{array}{llrcl} \max & & \sum_{e \in E} w_e x_e & & \\ \text{s.t.} & \forall v \in V & \sum_{e: v \in e} x_e & \leq & 1 \\ & \forall e \in E & x_e & \in & \{0,1\} \end{array}$$

Maximum Weighted Matching

Given a graph $G = (V, E)$, and a weight w_e for every edge $e \in E$. Find a subset of edges of maximum weight such that no vertex is incident to more than one edge.

$$\begin{array}{llrcl} \max & & \sum_{e \in E} w_e x_e & & \\ \text{s.t.} & \forall v \in V & \sum_{e: v \in e} x_e & \leq & 1 \\ & \forall e \in E & x_e & \in & \{0, 1\} \end{array}$$

Maximum Independent Set

Given a graph $G = (V, E)$, and a weight w_v for every node $v \in V$. Find a subset $S \subseteq V$ of nodes of maximum weight such that no two vertices in S are adjacent.

$$
\begin{array}{llrcl}
\max & & \sum_{v \in V} w_v x_v & & \\
\text{s.t.} & \forall e = (i,j) \in E & x_i + x_j & \leq & 1 \\
 & \forall v \in V & x_v & \in & \{0,1\}
\end{array}
$$

Maximum Independent Set

Given a graph $G = (V, E)$, and a weight w_v for every node $v \in V$. Find a subset $S \subseteq V$ of nodes of maximum weight such that no two vertices in S are adjacent.

$$\begin{array}{llrcl} \max & & \sum_{v \in V} w_v x_v & & \\ \text{s.t.} & \forall e = (i,j) \in E & x_i + x_j & \leq & 1 \\ & \forall v \in V & x_v & \in & \{0,1\} \end{array}$$

Knapsack

Given a set of items $\{1,\dots,n\}$, where the i-th item has weight w_i and profit p_i, and given a threshold K. Find a subset $I \subseteq \{1,\dots,n\}$ of items of total weight at most K such that the profit is maximized.

$$\begin{array}{llrcl}
\max & & \sum_{i=1}^{n} p_i x_i & & \\
\text{s.t.} & & \sum_{i=1}^{n} w_i x_i & \le & K \\
& \forall i \in \{1,\dots,n\} & x_i & \in & \{0,1\}
\end{array}$$

Knapsack

Given a set of items $\{1, \ldots, n\}$, where the i-th item has weight w_i and profit p_i, and given a threshold K. Find a subset $I \subseteq \{1, \ldots, n\}$ of items of total weight at most K such that the profit is maximized.

$$
\begin{array}{llrcl}
\max & & \sum_{i=1}^{n} p_i x_i & & \\
\text{s.t.} & & \sum_{i=1}^{n} w_i x_i & \leq & K \\
 & \forall i \in \{1, \ldots, n\} & x_i & \in & \{0, 1\}
\end{array}
$$

Relaxations

Definition 54

A linear program LP is a relaxation of an integer program IP if any feasible solution for IP is also feasible for LP and if the objective values of these solutions are identical in both programs.

We obtain a relaxation for all examples by writing $x_i \in [0,1]$ instead of $x_i \in \{0,1\}$.

Relaxations

Definition 54

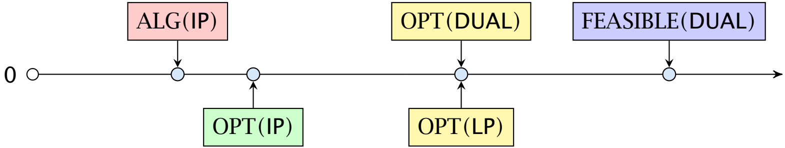
A linear program LP is a relaxation of an integer program IP if any feasible solution for IP is also feasible for LP and if the objective values of these solutions are identical in both programs.

We obtain a relaxation for all examples by writing $x_i \in [0,1]$ instead of $x_i \in \{0,1\}$.

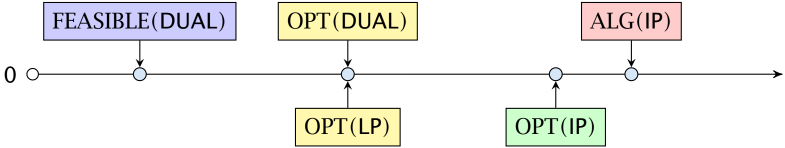
By solving a relaxation we obtain an upper bound for a maximization problem and a lower bound for a minimization problem.

Relations

Maximization Problems:



Minimization Problems:



Technique 1: Round the LP solution.

We first solve the LP-relaxation and then we round the fractional values so that we obtain an integral solution.

Set Cover relaxation:

$$\begin{array}{lrrcl} \min & & \sum_{i=1}^{k} w_i x_i & & \\ \text{s.t.} & \forall u \in U & \sum_{i: u \in S_i} x_i & \geq & 1 \\ & \forall i \in \{1,\dots,k\} & x_i & \in & [0,1] \end{array}$$

Let f_u be the number of sets that the element u is contained in (the frequency of u). Let $f = \max_u \{f_u\}$ be the maximum frequency.

Technique 1: Round the LP solution.

We first solve the LP-relaxation and then we round the fractional values so that we obtain an integral solution.

Set Cover relaxation:

$$\begin{array}{llrcl} \min & & \sum_{i=1}^{k} w_i x_i & & \\ \text{s.t.} & \forall u \in U & \sum_{i:u\in S_i} x_i & \geq & 1 \\ & \forall i \in \{1,\dots,k\} & x_i & \in & [0,1] \end{array}$$

Let f_u be the number of sets that the element u is contained in (the frequency of u). Let $f = \max_u\{f_u\}$ be the maximum frequency.

Technique 1: Round the LP solution.

We first solve the LP-relaxation and then we round the fractional values so that we obtain an integral solution.

Set Cover relaxation:

min		$\sum_{i=1}^{k} w_i x_i$		
s.t.	$\forall u \in U$	$\sum_{i:u\in S_i} x_i$	$\geq$	1
	$\forall i \in \{1,\dots,k\}$	x_i	$\in$	$[0,1]$

Let f_u be the number of sets that the element u is contained in (the frequency of u). Let $f = \max_u\{f_u\}$ be the maximum frequency.

Technique 1: Round the LP solution.

Rounding Algorithm:

Set all x_i-values with $x_i \geq \frac{1}{f}$ to 1. Set all other x_i-values to 0.

Technique 1: Round the LP solution.

Lemma 55

The rounding algorithm gives an f-approximation.

Proof: Every $u \in U$ is covered.

- We know that $\sum_{i: u \in S_i} x_i \geq 1$.
- The sum contains at most $f_u \leq f$ elements.
- Therefore one of the sets that contain u must have $x_i \geq 1/f$.
- This set will be selected. Hence, u is covered.

Technique 1: Round the LP solution.

Lemma 55

The rounding algorithm gives an f-approximation.

Proof: Every $u \in U$ is covered.

- We know that $\sum_{i:u\in S_i} x_i \geq 1$.
- The sum contains at most $f_u \leq f$ elements.
- Therefore one of the sets that contain u must have $x_i \geq 1/f$.
- This set will be selected. Hence, u is covered.

Technique 1: Round the LP solution.

Lemma 55

The rounding algorithm gives an f-approximation.

Proof: Every $u \in U$ is covered.

- We know that $\sum_{i:u\in S_i} x_i \geq 1$.
- The sum contains at most $f_u \leq f$ elements.
- Therefore one of the sets that contain u must have $x_i \geq 1/f$.
- This set will be selected. Hence, u is covered.

Technique 1: Round the LP solution.

Lemma 55

The rounding algorithm gives an f-approximation.

Proof: Every $u \in U$ is covered.

- We know that $\sum_{i:u\in S_i} x_i \geq 1$.
- The sum contains at most $f_u \leq f$ elements.
- Therefore one of the sets that contain u must have $x_i \geq 1/f$.
- This set will be selected. Hence, u is covered.

Technique 1: Round the LP solution.

Lemma 55

The rounding algorithm gives an f-approximation.

Proof: Every $u \in U$ is covered.

- We know that $\sum_{i:u \in S_i} x_i \geq 1$.
- The sum contains at most $f_u \leq f$ elements.
- Therefore one of the sets that contain u must have $x_i \geq 1/f$.
- This set will be selected. Hence, u is covered.

Technique 1: Round the LP solution.

Lemma 55

The rounding algorithm gives an f-approximation.

Proof: Every $u \in U$ is covered.

- We know that $\sum_{i:u\in S_i} x_i \geq 1$.
- The sum contains at most $f_u \leq f$ elements.
- Therefore one of the sets that contain u must have $x_i \geq 1/f$.
- This set will be selected. Hence, u is covered.

Technique 1: Round the LP solution.

The cost of the rounded solution is at most $f \cdot \mathrm{OPT}$.

Technique 1: Round the LP solution.

The cost of the rounded solution is at most $f \cdot \mathrm{OPT}$.

$$\sum_{i \in I} w_i$$

Technique 1: Round the LP solution.

The cost of the rounded solution is at most $f \cdot \mathrm{OPT}$.

$$\sum_{i \in I} w_i \leq \sum_{i=1}^{k} w_i (f \cdot x_i)$$

Technique 1: Round the LP solution.

The cost of the rounded solution is at most $f \cdot \mathrm{OPT}$.

$$\begin{aligned}\sum_{i\in I} w_i &\le \sum_{i=1}^{k} w_i(f \cdot x_i) \\ &= f \cdot \mathrm{cost}(x)\end{aligned}$$

Technique 1: Round the LP solution.

The cost of the rounded solution is at most $f \cdot \text{OPT}$.

$$\begin{aligned}\sum_{i\in I} w_i &\le \sum_{i=1}^{k} w_i(f \cdot x_i)\\ &= f \cdot \text{cost}(x)\\ &\le f \cdot \text{OPT} \enspace .\end{aligned}$$

Technique 2: Rounding the Dual Solution.

Relaxation for Set Cover

Primal:

$$\begin{array}{lrl} \min & \sum_{i\in I} w_i x_i & \\ \text{s.t. } \forall u & \sum_{i:u\in S_i} x_i & \geq 1 \\ & x_i & \geq 0 \end{array}$$

Dual:

$$\begin{array}{lrl} \max & \sum_{u\in U} y_u & \\ \text{s.t. } \forall i & \sum_{u:u\in S_i} y_u & \leq w_i \\ & y_u & \geq 0 \end{array}$$

Technique 2: Rounding the Dual Solution.

Relaxation for Set Cover

Primal:

$$
\begin{array}{lrl}
\min & \sum_{i\in I} w_i x_i & \\
\text{s.t. } \forall u & \sum_{i:u\in S_i} x_i & \geq 1 \\
 & x_i & \geq 0
\end{array}
$$

Dual:

$$
\begin{array}{lrl}
\max & \sum_{u\in U} y_u & \\
\text{s.t. } \forall i & \sum_{u:u\in S_i} y_u & \leq w_i \\
 & y_u & \geq 0
\end{array}
$$

Technique 2: Rounding the Dual Solution.

Relaxation for Set Cover

Primal:

$$\begin{array}{lrl} \min & \sum_{i\in I} w_i x_i & \\ \text{s.t. } \forall u & \sum_{i:u\in S_i} x_i & \geq 1 \\ & x_i & \geq 0 \end{array}$$

Dual:

$$\begin{array}{lrl} \max & \sum_{u\in U} y_u & \\ \text{s.t. } \forall i & \sum_{u:u\in S_i} y_u & \leq w_i \\ & y_u & \geq 0 \end{array}$$

Technique 2: Rounding the Dual Solution.

Rounding Algorithm:

Let I denote the index set of sets for which the dual constraint is tight. This means for all $i \in I$

$$\sum_{u:u\in S_i} y_u = w_i$$

Technique 2: Rounding the Dual Solution.

Lemma 56

The resulting index set is an f-approximation.

Proof:

Every $u \in U$ is covered.

- Suppose there is a u that is not covered.
- This means $\sum_{u: u \in S_j} y_u < w_j$ for all sets S_j that contain u.
- But then y_u could be increased in the dual solution without violating any constraint. This is a contradiction to the fact that the dual solution is optimal.

Technique 2: Rounding the Dual Solution.

Lemma 56

The resulting index set is an f-approximation.

Proof:

Every $u \in U$ is covered.

Technique 2: Rounding the Dual Solution.

Lemma 56

The resulting index set is an f-approximation.

Proof:

Every $u \in U$ is covered.

- Suppose there is a u that is not covered.
- This means $\sum_{u:u\in S_i} y_u < w_i$ for all sets S_i that contain u.
- But then y_u could be increased in the dual solution without violating any constraint. This is a contradiction to the fact that the dual solution is optimal.

Technique 2: Rounding the Dual Solution.

Lemma 56

The resulting index set is an f-approximation.

Proof:

Every $u \in U$ is covered.

- Suppose there is a u that is not covered.
- This means $\sum_{u:u\in S_i} y_u < w_i$ for all sets S_i that contain u.
- But then y_u could be increased in the dual solution without violating any constraint. This is a contradiction to the fact that the dual solution is optimal.

Technique 2: Rounding the Dual Solution.

Lemma 56

The resulting index set is an f-approximation.

Proof:

Every $u \in U$ is covered.

- Suppose there is a u that is not covered.
- This means $\sum_{u:u\in S_i} y_u < w_i$ for all sets S_i that contain u.
- But then y_u could be increased in the dual solution without violating any constraint. This is a contradiction to the fact that the dual solution is optimal.

Technique 2: Rounding the Dual Solution.

Proof:

$$\sum_{i \in I} w_i$$

Technique 2: Rounding the Dual Solution.

Proof:

$$\sum_{i \in I} w_i = \sum_{i \in I} \sum_{u : u \in S_i} y_u$$

Technique 2: Rounding the Dual Solution.

Proof:

$$\begin{aligned}\sum_{i\in I} w_i &= \sum_{i\in I}\sum_{u:u\in S_i} y_u \\ &= \sum_u |\{i\in I : u\in S_i\}|\cdot y_u\end{aligned}$$

Technique 2: Rounding the Dual Solution.

Proof:

$$\begin{aligned}\sum_{i\in I} w_i &= \sum_{i\in I}\sum_{u:u\in S_i} y_u \\ &= \sum_u |\{i\in I : u\in S_i\}|\cdot y_u \\ &\le \sum_u f_u y_u\end{aligned}$$

Technique 2: Rounding the Dual Solution.

Proof:

$$\begin{aligned}
\sum_{i\in I} w_i &= \sum_{i\in I}\sum_{u:u\in S_i} y_u \\
&= \sum_u |\{i\in I : u\in S_i\}|\cdot y_u \\
&\le \sum_u f_u y_u \\
&\le f\sum_u y_u
\end{aligned}$$

Technique 2: Rounding the Dual Solution.

Proof:

$$\begin{aligned}\sum_{i\in I} w_i &= \sum_{i\in I}\sum_{u:u\in S_i} y_u \\ &= \sum_u |\{i \in I : u \in S_i\}| \cdot y_u \\ &\le \sum_u f_u y_u \\ &\le f \sum_u y_u \\ &\le f\,\mathrm{cost}(x^*)\end{aligned}$$

Technique 2: Rounding the Dual Solution.

Proof:

$$
\begin{aligned}
\sum_{i\in I} w_i &= \sum_{i\in I}\sum_{u:u\in S_i} y_u \\
&= \sum_u |\{i\in I : u\in S_i\}|\cdot y_u \\
&\le \sum_u f_u y_u \\
&\le f\sum_u y_u \\
&\le f\,\mathrm{cost}(x^*) \\
&\le f\cdot \mathrm{OPT}
\end{aligned}
$$

Let I denote the solution obtained by the first rounding algorithm and I' be the solution returned by the second algorithm. Then

$$I \subseteq I' \ .$$

This means I' is never better than I.

- Suppose that we take S_i in the first algorithm. I.e., $i \in I$.
- This means $x_i \geq \frac{1}{f}$.
- Because of Complementary Slackness Conditions the corresponding constraint in the dual must be tight.
- Hence, the second algorithm will also choose S_i.

Let I denote the solution obtained by the first rounding algorithm and I' be the solution returned by the second algorithm. Then

$$I \subseteq I' \ .$$

This means I' is never better than I.

- Suppose that we take S_i in the first algorithm. I.e., $i \in I$.
- This means $x_i \geq \frac{1}{f}$.
- Because of Complementary Slackness Conditions the corresponding constraint in the dual must be tight.
- Hence, the second algorithm will also choose S_i.

Let I denote the solution obtained by the first rounding algorithm and I' be the solution returned by the second algorithm. Then

$$I \subseteq I' \ .$$

This means I' is never better than I.

- Suppose that we take S_i in the first algorithm. I.e., $i \in I$.
- This means $x_i \geq \frac{1}{f}$.
- Because of Complementary Slackness Conditions the corresponding constraint in the dual must be tight.
- Hence, the second algorithm will also choose S_i.

Let I denote the solution obtained by the first rounding algorithm and I' be the solution returned by the second algorithm. Then

$$I \subseteq I' .$$

This means I' is never better than I.

- Suppose that we take S_i in the first algorithm. I.e., $i \in I$.
- This means $x_i \geq \frac{1}{f}$.
- Because of Complementary Slackness Conditions the corresponding constraint in the dual must be tight.
- Hence, the second algorithm will also choose S_i.

Let I denote the solution obtained by the first rounding algorithm and I' be the solution returned by the second algorithm. Then

$$I \subseteq I' \ .$$

This means I' is never better than I.

- Suppose that we take S_i in the first algorithm. I.e., $i \in I$.
- This means $x_i \geq \frac{1}{f}$.
- Because of Complementary Slackness Conditions the corresponding constraint in the dual must be tight.
- Hence, the second algorithm will also choose S_i.

Technique 3: The Primal Dual Method

The previous two rounding algorithms have the disadvantage that it is necessary to solve the LP. The following method also gives an f-approximation without solving the LP.

For estimating the cost of the solution we only required two properties.

1. The solution is dual feasible and, hence,

$$\sum_u y_u \le \text{cost}(x^*) \le \text{OPT}$$

   where x^* is an optimum solution to the primal LP.

2. The set I contains only sets for which the dual inequality is tight.

Of course, we also need that I is a cover.

Technique 3: The Primal Dual Method

The previous two rounding algorithms have the disadvantage that it is necessary to solve the LP. The following method also gives an f-approximation without solving the LP.

For estimating the cost of the solution we only required two properties.

- The solution is dual feasible and, hence,

$$\sum_u y_u \le \text{cost}(x^*) \le \text{OPT}$$

  where x^* is an optimum solution to the primal LP.

- The set I contains only sets for which the dual inequality is tight.

Of course, we also need that I is a cover.

Technique 3: The Primal Dual Method

The previous two rounding algorithms have the disadvantage that it is necessary to solve the LP. The following method also gives an f-approximation without solving the LP.

For estimating the cost of the solution we only required two properties.

1. The solution is dual feasible and, hence,

$$\sum_u y_u \leq \mathrm{cost}(x^*) \leq \mathrm{OPT}$$

   where x^* is an optimum solution to the primal LP.

2. The set I contains only sets for which the dual inequality is tight.

Of course, we also need that I is a cover.

Technique 3: The Primal Dual Method

The previous two rounding algorithms have the disadvantage that it is necessary to solve the LP. The following method also gives an f-approximation without solving the LP.

For estimating the cost of the solution we only required two properties.

1. The solution is dual feasible and, hence,

$$\sum_u y_u \leq \text{cost}(x^*) \leq \text{OPT}$$

   where x^* is an optimum solution to the primal LP.

2. The set I contains only sets for which the dual inequality is tight.

Of course, we also need that I is a cover.

Technique 3: The Primal Dual Method

The previous two rounding algorithms have the disadvantage that it is necessary to solve the LP. The following method also gives an f-approximation without solving the LP.

For estimating the cost of the solution we only required two properties.

1. The solution is dual feasible and, hence,

$$\sum_u y_u \leq \text{cost}(x^*) \leq \text{OPT}$$

   where x^* is an optimum solution to the primal LP.

2. The set I contains only sets for which the dual inequality is tight.

Of course, we also need that I is a cover.

Technique 3: The Primal Dual Method

Algorithm 1 PrimalDual

1: $y \leftarrow 0$
2: $I \leftarrow \emptyset$
3: **while** exists $u \notin \bigcup_{i \in I} S_i$ **do**
4: increase dual variable y_u until constraint for some new set S_ℓ becomes tight
5: $I \leftarrow I \cup \{\ell\}$

Technique 4: The Greedy Algorithm

Algorithm 1 Greedy

1: $I \leftarrow \emptyset$
2: $\hat{S}_j \leftarrow S_j$ for all j
3: **while** I not a set cover **do**
4: $\quad \ell \leftarrow \arg\min_{j:\hat{S}_j \neq 0} \frac{w_j}{|\hat{S}_j|}$
5: $\quad I \leftarrow I \cup \{\ell\}$
6: $\quad \hat{S}_j \leftarrow \hat{S}_j - S_\ell$ for all j

In every round the Greedy algorithm takes the set that covers remaining elements in the most cost-effective way.

We choose a set such that the ratio between cost and still uncovered elements in the set is minimized.

Technique 4: The Greedy Algorithm

Lemma 57

Given positive numbers $a_1, \dots, a_k$ *and* $b_1, \dots, b_k$*, and* $S \subseteq \{1, \dots, k\}$ *then*

$$\min_i \frac{a_i}{b_i} \le \frac{\sum_{i \in S} a_i}{\sum_{i \in S} b_i} \le \max_i \frac{a_i}{b_i}$$

Technique 4: The Greedy Algorithm

Let n_ℓ denote the number of elements that remain at the beginning of iteration ℓ. $n_1 = n = |U|$ and $n_{s+1} = 0$ if we need s iterations.

In the ℓ-th iteration

$$\min_j \frac{w_j}{|\hat{S}_j|} \le \frac{\sum_{j\in\mathrm{OPT}} w_j}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} = \frac{\mathrm{OPT}}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} \le \frac{\mathrm{OPT}}{n_\ell}$$

since an optimal algorithm can cover the remaining n_ℓ elements with cost OPT.

Let $\hat{S}_j$ be a subset that minimizes this ratio. Hence, $w_j/|\hat{S}_j| \le \frac{\mathrm{OPT}}{n_\ell}$.

Technique 4: The Greedy Algorithm

Let n_ℓ denote the number of elements that remain at the beginning of iteration ℓ. $n_1 = n = |U|$ and $n_{s+1} = 0$ if we need s iterations.

In the ℓ-th iteration

$$\min_j \frac{w_j}{|\hat{S}_j|} \le \frac{\sum_{j\in\mathrm{OPT}} w_j}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} = \frac{\mathrm{OPT}}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} \le \frac{\mathrm{OPT}}{n_\ell}$$

since an optimal algorithm can cover the remaining n_ℓ elements with cost OPT.

Let $\hat{S}_j$ be a subset that minimizes this ratio. Hence, $w_j/|\hat{S}_j| \le \frac{\mathrm{OPT}}{n_\ell}$.

Technique 4: The Greedy Algorithm

Let n_ℓ denote the number of elements that remain at the beginning of iteration ℓ. $n_1 = n = |U|$ and $n_{s+1} = 0$ if we need s iterations.

In the ℓ-th iteration

$$\min_j \frac{w_j}{|\hat{S}_j|} \le \frac{\sum_{j\in\mathrm{OPT}} w_j}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} = \frac{\mathrm{OPT}}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} \le \frac{\mathrm{OPT}}{n_\ell}$$

since an optimal algorithm can cover the remaining n_ℓ elements with cost OPT.

Let $\hat{S}_j$ be a subset that minimizes this ratio. Hence, $w_j/|\hat{S}_j| \le \frac{\mathrm{OPT}}{n_\ell}$.

Technique 4: The Greedy Algorithm

Let n_ℓ denote the number of elements that remain at the beginning of iteration ℓ. $n_1 = n = |U|$ and $n_{s+1} = 0$ if we need s iterations.

In the ℓ-th iteration

$$\min_j \frac{w_j}{|\hat{S}_j|} \le \frac{\sum_{j\in\mathrm{OPT}} w_j}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} = \frac{\mathrm{OPT}}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} \le \frac{\mathrm{OPT}}{n_\ell}$$

since an optimal algorithm can cover the remaining n_ℓ elements with cost OPT.

Let $\hat{S}_j$ be a subset that minimizes this ratio. Hence, $w_j/|\hat{S}_j| \le \frac{\mathrm{OPT}}{n_\ell}$.

Technique 4: The Greedy Algorithm

Let n_ℓ denote the number of elements that remain at the beginning of iteration ℓ. $n_1 = n = |U|$ and $n_{s+1} = 0$ if we need s iterations.

In the ℓ-th iteration

$$\min_j \frac{w_j}{|\hat{S}_j|} \leq \frac{\sum_{j\in\mathrm{OPT}} w_j}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} = \frac{\mathrm{OPT}}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} \leq \frac{\mathrm{OPT}}{n_\ell}$$

since an optimal algorithm can cover the remaining n_ℓ elements with cost OPT.

Let $\hat{S}_j$ be a subset that minimizes this ratio. Hence, $w_j/|\hat{S}_j| \leq \frac{\mathrm{OPT}}{n_\ell}$.

Technique 4: The Greedy Algorithm

Let n_ℓ denote the number of elements that remain at the beginning of iteration ℓ. $n_1 = n = |U|$ and $n_{s+1} = 0$ if we need s iterations.

In the ℓ-th iteration

$$\min_j \frac{w_j}{|\hat{S}_j|} \le \frac{\sum_{j\in\mathrm{OPT}} w_j}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} = \frac{\mathrm{OPT}}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} \le \frac{\mathrm{OPT}}{n_\ell}$$

since an optimal algorithm can cover the remaining n_ℓ elements with cost OPT.

Let $\hat{S}_j$ be a subset that minimizes this ratio. Hence, $w_j/|\hat{S}_j| \le \frac{\mathrm{OPT}}{n_\ell}$.

Technique 4: The Greedy Algorithm

Let n_ℓ denote the number of elements that remain at the beginning of iteration ℓ. $n_1 = n = |U|$ and $n_{s+1} = 0$ if we need s iterations.

In the ℓ-th iteration

$$\min_j \frac{w_j}{|\hat{S}_j|} \leq \frac{\sum_{j\in\mathrm{OPT}} w_j}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} = \frac{\mathrm{OPT}}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} \leq \frac{\mathrm{OPT}}{n_\ell}$$

since an optimal algorithm can cover the remaining n_ℓ elements with cost OPT.

Let $\hat{S}_j$ be a subset that minimizes this ratio. Hence, $w_j/|\hat{S}_j| \leq \frac{\mathrm{OPT}}{n_\ell}$.

Technique 4: The Greedy Algorithm

Adding this set to our solution means $n_{\ell+1} = n_\ell - |\hat{S}_j|$.

$$w_j \le \frac{|\hat{S}_j|\text{OPT}}{n_\ell} = \frac{n_\ell - n_{\ell+1}}{n_\ell} \cdot \text{OPT}$$

Technique 4: The Greedy Algorithm

Adding this set to our solution means $n_{\ell+1} = n_\ell - |\hat{S}_j|$.

$$w_j \le \frac{|\hat{S}_j|\mathrm{OPT}}{n_\ell} = \frac{n_\ell - n_{\ell+1}}{n_\ell} \cdot \mathrm{OPT}$$

Technique 4: The Greedy Algorithm

$$\sum_{j \in I} w_j$$

Technique 4: The Greedy Algorithm

$$\sum_{j\in I} w_j \le \sum_{\ell=1}^{s} \frac{n_\ell - n_{\ell+1}}{n_\ell} \cdot \mathrm{OPT}$$

Technique 4: The Greedy Algorithm

$$\sum_{j\in I} w_j \le \sum_{\ell=1}^{s} \frac{n_\ell - n_{\ell+1}}{n_\ell} \cdot \mathrm{OPT}$$

$$\le \mathrm{OPT} \sum_{\ell=1}^{s} \left(\frac{1}{n_\ell} + \frac{1}{n_\ell - 1} + \dots + \frac{1}{n_{\ell+1} + 1} \right)$$

Technique 4: The Greedy Algorithm

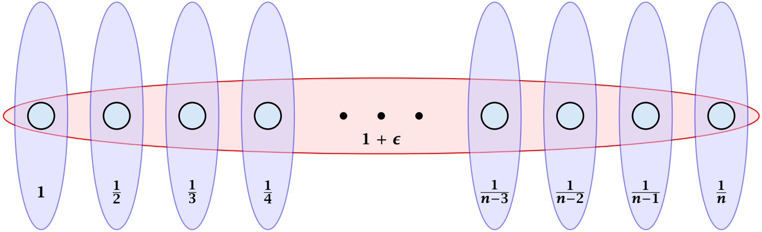
$$\begin{aligned}\sum_{j\in I} w_j &\le \sum_{\ell=1}^{s} \frac{n_\ell - n_{\ell+1}}{n_\ell}\cdot \mathrm{OPT}\\ &\le \mathrm{OPT}\sum_{\ell=1}^{s}\left(\frac{1}{n_\ell}+\frac{1}{n_\ell - 1}+\cdots+\frac{1}{n_{\ell+1}+1}\right)\\ &= \mathrm{OPT}\sum_{i=1}^{k}\frac{1}{i}\end{aligned}$$

Technique 4: The Greedy Algorithm

$$
\begin{aligned}
\sum_{j\in I} w_j &\le \sum_{\ell=1}^{s} \frac{n_\ell - n_{\ell+1}}{n_\ell}\cdot \mathrm{OPT} \\
&\le \mathrm{OPT}\sum_{\ell=1}^{s}\left(\frac{1}{n_\ell}+\frac{1}{n_\ell - 1}+\cdots+\frac{1}{n_{\ell+1}+1}\right) \\
&= \mathrm{OPT}\sum_{i=1}^{k}\frac{1}{i} \\
&= H_n\cdot \mathrm{OPT} \le \mathrm{OPT}(\ln n + 1)\enspace .
\end{aligned}
$$

Technique 4: The Greedy Algorithm

A tight example:



Technique 5: Randomized Rounding

One round of randomized rounding:
Pick set S_j uniformly at random with probability $1 - x_j$ (for all j).

Version A: Repeat rounds until you have a cover.

Version B: Repeat for s rounds. If you have a cover STOP.
Otherwise, repeat the whole algorithm.

Technique 5: Randomized Rounding

One round of randomized rounding:
Pick set S_j uniformly at random with probability $1 - x_j$ (for all j).

Version A: Repeat rounds until you have a cover.

Version B: Repeat for s rounds. If you have a cover STOP.
Otherwise, repeat the whole algorithm.

Technique 5: Randomized Rounding

One round of randomized rounding:
Pick set S_j uniformly at random with probability $1 - x_j$ (for all j).

Version A: Repeat rounds until you have a cover.

Version B: Repeat for s rounds. If you have a cover STOP.
Otherwise, repeat the whole algorithm.

Probability that $u \in U$ is not covered (in one round):

$$\Pr[u \text{ not covered in one round}]$$

Probability that $u \in U$ is not covered (in one round):

$$\Pr[u \text{ not covered in one round}] = \prod_{j:u\in S_j}(1-x_j)$$

Probability that $u \in U$ is not covered (in one round):

$$\Pr[u \text{ not covered in one round}] = \prod_{j:u\in S_j}(1-x_j) \leq \prod_{j:u\in S_j} e^{-x_j}$$

Probability that $u \in U$ is not covered (in one round):

$$\Pr[u \text{ not covered in one round}]$$
$$= \prod_{j:u\in S_j} (1 - x_j) \le \prod_{j:u\in S_j} e^{-x_j}$$
$$= e^{-\sum_{j:u\in S_j} x_j}$$

Probability that $u \in U$ is not covered (in one round):

$$\begin{aligned}
\Pr[u \text{ not covered in one round}] \\
&= \prod_{j:u\in S_j} (1 - x_j) \le \prod_{j:u\in S_j} e^{-x_j} \\
&= e^{-\sum_{j:u\in S_j} x_j} \le e^{-1} \enspace .
\end{aligned}$$

Probability that $u \in U$ is not covered (in one round):

$$
\begin{aligned}
\Pr[u \text{ not covered in one round}] \\
&= \prod_{j:u\in S_j} (1 - x_j) \le \prod_{j:u\in S_j} e^{-x_j} \\
&= e^{-\sum_{j:u\in S_j} x_j} \le e^{-1} \enspace .
\end{aligned}
$$

Probability that $u \in U$ is not covered (after ℓ rounds):

$$
\Pr[u \text{ not covered after } \ell \text{ round}] \le \frac{1}{e^{\ell}} \enspace .
$$

$\Pr[\exists u \in U \text{ not covered after } \ell \text{ round}]$

$$\Pr[\exists u \in U \text{ not covered after } \ell \text{ round}]$$
$$= \Pr[u_1 \text{ not covered} \vee u_2 \text{ not covered} \vee \ldots \vee u_n \text{ not covered}]$$

$$
\begin{aligned}
&\Pr[\exists u \in U \text{ not covered after } \ell \text{ round}] \\
&\qquad = \Pr[u_1 \text{ not covered} \vee u_2 \text{ not covered} \vee \ldots \vee u_n \text{ not covered}] \\
&\qquad \le \sum_i \Pr[u_i \text{ not covered after } \ell \text{ rounds}]
\end{aligned}
$$

$$
\begin{aligned}
\Pr[\exists u \in U \text{ not covered after } \ell \text{ round}] \\
&= \Pr[u_1 \text{ not covered} \vee u_2 \text{ not covered} \vee \ldots \vee u_n \text{ not covered}] \\
&\le \sum_i \Pr[u_i \text{ not covered after } \ell \text{ rounds}] \le ne^{-\ell} \enspace .
\end{aligned}
$$

$$
\begin{aligned}
&\Pr[\exists u \in U \text{ not covered after } \ell \text{ round}] \\
&\quad = \Pr[u_1 \text{ not covered} \vee u_2 \text{ not covered} \vee \ldots \vee u_n \text{ not covered}] \\
&\quad \le \sum_i \Pr[u_i \text{ not covered after } \ell \text{ rounds}] \le ne^{-\ell} \enspace .
\end{aligned}
$$

Lemma 58

With high probability $\mathcal{O}(\log n)$ rounds suffice.

$$\begin{aligned}
&\Pr[\exists u \in U \text{ not covered after } \ell \text{ round}] \\
&\qquad = \Pr[u_1 \text{ not covered} \vee u_2 \text{ not covered} \vee \ldots \vee u_n \text{ not covered}] \\
&\qquad \leq \sum_i \Pr[u_i \text{ not covered after } \ell \text{ rounds}] \leq ne^{-\ell} \enspace .
\end{aligned}$$

Lemma 58

With high probability $\mathcal{O}(\log n)$ rounds suffice.

With high probability:

For any constant α the number of rounds is at most $\mathcal{O}(\log n)$ with probability at least $1 - n^{-\alpha}$.

Proof: We have

$$\Pr[\#\text{rounds} \geq (\alpha+1)\ln n] \leq ne^{-(\alpha+1)\ln n} = n^{-\alpha} \ .$$

Expected Cost

- Version A.
  Repeat for $s = (\alpha + 1)\ln n$ rounds. If you don't have a cover simply take for each element u the cheapest set that contains u.

Expected Cost

- Version A.
  Repeat for $s = (\alpha + 1)\ln n$ rounds. If you don't have a cover simply take for each element u the cheapest set that contains u.

  $E[\text{cost}]$

Expected Cost

- Version A.
  Repeat for $s = (\alpha + 1)\ln n$ rounds. If you don't have a cover simply take for each element u the cheapest set that contains u.

  $$E[\text{cost}] \leq (\alpha+1)\ln n \cdot \text{cost}(LP) + (n \cdot \text{OPT})n^{-\alpha}$$

Expected Cost

- Version A.
  Repeat for $s = (\alpha + 1)\ln n$ rounds. If you don't have a cover simply take for each element u the cheapest set that contains u.

$$E[\text{cost}] \leq (\alpha+1)\ln n \cdot \text{cost}(LP) + (n \cdot \text{OPT})n^{-\alpha} = \mathcal{O}(\ln n) \cdot \text{OPT}$$

Expected Cost

- Version B.
  Repeat for $s = (\alpha + 1)\ln n$ rounds. If you don't have a cover simply repeat the whole process.

$$E[\text{cost}] =$$

Expected Cost

- Version B.
  Repeat for $s = (\alpha + 1)\ln n$ rounds. If you don't have a cover simply repeat the whole process.

$$E[\text{cost}] = \Pr[\text{success}] \cdot E[\text{cost} \mid \text{success}] + \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}]$$

Expected Cost

- Version B.
  Repeat for $s = (\alpha + 1)\ln n$ rounds. If you don't have a cover simply repeat the whole process.

$$
\begin{aligned}
E[\text{cost}] = \Pr[\text{success}] \cdot E[\text{cost} \mid \text{success}] \\
+ \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}]
\end{aligned}
$$

This means

$E[\text{cost} \mid \text{success}]$

Expected Cost

- Version B.
  Repeat for $s = (\alpha + 1)\ln n$ rounds. If you don't have a cover simply repeat the whole process.

$$
\begin{aligned}
E[\text{cost}] = \Pr[\text{success}] \cdot E[\text{cost} \mid \text{success}] \\
+ \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}]
\end{aligned}
$$

This means

$$
\begin{aligned}
&E[\text{cost} \mid \text{success}] \\
&\quad = \frac{1}{\Pr[\text{succ.}]}\Big(E[\text{cost}] - \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}]\Big)
\end{aligned}
$$

Expected Cost

- Version B.
  Repeat for $s = (\alpha + 1)\ln n$ rounds. If you don't have a cover simply repeat the whole process.

$$E[\text{cost}] = \Pr[\text{success}] \cdot E[\text{cost} \mid \text{success}] + \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}]$$

This means

$$\begin{aligned} &E[\text{cost} \mid \text{success}] \\ &\quad = \frac{1}{\Pr[\text{succ.}]}\Big(E[\text{cost}] - \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}]\Big) \\ &\quad \le \frac{1}{\Pr[\text{succ.}]} E[\text{cost}] \le \frac{1}{1 - n^{-\alpha}} (\alpha + 1)\ln n \cdot \text{cost}(\text{LP}) \end{aligned}$$

Expected Cost

- Version B.
  Repeat for $s = (\alpha + 1)\ln n$ rounds. If you don't have a cover simply repeat the whole process.

$$\begin{aligned} E[\text{cost}] = \Pr[\text{success}] \cdot E[\text{cost} \mid \text{success}] \\ + \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}] \end{aligned}$$

This means

$$\begin{aligned} & E[\text{cost} \mid \text{success}] \\ & = \frac{1}{\Pr[\text{succ.}]}\Big(E[\text{cost}] - \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}]\Big) \\ & \leq \frac{1}{\Pr[\text{succ.}]} E[\text{cost}] \leq \frac{1}{1 - n^{-\alpha}} (\alpha + 1)\ln n \cdot \text{cost}(\text{LP}) \\ & \leq 2(\alpha + 1)\ln n \cdot \text{OPT} \end{aligned}$$

Expected Cost

- Version B.
  Repeat for $s = (\alpha + 1)\ln n$ rounds. If you don't have a cover simply repeat the whole process.

$$E[\text{cost}] = \Pr[\text{success}] \cdot E[\text{cost} \mid \text{success}] + \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}]$$

This means

$$\begin{aligned} E[\text{cost} \mid \text{success}] &= \frac{1}{\Pr[\text{succ.}]}\Big(E[\text{cost}] - \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}]\Big) \\ &\le \frac{1}{\Pr[\text{succ.}]} E[\text{cost}] \le \frac{1}{1 - n^{-\alpha}}(\alpha + 1)\ln n \cdot \text{cost}(LP) \\ &\le 2(\alpha + 1)\ln n \cdot \text{OPT} \end{aligned}$$

for $n \ge 2$ and $\alpha \ge 1$.

Randomized rounding gives an $\mathcal{O}(\log n)$ approximation. The running time is polynomial with high probability.

Theorem 59 (without proof)

There is no approximation algorithm for set cover with approximation guarantee better than $\frac{1}{2}\log n$ unless NP has quasi-polynomial time algorithms (algorithms with running time $2^{\text{poly}(\log n)}$).

Randomized rounding gives an $\mathcal{O}(\log n)$ approximation. The running time is polynomial with high probability.

Theorem 59 (without proof)

There is no approximation algorithm for set cover with approximation guarantee better than $\frac{1}{2}\log n$ unless NP has quasi-polynomial time algorithms (algorithms with running time $2^{\mathrm{poly}(\log n)}$).

Integrality Gap

The integrality gap of the SetCover LP is $\Omega(\log n)$.

- $n = 2^k - 1$
- Elements are all vectors $\boldsymbol{i}$ over $GF[2]$ of length k (excluding zero vector).
- Every vector $\boldsymbol{j}$ defines a set as follows

$$S_{\boldsymbol{j}} := \{\boldsymbol{i} \mid \boldsymbol{i} \cdot \boldsymbol{j} = 1\}$$

- each set contains 2^{k-1} vectors; each vector is contained in 2^{k-1} sets
- $x_i = \frac{1}{2^{k-1}} = \frac{2}{n+1}$ is fractional solution.

Integrality Gap

Every collection of $p < k$ sets does not cover all elements.

Hence, we get a gap of $\Omega(\log n)$.

Techniques:

- Deterministic Rounding
- Rounding of the Dual
- Primal Dual
- Greedy
- Randomized Rounding
- Local Search
- Rounding Data + Dynamic Programming

Scheduling Jobs on Identical Parallel Machines

Given n jobs, where job $j \in \{1, \ldots, n\}$ has processing time p_j. Schedule the jobs on m identical parallel machines such that the Makespan (finishing time of the last job) is minimized.

$$\begin{array}{llll} \min & & & L \\ \text{s.t.} & \forall \text{machines } i & \sum_j p_j \cdot x_{j,i} & \leq & L \\ & \forall \text{jobs } j & \sum_i x_{j,i} \geq 1 & & \\ & \forall i,j & x_{j,i} & \in & \{0,1\} \end{array}$$

Here the variable $x_{j,i}$ is the decision variable that describes whether job j is assigned to machine i.

Scheduling Jobs on Identical Parallel Machines

Given n jobs, where job $j \in \{1, \ldots, n\}$ has processing time p_j. Schedule the jobs on m identical parallel machines such that the Makespan (finishing time of the last job) is minimized.

$$\begin{array}{llrcl}
\min & & & L & \\
\text{s.t.} & \forall \text{machines } i & \sum_j p_j \cdot x_{j,i} & \leq & L \\
& \forall \text{jobs } j & \sum_i x_{j,i} \geq 1 & & \\
& \forall i, j & x_{j,i} & \in & \{0, 1\}
\end{array}$$

Here the variable $x_{j,i}$ is the decision variable that describes whether job j is assigned to machine i.

Lower Bounds on the Solution

Let for a given schedule C_j denote the finishing time of machine j, and let $C_{\max}$ be the makespan.

Let $C^*_{\max}$ denote the makespan of an optimal solution.

Clearly

$$C^*_{\max} \geq \max_j p_j$$

as the longest job needs to be scheduled somewhere.

Lower Bounds on the Solution

Let for a given schedule C_j denote the finishing time of machine j, and let $C_{\max}$ be the makespan.

Let $C^*_{\max}$ denote the makespan of an optimal solution.

Clearly

$$C^*_{\max} \geq \max_j p_j$$

as the longest job needs to be scheduled somewhere.

Lower Bounds on the Solution

Let for a given schedule C_j denote the finishing time of machine j, and let $C_{\max}$ be the makespan.

Let $C^*_{\max}$ denote the makespan of an optimal solution.

Clearly

$$C^*_{\max} \geq \max_j p_j$$

as the longest job needs to be scheduled somewhere.

Lower Bounds on the Solution

The average work performed by a machine is $\frac{1}{m}\sum_j p_j$.

Therefore,

$$C^*_{\max} \geq \frac{1}{m}\sum_j p_j$$

Lower Bounds on the Solution

The average work performed by a machine is $\frac{1}{m}\sum_j p_j$. Therefore,

$$C^*_{\max} \geq \frac{1}{m}\sum_j p_j$$

Local Search

A local search algorithm successivley makes certain small (cost/profit improving) changes to a solution until it does not find such changes anymore.

It is conceptionally very different from a Greedy algorithm as a feasible solution is always maintained.

Sometimes the running time is difficult to prove.

Local Search

A local search algorithm successivley makes certain small (cost/profit improving) changes to a solution until it does not find such changes anymore.

It is conceptionally very different from a Greedy algorithm as a feasible solution is always maintained.

Sometimes the running time is difficult to prove.

Local Search

A local search algorithm successivley makes certain small (cost/profit improving) changes to a solution until it does not find such changes anymore.

It is conceptionally very different from a Greedy algorithm as a feasible solution is always maintained.

Sometimes the running time is difficult to prove.

Local Search

A local search algorithm successivley makes certain small (cost/profit improving) changes to a solution until it does not find such changes anymore.

It is conceptionally very different from a Greedy algorithm as a feasible solution is always maintained.

Sometimes the running time is difficult to prove.

Local Search for Scheduling

Local Search Strategy: Take the job that finishes last and try to move it to another machine. If there is such a move that reduces the makespan, perform the switch.

REPEAT

Local Search for Scheduling

Local Search Strategy: Take the job that finishes last and try to move it to another machine. If there is such a move that reduces the makespan, perform the switch.

REPEAT

Local Search for Scheduling

Local Search Strategy: Take the job that finishes last and try to move it to another machine. If there is such a move that reduces the makespan, perform the switch.

REPEAT

Local Search Analysis

Let ℓ be the job that finishes last in the produced schedule.

Let S_ℓ be its start time, and let C_ℓ be its completion time.

Note that every machine is busy before time S_ℓ, because otherwise we could move the job ℓ and hence our schedule would not be locally optimal.

Local Search Analysis

Let ℓ be the job that finishes last in the produced schedule.

Let S_ℓ be its start time, and let C_ℓ be its completion time.

Note that every machine is busy before time S_ℓ, because otherwise we could move the job ℓ and hence our schedule would not be locally optimal.

Local Search Analysis

Let ℓ be the job that finishes last in the produced schedule.

Let S_ℓ be its start time, and let C_ℓ be its completion time.

Note that every machine is busy before time S_ℓ, because otherwise we could move the job ℓ and hence our schedule would not be locally optimal.

Local Search Analysis

Let ℓ be the job that finishes last in the produced schedule.

Let S_ℓ be its start time, and let C_ℓ be its completion time.

Note that every machine is busy before time S_ℓ, because otherwise we could move the job ℓ and hence our schedule would not be locally optimal.

We can split the total processing time into two intervals one from 0 to S_ℓ the other from S_ℓ to C_ℓ.

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \le C^*_{\max}$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \le \sum_{j \neq \ell} p_j \ .$$

Hence, the length of the schedule is at most

$$p_\ell + \frac{1}{m}\sum_{j \neq \ell} p_j = (1 - \frac{1}{m})p_\ell + \frac{1}{m}\sum_j p_j \le (2 - \frac{1}{m})C^*_{\max}$$

We can split the total processing time into two intervals one from 0 to S_ℓ the other from S_ℓ to C_ℓ.

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \le C^*_{\max}$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \le \sum_{j \neq \ell} p_j \enspace .$$

Hence, the length of the schedule is at most

$$p_\ell + \frac{1}{m}\sum_{j \neq \ell} p_j = (1 - \frac{1}{m})p_\ell + \frac{1}{m}\sum_j p_j \le (2 - \frac{1}{m})C^*_{\max}$$

We can split the total processing time into two intervals one from 0 to S_ℓ the other from S_ℓ to C_ℓ.

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \le C^*_{\max}$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \le \sum_{j \ne \ell} p_j \enspace .$$

Hence, the length of the schedule is at most

$$p_\ell + \frac{1}{m}\sum_{j\ne\ell} p_j = (1 - \frac{1}{m})p_\ell + \frac{1}{m}\sum_j p_j \le (2 - \frac{1}{m})C^*_{\max}$$

We can split the total processing time into two intervals one from 0 to S_ℓ the other from S_ℓ to C_ℓ.

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \le C^*_{\max}$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \le \sum_{j \neq \ell} p_j \ .$$

Hence, the length of the schedule is at most

$$p_\ell + \frac{1}{m}\sum_{j \neq \ell} p_j = (1 - \frac{1}{m})p_\ell + \frac{1}{m}\sum_j p_j \le (2 - \frac{1}{m})C^*_{\max}$$

We can split the total processing time into two intervals one from 0 to S_ℓ the other from S_ℓ to C_ℓ.

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \leq C^*_{\max}$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \leq \sum_{j \neq \ell} p_j \enspace .$$

Hence, the length of the schedule is at most

$$p_\ell + \frac{1}{m}\sum_{j \neq \ell} p_j = (1 - \frac{1}{m})p_\ell + \frac{1}{m}\sum_j p_j \leq (2 - \frac{1}{m})C^*_{\max}$$

We can split the total processing time into two intervals one from 0 to S_ℓ the other from S_ℓ to C_ℓ.

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \leq C^*_{\max}$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \leq \sum_{j \neq \ell} p_j \enspace .$$

Hence, the length of the schedule is at most

$$p_\ell + \frac{1}{m} \sum_{j \neq \ell} p_j = (1 - \frac{1}{m}) p_\ell + \frac{1}{m} \sum_j p_j \leq (2 - \frac{1}{m}) C^*_{\max}$$

We can split the total processing time into two intervals one from 0 to S_ℓ the other from S_ℓ to C_ℓ.

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \le C^*_{\max}$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \le \sum_{j \ne \ell} p_j \ .$$

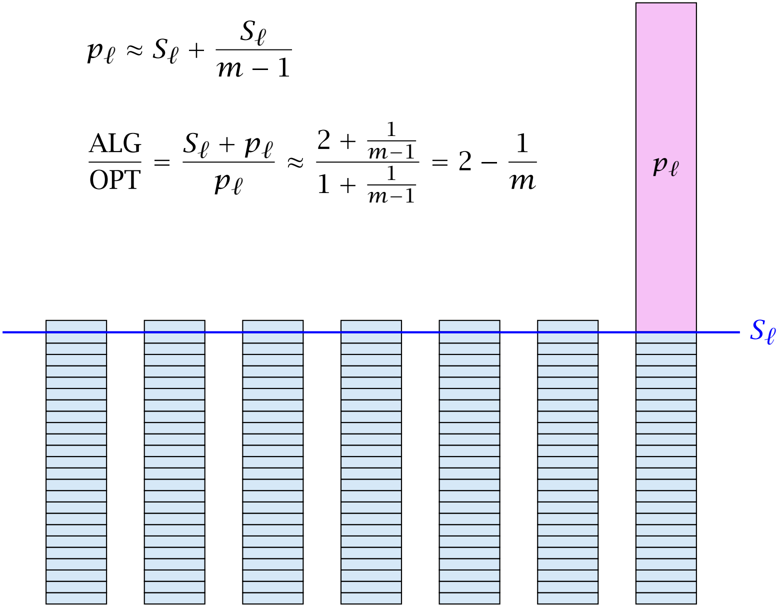
Hence, the length of the schedule is at most

$$p_\ell + \frac{1}{m} \sum_{j \ne \ell} p_j = (1 - \frac{1}{m}) p_\ell + \frac{1}{m} \sum_j p_j \le (2 - \frac{1}{m}) C^*_{\max}$$

A Tight Example

$$p_\ell \approx S_\ell + \frac{S_\ell}{m-1}$$

$$\frac{\text{ALG}}{\text{OPT}} = \frac{S_\ell + p_\ell}{p_\ell} \approx \frac{2 + \frac{1}{m-1}}{1 + \frac{1}{m-1}} = 2 - \frac{1}{m}$$



A Greedy Strategy

List Scheduling:
Order all processes in a list. When a machine runs empty assign the next yet unprocessed job to it.

Alternatively:
Consider processes in some order. Assign the i-th process to the least loaded machine.

It is easy to see that the result of these greedy strategies fulfill the local optimally condition of our local search algorithm. Hence, these also give 2-approximations.

A Greedy Strategy

List Scheduling:
Order all processes in a list. When a machine runs empty assign the next yet unprocessed job to it.

Alternatively:
Consider processes in some order. Assign the i-th process to the least loaded machine.

It is easy to see that the result of these greedy strategies fulfill the local optimally condition of our local search algorithm. Hence, these also give 2-approximations.

A Greedy Strategy

List Scheduling:
Order all processes in a list. When a machine runs empty assign the next yet unprocessed job to it.

Alternatively:
Consider processes in some order. Assign the i-th process to the least loaded machine.

It is easy to see that the result of these greedy strategies fulfill the local optimally condition of our local search algorithm. Hence, these also give 2-approximations.

A Greedy Strategy

List Scheduling:

Order all processes in a list. When a machine runs empty assign the next yet unprocessed job to it.

Alternatively:

Consider processes in some order. Assign the i-th process to the least loaded machine.

It is easy to see that the result of these greedy strategies fulfill the local optimally condition of our local search algorithm. Hence, these also give 2-approximations.

A Greedy Strategy

Lemma 60
If we order the list according to non-increasing processing times the approximation guarantee of the list scheduling strategy improves to $4/3$.

Proof:

- Let $p_1 \geq \cdots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- Wlog. the last job to finish is n (otw. deleting this job gives another counter-example with fewer jobs).
- If $p_n \leq C^*_{\max}/3$ the previous analysis gives us a schedule length of at most

$$C^*_{\max} + p_n \leq \frac{4}{3}C^*_{\max} \ .$$

  Hence, $p_n > C^*_{\max}/3$.
- This means that all jobs must have a processing time $> C^*_{\max}/3$.
- But then any machine in the optimum schedule can handle at most two jobs.
- For such instances Longest-Processing-Time-First is optimal.

Proof:

- Let $p_1 \geq \cdots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- Wlog. the last job to finish is n (otw. deleting this job gives another counter-example with fewer jobs).
- If $p_n \leq C^*_{\max}/3$ the previous analysis gives us a schedule length of at most

$$C^*_{\max} + p_n \leq \frac{4}{3} C^*_{\max} \ .$$

  Hence, $p_n > C^*_{\max}/3$.
- This means that all jobs must have a processing time $> C^*_{\max}/3$.
- But then any machine in the optimum schedule can handle at most two jobs.
- For such instances Longest-Processing-Time-First is optimal.

Proof:

- Let $p_1 \geq \cdots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- Wlog. the last job to finish is n (otw. deleting this job gives another counter-example with fewer jobs).
- If $p_n \leq C^*_{\max}/3$ the previous analysis gives us a schedule length of at most

$$C^*_{\max} + p_n \leq \frac{4}{3} C^*_{\max} \ .$$

- Hence, $p_n > C^*_{\max}/3$.
- This means that all jobs must have a processing time $> C^*_{\max}/3$.
- But then any machine in the optimum schedule can handle at most two jobs.
- For such instances Longest-Processing-Time-First is optimal.

Proof:

- Let $p_1 \geq \cdots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- Wlog. the last job to finish is n (otw. deleting this job gives another counter-example with fewer jobs).
- If $p_n \leq C^*_{\max}/3$ the previous analysis gives us a schedule length of at most

$$C^*_{\max} + p_n \leq \frac{4}{3} C^*_{\max} \enspace .$$

  Hence, $p_n > C^*_{\max}/3$.
- This means that all jobs must have a processing time $> C^*_{\max}/3$.
- But then any machine in the optimum schedule can handle at most two jobs.
- For such instances Longest-Processing-Time-First is optimal.

Proof:

- Let $p_1 \geq \cdots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- Wlog. the last job to finish is n (otw. deleting this job gives another counter-example with fewer jobs).
- If $p_n \leq C^*_{\max}/3$ the previous analysis gives us a schedule length of at most

$$C^*_{\max} + p_n \leq \frac{4}{3} C^*_{\max} \enspace .$$

  Hence, $p_n > C^*_{\max}/3$.
- This means that all jobs must have a processing time $> C^*_{\max}/3$.
- But then any machine in the optimum schedule can handle at most two jobs.
- For such instances Longest-Processing-Time-First is optimal.

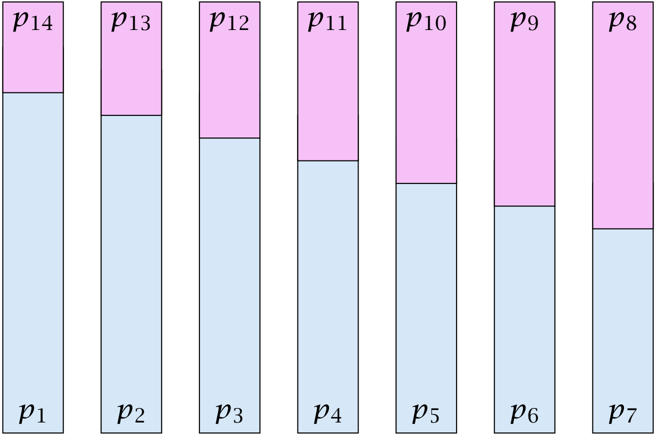
Proof:

- Let $p_1 \geq \cdots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- Wlog. the last job to finish is n (otw. deleting this job gives another counter-example with fewer jobs).
- If $p_n \leq C^*_{\max}/3$ the previous analysis gives us a schedule length of at most

$$C^*_{\max} + p_n \leq \frac{4}{3} C^*_{\max} \enspace .$$

  Hence, $p_n > C^*_{\max}/3$.
- This means that all jobs must have a processing time $> C^*_{\max}/3$.
- But then any machine in the optimum schedule can handle at most two jobs.
- For such instances Longest-Processing-Time-First is optimal.

When in an optimal solution a machine can have at most 2 jobs the optimal solution looks as follows.



- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

- We can assume that one machine schedules p_1 and p_n (the largest and smallest job).
- If not assume wlog. that p_1 is scheduled on machine A and p_n on machine B.
- Let p_A and p_B be the other job scheduled on A and B, respectively.
- $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling p_1 and p_n on one machine and p_A and p_B on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

16 Rounding Data + Dynamic Programming

Knapsack:
Given a set of items $\{1, \ldots, n\}$, where the i-th item has weight $w_i \in \mathbb{N}$ and profit $p_i \in \mathbb{N}$, and given a threshold W. Find a subset $I \subseteq \{1, \ldots, n\}$ of items of total weight at most W such that the profit is maximized (we can assume each $w_i \leq W$).

$$\begin{array}{lrcl} \max & \sum_{i=1}^{n} p_i x_i & & \\ \text{s.t.} & \sum_{i=1}^{n} w_i x_i & \leq & W \\ \forall i \in \{1, \ldots, n\} & x_i & \in & \{0, 1\} \end{array}$$

16 Rounding Data + Dynamic Programming

Knapsack:
Given a set of items $\{1, \ldots, n\}$, where the i-th item has weight $w_i \in \mathbb{N}$ and profit $p_i \in \mathbb{N}$, and given a threshold W. Find a subset $I \subseteq \{1, \ldots, n\}$ of items of total weight at most W such that the profit is maximized (we can assume each $w_i \leq W$).

$$\begin{array}{llrcl} \max & & \sum_{i=1}^{n} p_i x_i & & \\ \text{s.t.} & & \sum_{i=1}^{n} w_i x_i & \leq & W \\ & \forall i \in \{1, \ldots, n\} & x_i & \in & \{0, 1\} \end{array}$$

16 Rounding Data + Dynamic Programming

Algorithm 1 Knapsack

$A(1) \leftarrow [(0,0),(p_1,w_1)]$
for $j \leftarrow 2$ to n **do**
    $A(j) \leftarrow A(j-1)$
    for each $(p,w) \in A(j-1)$ **do**
        if $w + w_j \le W$ **then**
            add $(p+p_j, w+w_j)$ to $A(j)$
        remove dominated pairs from $A(j)$
return $\max_{(p,w)\in A(n)} p$

The running time is $\mathcal{O}(n \cdot \min\{W,P\})$, where $P = \sum_i p_i$ is the total profit of all items. This is only pseudo-polynomial.

16 Rounding Data + Dynamic Programming

Definition 61
An algorithm is said to have pseudo-polynomial running time if the running time is polynomial when the numerical part of the input is encoded in unary.

16 Rounding Data + Dynamic Programming

- Let M be the maximum profit of an element.

16 Rounding Data + Dynamic Programming

- Let M be the maximum profit of an element.
- Set $\mu := \epsilon M/n$.

16 Rounding Data + Dynamic Programming

- Let M be the maximum profit of an element.
- Set $\mu := \epsilon M/n$.
- Set $p_i' := \lfloor p_i/\mu \rfloor$ for all i.

16 Rounding Data + Dynamic Programming

- Let M be the maximum profit of an element.
- Set $\mu := \epsilon M/n$.
- Set $p'_i := \lfloor p_i/\mu \rfloor$ for all i.
- Run the dynamic programming algorithm on this revised instance.

16 Rounding Data + Dynamic Programming

- Let M be the maximum profit of an element.
- Set $\mu := \epsilon M/n$.
- Set $p_i' := \lfloor p_i/\mu \rfloor$ for all i.
- Run the dynamic programming algorithm on this revised instance.

Running time is at most

$$\mathcal{O}(nP')$$

16 Rounding Data + Dynamic Programming

- Let M be the maximum profit of an element.
- Set $\mu := \epsilon M/n$.
- Set $p'_i := \lfloor p_i/\mu \rfloor$ for all i.
- Run the dynamic programming algorithm on this revised instance.

Running time is at most

$$\mathcal{O}(nP') = \mathcal{O}\Big(n \sum_i p'_i\Big)$$

16 Rounding Data + Dynamic Programming

- Let M be the maximum profit of an element.
- Set $\mu := \epsilon M/n$.
- Set $p'_i := \lfloor p_i/\mu \rfloor$ for all i.
- Run the dynamic programming algorithm on this revised instance.

Running time is at most

$$\mathcal{O}(nP') = \mathcal{O}\Big(n\sum_i p'_i\Big) = \mathcal{O}\Big(n\sum_i \Big\lfloor \frac{p_i}{\epsilon M/n}\Big\rfloor\Big)$$

16 Rounding Data + Dynamic Programming

- Let M be the maximum profit of an element.
- Set $\mu := \epsilon M/n$.
- Set $p'_i := \lfloor p_i/\mu \rfloor$ for all i.
- Run the dynamic programming algorithm on this revised instance.

Running time is at most

$$\mathcal{O}(nP') = \mathcal{O}\Big(n\sum_i p'_i\Big) = \mathcal{O}\Big(n\sum_i \Big\lfloor \frac{p_i}{\epsilon M/n} \Big\rfloor\Big) \le \mathcal{O}\Big(\frac{n^3}{\epsilon}\Big) \ .$$

16 Rounding Data + Dynamic Programming

Let S be the set of items returned by the algorithm, and let O be an optimum set of items.

$$\sum_{i\in S} p_i$$

16 Rounding Data + Dynamic Programming

Let S be the set of items returned by the algorithm, and let O be an optimum set of items.

$$\sum_{i \in S} p_i \geq \mu \sum_{i \in S} p_i'$$

16 Rounding Data + Dynamic Programming

Let S be the set of items returned by the algorithm, and let O be an optimum set of items.

$$\sum_{i\in S} p_i \geq \mu \sum_{i\in S} p'_i$$
$$\geq \mu \sum_{i\in O} p'_i$$

16 Rounding Data + Dynamic Programming

Let S be the set of items returned by the algorithm, and let O be an optimum set of items.

$$\begin{aligned}\sum_{i\in S} p_i &\geq \mu \sum_{i\in S} p_i' \\ &\geq \mu \sum_{i\in O} p_i' \\ &\geq \sum_{i\in O} p_i - |O|\mu\end{aligned}$$

16 Rounding Data + Dynamic Programming

Let S be the set of items returned by the algorithm, and let O be an optimum set of items.

$$\begin{aligned}
\sum_{i\in S} p_i &\geq \mu \sum_{i\in S} p_i' \\
&\geq \mu \sum_{i\in O} p_i' \\
&\geq \sum_{i\in O} p_i - |O|\mu \\
&\geq \sum_{i\in O} p_i - n\mu
\end{aligned}$$

16 Rounding Data + Dynamic Programming

Let S be the set of items returned by the algorithm, and let O be an optimum set of items.

$$\begin{aligned}\sum_{i\in S} p_i &\geq \mu \sum_{i\in S} p_i' \\ &\geq \mu \sum_{i\in O} p_i' \\ &\geq \sum_{i\in O} p_i - |O|\mu \\ &\geq \sum_{i\in O} p_i - n\mu \\ &= \sum_{i\in O} p_i - \epsilon M\end{aligned}$$

16 Rounding Data + Dynamic Programming

Let S be the set of items returned by the algorithm, and let O be an optimum set of items.

$$\begin{aligned}\sum_{i\in S} p_i &\ge \mu \sum_{i\in S} p_i' \\ &\ge \mu \sum_{i\in O} p_i' \\ &\ge \sum_{i\in O} p_i - |O|\mu \\ &\ge \sum_{i\in O} p_i - n\mu \\ &= \sum_{i\in O} p_i - \epsilon M \\ &\ge (1-\epsilon)\mathrm{OPT} \ .\end{aligned}$$

Scheduling Revisited

The previous analysis of the scheduling algorithm gave a makespan of

$$\frac{1}{m}\sum_{j\neq\ell} p_j + p_\ell$$

where ℓ is the last job to complete.

Scheduling Revisited

The previous analysis of the scheduling algorithm gave a makespan of

$$\frac{1}{m}\sum_{j\neq\ell} p_j + p_\ell$$

where ℓ is the last job to complete.

Together with the obervation that if each $p_i \geq \frac{1}{3}C^*_{\max}$ then LPT is optimal this gave a $4/3$-approximation.

16.2 Scheduling Revisited

Partition the input into long jobs and short jobs.

16.2 Scheduling Revisited

Partition the input into long jobs and short jobs.

A job j is called short if

$$p_j \leq \frac{1}{km} \sum_i p_i$$

16.2 Scheduling Revisited

Partition the input into long jobs and short jobs.

A job j is called short if

$$p_j \leq \frac{1}{km} \sum_i p_i$$

Idea:

1. Find the optimum Makespan for the long jobs by brute force.

16.2 Scheduling Revisited

Partition the input into long jobs and short jobs.

A job j is called short if

$$p_j \leq \frac{1}{km} \sum_i p_i$$

Idea:

1. Find the optimum Makespan for the long jobs by brute force.
2. Then use the list scheduling algorithm for the short jobs, always assigning the next job to the least loaded machine.

We still have the inequality

$$\frac{1}{m}\sum_{j\neq\ell} p_j + p_\ell$$

where ℓ is the last job (this only requires that all machines are busy before time S_ℓ).

We still have the inequality

$$\frac{1}{m}\sum_{j\neq\ell} p_j + p_\ell$$

where ℓ is the last job (this only requires that all machines are busy before time S_ℓ).

If ℓ is a long job, then the schedule must be optimal, as it consists of an optimal schedule of long jobs plus a schedule for short jobs.

We still have the inequality

$$\frac{1}{m}\sum_{j\neq\ell} p_j + p_\ell$$

where ℓ is the last job (this only requires that all machines are busy before time S_ℓ).

If ℓ is a long job, then the schedule must be optimal, as it consists of an optimal schedule of long jobs plus a schedule for short jobs.

If ℓ is a short job its length is at most

$$p_\ell \leq \sum_j p_j/(mk)$$

which is at most $C^*_{\max}/k$.

Hence we get a schedule of length at most

$$\left(1 + \frac{1}{k}\right) C^*_{\max}$$

There are at most km long jobs. Hence, the number of possibilities of scheduling these jobs on m machines is at most m^{km}, which is constant if m is constant. Hence, it is easy to implement the algorithm in polynomial time.

Theorem 62

The above algorithm gives a polynomial time approximation scheme (PTAS) for the problem of scheduling n jobs on m identical machines if m is constant.

We choose $k = \lceil \frac{1}{\epsilon} \rceil$.

Hence we get a schedule of length at most

$$\left(1+\frac{1}{k}\right)C^*_{\max}$$

There are at most km long jobs. Hence, the number of possibilities of scheduling these jobs on m machines is at most m^{km}, which is constant if m is constant. Hence, it is easy to implement the algorithm in polynomial time.

Theorem 62

The above algorithm gives a polynomial time approximation scheme (PTAS) for the problem of scheduling n jobs on m identical machines if m is constant.

We choose $k = \lceil \frac{1}{\epsilon} \rceil$.

Hence we get a schedule of length at most

$$\left(1 + \frac{1}{k}\right) C^*_{\max}$$

There are at most km long jobs. Hence, the number of possibilities of scheduling these jobs on m machines is at most m^{km}, which is constant if m is constant. Hence, it is easy to implement the algorithm in polynomial time.

Theorem 62

The above algorithm gives a polynomial time approximation scheme (PTAS) for the problem of scheduling n jobs on m identical machines if m is constant.

We choose $k = \lceil \frac{1}{\epsilon} \rceil$.

How to get rid of the requirement that m is constant?

We first design an algorithm that works as follows:
On input of T it either finds a schedule of length $(1 + \frac{1}{k})T$ or certifies that no schedule of length at most T exists (assume $T \geq \frac{1}{m}\sum_j p_j$).

We partition the jobs into long jobs and short jobs:

- A job is long if its size is larger than T/k.
- Otw. it is a short job.

How to get rid of the requirement that m is constant?

We first design an algorithm that works as follows:
On input of T it either finds a schedule of length $(1 + \frac{1}{k})T$ or certifies that no schedule of length at most T exists (assume $T \geq \frac{1}{m}\sum_j p_j$).

We partition the jobs into long jobs and short jobs:

- A job is long if its size is larger than T/k.
- Otw. it is a short job.

How to get rid of the requirement that m is constant?

We first design an algorithm that works as follows:
On input of T it either finds a schedule of length $(1+\frac{1}{k})T$ or certifies that no schedule of length at most T exists (assume $T \geq \frac{1}{m}\sum_j p_j$).

We partition the jobs into long jobs and short jobs:

- A job is long if its size is larger than T/k.
- Otw. it is a short job.

How to get rid of the requirement that m is constant?

We first design an algorithm that works as follows:
On input of T it either finds a schedule of length $(1 + \frac{1}{k})T$ or certifies that no schedule of length at most T exists (assume $T \geq \frac{1}{m}\sum_j p_j$).

We partition the jobs into long jobs and short jobs:

- A job is long if its size is larger than T/k.
- Otw. it is a short job.

- We round all long jobs down to multiples of T/k^2.
- For these rounded sizes we first find an optimal schedule.
- If this schedule does not have length at most T we conclude that also the original sizes don't allow such a schedule.
- If we have a good schedule we extend it by adding the short jobs according to the LPT rule.

- We round all long jobs down to multiples of T/k^2.
- For these rounded sizes we first find an optimal schedule.
- If this schedule does not have length at most T we conclude that also the original sizes don't allow such a schedule.
- If we have a good schedule we extend it by adding the short jobs according to the LPT rule.

- We round all long jobs down to multiples of T/k^2.
- For these rounded sizes we first find an optimal schedule.
- If this schedule does not have length at most T we conclude that also the original sizes don't allow such a schedule.
- If we have a good schedule we extend it by adding the short jobs according to the LPT rule.

- We round all long jobs down to multiples of T/k^2.
- For these rounded sizes we first find an optimal schedule.
- If this schedule does not have length at most T we conclude that also the original sizes don't allow such a schedule.
- If we have a good schedule we extend it by adding the short jobs according to the LPT rule.

After the first phase the rounded sizes of the long jobs assigned to a machine add up to at most T.

There can be at most k (long) jobs assigned to a machine as otw. their rounded sizes would add up to more than T (note that the rounded size of a long job is at least T/k).

Since, jobs had been rounded to multiples of T/k^2 going from rounded sizes to original sizes gives that the Makespan is at most

$$\left(1+\frac{1}{k}\right)T \enspace .$$

After the first phase the rounded sizes of the long jobs assigned to a machine add up to at most T.

There can be at most k (long) jobs assigned to a machine as otw. their rounded sizes would add up to more than T (note that the rounded size of a long job is at least T/k).

Since, jobs had been rounded to multiples of T/k^2 going from rounded sizes to original sizes gives that the Makespan is at most

$$\left(1+\frac{1}{k}\right)T \enspace .$$

After the first phase the rounded sizes of the long jobs assigned to a machine add up to at most T.

There can be at most k (long) jobs assigned to a machine as otw. their rounded sizes would add up to more than T (note that the rounded size of a long job is at least T/k).

Since, jobs had been rounded to multiples of T/k^2 going from rounded sizes to original sizes gives that the Makespan is at most

$$\left(1+\frac{1}{k}\right)T \ .$$

During the second phase there always must exist a machine with load at most T, since T is larger than the average load.

Assigning the current (short) job to such a machine gives that the new load is at most

$$T + \frac{T}{k} \le \left(1 + \frac{1}{k}\right)T \ .$$

During the second phase there always must exist a machine with load at most T, since T is larger than the average load. Assigning the current (short) job to such a machine gives that the new load is at most

$$T + \frac{T}{k} \leq \left(1 + \frac{1}{k}\right)T \ .$$

Running Time for scheduling large jobs: There should not be a job with rounded size more than T as otw. the problem becomes trivial.

Hence, any large job has rounded size of $\frac{i}{k^2}T$ for $i \in \{k, \ldots, k^2\}$. Therefore the number of different inputs is at most n^{k^2} (described by a vector of length k^2 where, the i-th entry describes the number of jobs of size $\frac{i}{k^2}T$). This is polynomial.

The schedule/configuration of a particular machine x can be described by a vector of length k^2 where the i-th entry describes the number of jobs of rounded size $\frac{i}{k^2}T$ assigned to x. There are only $(k+1)^{k^2}$ different vectors.

This means there are a constant number of different machine configurations.

Running Time for scheduling large jobs: There should not be a job with rounded size more than T as otw. the problem becomes trivial.

Hence, any large job has rounded size of $\frac{i}{k^2}T$ for $i \in \{k, \dots, k^2\}$. Therefore the number of different inputs is at most n^{k^2} (described by a vector of length k^2 where, the i-th entry describes the number of jobs of size $\frac{i}{k^2}T$). This is polynomial.

The schedule/configuration of a particular machine x can be described by a vector of length k^2 where the i-th entry describes the number of jobs of rounded size $\frac{i}{k^2}T$ assigned to x. There are only $(k+1)^{k^2}$ different vectors.

This means there are a constant number of different machine configurations.

Running Time for scheduling large jobs: There should not be a job with rounded size more than T as otw. the problem becomes trivial.

Hence, any large job has rounded size of $\frac{i}{k^2}T$ for $i \in \{k, \ldots, k^2\}$. Therefore the number of different inputs is at most n^{k^2} (described by a vector of length k^2 where, the i-th entry describes the number of jobs of size $\frac{i}{k^2}T$). This is polynomial.

The schedule/configuration of a particular machine x can be described by a vector of length k^2 where the i-th entry describes the number of jobs of rounded size $\frac{i}{k^2}T$ assigned to x. There are only $(k+1)^{k^2}$ different vectors.

This means there are a constant number of different machine configurations.

Running Time for scheduling large jobs: There should not be a job with rounded size more than T as otw. the problem becomes trivial.

Hence, any large job has rounded size of $\frac{i}{k^2}T$ for $i \in \{k, \ldots, k^2\}$. Therefore the number of different inputs is at most n^{k^2} (described by a vector of length k^2 where, the i-th entry describes the number of jobs of size $\frac{i}{k^2}T$). This is polynomial.

The schedule/configuration of a particular machine x can be described by a vector of length k^2 where the i-th entry describes the number of jobs of rounded size $\frac{i}{k^2}T$ assigned to x. There are only $(k+1)^{k^2}$ different vectors.

This means there are a constant number of different machine configurations.

Let $\mathrm{OPT}(n_1,\ldots,n_{k^2})$ be the number of machines that are required to schedule input vector $(n_1,\ldots,n_{k^2})$ with Makespan at most T.

If $\mathrm{OPT}(n_1,\ldots,n_{k^2}) \le m$ we can schedule the input.

We have

$$\mathrm{OPT}(n_1,\ldots,n_{k^2}) = \begin{cases} 0 & (n_1,\ldots,n_{k^2}) = 0 \\ 1 + \min\limits_{(s_1,\ldots,s_{k^2})\in C} \mathrm{OPT}(n_1 - s_1,\ldots,n_{k^2} - s_{k^2}) & (n_1,\ldots,n_{k^2}) \gneq 0 \\ \infty & \text{otw.} \end{cases}$$

where C is the set of all configurations.

Hence, the running time is roughly $(k+1)^{k^2} n^{k^2} \approx (nk)^{k^2}$.

Let $\mathrm{OPT}(n_1,\ldots,n_{k^2})$ be the number of machines that are required to schedule input vector $(n_1,\ldots,n_{k^2})$ with Makespan at most T.

If $\mathrm{OPT}(n_1,\ldots,n_{k^2}) \le m$ we can schedule the input.

We have

$$\mathrm{OPT}(n_1,\ldots,n_{k^2}) = \begin{cases} 0 & (n_1,\ldots,n_{k^2}) = 0 \\ 1 + \min\limits_{(s_1,\ldots,s_{k^2})\in C} \mathrm{OPT}(n_1 - s_1,\ldots,n_{k^2} - s_{k^2}) & (n_1,\ldots,n_{k^2}) \gneq 0 \\ \infty & \text{otw.} \end{cases}$$

where C is the set of all configurations.

Hence, the running time is roughly $(k+1)^{k^2} n^{k^2} \approx (nk)^{k^2}$.

Let $\text{OPT}(n_1,\dots,n_{k^2})$ be the number of machines that are required to schedule input vector $(n_1,\dots,n_{k^2})$ with Makespan at most T.

If $\text{OPT}(n_1,\dots,n_{k^2}) \leq m$ we can schedule the input.

We have

$$\text{OPT}(n_1,\dots,n_{k^2}) = \begin{cases} 0 & (n_1,\dots,n_{k^2}) = 0 \\ 1 + \min\limits_{(s_1,\dots,s_{k^2})\in C} \text{OPT}(n_1 - s_1,\dots,n_{k^2} - s_{k^2}) & (n_1,\dots,n_{k^2}) \gneq 0 \\ \infty & \text{otw.} \end{cases}$$

where C is the set of all configurations.

Hence, the running time is roughly $(k+1)^{k^2} n^{k^2} \approx (nk)^{k^2}$.

Let $\mathrm{OPT}(n_1, \ldots, n_{k^2})$ be the number of machines that are required to schedule input vector $(n_1, \ldots, n_{k^2})$ with Makespan at most T.

If $\mathrm{OPT}(n_1, \ldots, n_{k^2}) \leq m$ we can schedule the input.

We have

$$\mathrm{OPT}(n_1, \ldots, n_{k^2}) = \begin{cases} 0 & (n_1, \ldots, n_{k^2}) = 0 \\ 1 + \min\limits_{(s_1, \ldots, s_{k^2}) \in C} \mathrm{OPT}(n_1 - s_1, \ldots, n_{k^2} - s_{k^2}) & (n_1, \ldots, n_{k^2}) \gneq 0 \\ \infty & \text{otw.} \end{cases}$$

where C is the set of all configurations.

Hence, the running time is roughly $(k+1)^{k^2} n^{k^2} \approx (nk)^{k^2}$.

We can turn this into a PTAS by choosing $k = \lceil 1/\epsilon \rceil$ and using binary search. This gives a running time that is exponential in $1/\epsilon$.

Can we do better?
Scheduling on identical machines with the goal of minimizing Makespan is a strongly NP-complete problem.

Theorem 63
There is no FPTAS for problems that are strongly NP-hard.

We can turn this into a PTAS by choosing $k = \lceil 1/\epsilon \rceil$ and using binary search. This gives a running time that is exponential in $1/\epsilon$.

Can we do better?

Scheduling on identical machines with the goal of minimizing Makespan is a strongly NP-complete problem.

Theorem 63

There is no FPTAS for problems that are strongly NP-hard.

We can turn this into a PTAS by choosing $k = \lceil 1/\epsilon \rceil$ and using binary search. This gives a running time that is exponential in $1/\epsilon$.

Can we do better?

Scheduling on identical machines with the goal of minimizing Makespan is a strongly NP-complete problem.

Theorem 63

There is no FPTAS for problems that are strongly NP-hard.

We can turn this into a PTAS by choosing $k = \lceil 1/\epsilon \rceil$ and using binary search. This gives a running time that is exponential in $1/\epsilon$.

Can we do better?

Scheduling on identical machines with the goal of minimizing Makespan is a strongly NP-complete problem.

Theorem 63

There is no FPTAS for problems that are strongly NP-hard.

- Suppose we have an instance with polynomially bounded processing times $p_i \leq q(n)$
- We set $k := \lceil 2nq(n) \rceil \geq 2\,\mathrm{OPT}$
- Then

$$\mathrm{ALG} \leq \left(1 + \frac{1}{k}\right)\mathrm{OPT} \leq \mathrm{OPT} + \frac{1}{2}$$

- But this means that the algorithm computes the optimal solution as the optimum is integral.
- This means we can solve problem instances if processing times are polynomially bounded
- Running time is $\mathcal{O}(\mathrm{poly}(n,k)) = \mathcal{O}(\mathrm{poly}(n))$
- For strongly NP-complete problems this is not possible unless P=NP

- Suppose we have an instance with polynomially bounded processing times $p_i \leq q(n)$
- We set $k := \lceil 2nq(n) \rceil \geq 2\,\mathrm{OPT}$
- Then

$$\mathrm{ALG} \leq \left(1 + \frac{1}{k}\right)\mathrm{OPT} \leq \mathrm{OPT} + \frac{1}{2}$$

- But this means that the algorithm computes the optimal solution as the optimum is integral.
- This means we can solve problem instances if processing times are polynomially bounded
- Running time is $\mathcal{O}(\mathrm{poly}(n,k)) = \mathcal{O}(\mathrm{poly}(n))$
- For strongly NP-complete problems this is not possible unless P=NP

- Suppose we have an instance with polynomially bounded processing times $p_i \le q(n)$
- We set $k := \lceil 2nq(n) \rceil \ge 2\,\mathrm{OPT}$
- Then
$$\mathrm{ALG} \le \left(1 + \frac{1}{k}\right)\mathrm{OPT} \le \mathrm{OPT} + \frac{1}{2}$$
- But this means that the algorithm computes the optimal solution as the optimum is integral.
- This means we can solve problem instances if processing times are polynomially bounded
- Running time is $\mathcal{O}(\mathrm{poly}(n,k)) = \mathcal{O}(\mathrm{poly}(n))$
- For strongly NP-complete problems this is not possible unless P=NP

- Suppose we have an instance with polynomially bounded processing times $p_i \leq q(n)$
- We set $k := \lceil 2nq(n) \rceil \geq 2\,\mathrm{OPT}$
- Then
$$\mathrm{ALG} \leq \left(1 + \frac{1}{k}\right)\mathrm{OPT} \leq \mathrm{OPT} + \frac{1}{2}$$
- But this means that the algorithm computes the optimal solution as the optimum is integral.
- This means we can solve problem instances if processing times are polynomially bounded
- Running time is $\mathcal{O}(\mathrm{poly}(n,k)) = \mathcal{O}(\mathrm{poly}(n))$
- For strongly NP-complete problems this is not possible unless P=NP

- Suppose we have an instance with polynomially bounded processing times $p_i \le q(n)$
- We set $k := \lceil 2nq(n) \rceil \ge 2\,\mathrm{OPT}$
- Then

$$\mathrm{ALG} \le \left(1 + \frac{1}{k}\right)\mathrm{OPT} \le \mathrm{OPT} + \frac{1}{2}$$

- But this means that the algorithm computes the optimal solution as the optimum is integral.
- This means we can solve problem instances if processing times are polynomially bounded
- Running time is $\mathcal{O}(\mathrm{poly}(n,k)) = \mathcal{O}(\mathrm{poly}(n))$
- For strongly NP-complete problems this is not possible unless P=NP

- Suppose we have an instance with polynomially bounded processing times $p_i \leq q(n)$
- We set $k := \lceil 2nq(n) \rceil \geq 2\,\mathrm{OPT}$
- Then

$$\mathrm{ALG} \leq \left(1 + \frac{1}{k}\right)\mathrm{OPT} \leq \mathrm{OPT} + \frac{1}{2}$$

- But this means that the algorithm computes the optimal solution as the optimum is integral.
- This means we can solve problem instances if processing times are polynomially bounded
- Running time is $\mathcal{O}(\mathrm{poly}(n,k)) = \mathcal{O}(\mathrm{poly}(n))$
- For strongly NP-complete problems this is not possible unless P=NP

- Suppose we have an instance with polynomially bounded processing times $p_i \le q(n)$
- We set $k := \lceil 2nq(n) \rceil \ge 2\,\mathrm{OPT}$
- Then
$$\mathrm{ALG} \le \left(1 + \frac{1}{k}\right)\mathrm{OPT} \le \mathrm{OPT} + \frac{1}{2}$$
- But this means that the algorithm computes the optimal solution as the optimum is integral.
- This means we can solve problem instances if processing times are polynomially bounded
- Running time is $\mathcal{O}(\mathrm{poly}(n,k)) = \mathcal{O}(\mathrm{poly}(n))$
- For strongly NP-complete problems this is not possible unless P=NP

More General

Let $\mathrm{OPT}(n_1, \dots, n_A)$ be the number of machines that are required to schedule input vector $(n_1, \dots, n_A)$ with Makespan at most T (A: number of different sizes).

If $\mathrm{OPT}(n_1, \dots, n_A) \le m$ we can schedule the input.

$$\mathrm{OPT}(n_1, \dots, n_A) = \begin{cases} 0 & (n_1, \dots, n_A) = 0 \\ 1 + \min\limits_{(s_1, \dots, s_A) \in C} \mathrm{OPT}(n_1 - s_1, \dots, n_A - s_A) & (n_1, \dots, n_A) \gneq 0 \\ \infty & \text{otw.} \end{cases}$$

where C is the set of all configurations.

$|C| \le (B+1)^A$, where B is the number of jobs that possibly can fit on the same machine.

The running time is then $O((B+1)^A n^A)$ because the dynamic programming table has just n^A entries.

More General

Let $\mathrm{OPT}(n_1, \dots, n_A)$ be the number of machines that are required to schedule input vector $(n_1, \dots, n_A)$ with Makespan at most T (A: number of different sizes).

If $\mathrm{OPT}(n_1, \dots, n_A) \leq m$ we can schedule the input.

$$\mathrm{OPT}(n_1, \dots, n_A) = \begin{cases} 0 & (n_1, \dots, n_A) = 0 \\ 1 + \min\limits_{(s_1,\dots,s_A) \in C} \mathrm{OPT}(n_1 - s_1, \dots, n_A - s_A) & (n_1, \dots, n_A) \gneq 0 \\ \infty & \text{otw.} \end{cases}$$

where C is the set of all configurations.

$|C| \leq (B+1)^A$, where B is the number of jobs that possibly can fit on the same machine.

The running time is then $O((B+1)^A n^A)$ because the dynamic programming table has just n^A entries.

More General

Let $\mathrm{OPT}(n_1, \dots, n_A)$ be the number of machines that are required to schedule input vector $(n_1, \dots, n_A)$ with Makespan at most T (A: number of different sizes).

If $\mathrm{OPT}(n_1, \dots, n_A) \leq m$ we can schedule the input.

$$\mathrm{OPT}(n_1, \dots, n_A) = \begin{cases} 0 & (n_1, \dots, n_A) = 0 \\ 1 + \min\limits_{(s_1, \dots, s_A) \in C} \mathrm{OPT}(n_1 - s_1, \dots, n_A - s_A) & (n_1, \dots, n_A) \gneq 0 \\ \infty & \text{otw.} \end{cases}$$

where C is the set of all configurations.

$|C| \leq (B+1)^A$, where B is the number of jobs that possibly can fit on the same machine.

The running time is then $O((B+1)^A n^A)$ because the dynamic programming table has just n^A entries.

Bin Packing

Given n items with sizes $s_1, \ldots, s_n$ where

$$1 > s_1 \geq \cdots \geq s_n > 0 \ .$$

Pack items into a minimum number of bins where each bin can hold items of total size at most 1.

Theorem 64

There is no ρ-approximation for Bin Packing with $\rho < 3/2$ unless $\mathrm{P} = \mathrm{NP}$.

Bin Packing

Given n items with sizes $s_1, \ldots, s_n$ where

$$1 > s_1 \geq \cdots \geq s_n > 0 \ .$$

Pack items into a minimum number of bins where each bin can hold items of total size at most 1.

Theorem 64

There is no ρ-approximation for Bin Packing with $\rho < 3/2$ unless $\mathrm{P} = \mathrm{NP}$.

Bin Packing

Proof

- In the partition problem we are given positive integers $b_1, \ldots, b_n$ with $B = \sum_i b_i$ even. Can we partition the integers into two sets S and T s.t.

$$\sum_{i \in S} b_i = \sum_{i \in T} b_i \quad ?$$

- We can solve this problem by setting $s_i := 2b_i/B$ and asking whether we can pack the resulting items into 2 bins or not.
- A ρ-approximation algorithm with $\rho < 3/2$ cannot output 3 or more bins when 2 are optimal.
- Hence, such an algorithm can solve Partition.

Bin Packing

Proof

- In the partition problem we are given positive integers $b_1, \dots, b_n$ with $B = \sum_i b_i$ even. Can we partition the integers into two sets S and T s.t.

$$\sum_{i \in S} b_i = \sum_{i \in T} b_i \quad ?$$

- We can solve this problem by setting $s_i := 2b_i/B$ and asking whether we can pack the resulting items into 2 bins or not.
- A ρ-approximation algorithm with $\rho < 3/2$ cannot output 3 or more bins when 2 are optimal.
- Hence, such an algorithm can solve Partition.

Bin Packing

Proof

- In the partition problem we are given positive integers $b_1, \dots, b_n$ with $B = \sum_i b_i$ even. Can we partition the integers into two sets S and T s.t.

$$\sum_{i \in S} b_i = \sum_{i \in T} b_i \quad ?$$

- We can solve this problem by setting $s_i := 2b_i/B$ and asking whether we can pack the resulting items into 2 bins or not.
- A ρ-approximation algorithm with $\rho < 3/2$ cannot output 3 or more bins when 2 are optimal.
- Hence, such an algorithm can solve Partition.

Bin Packing

Proof

- In the partition problem we are given positive integers $b_1, \dots, b_n$ with $B = \sum_i b_i$ even. Can we partition the integers into two sets S and T s.t.

$$\sum_{i \in S} b_i = \sum_{i \in T} b_i \quad ?$$

- We can solve this problem by setting $s_i := 2b_i/B$ and asking whether we can pack the resulting items into 2 bins or not.
- A ρ-approximation algorithm with $\rho < 3/2$ cannot output 3 or more bins when 2 are optimal.
- Hence, such an algorithm can solve Partition.

Bin Packing

Definition 65

An asymptotic polynomial-time approximation scheme (APTAS) is a family of algorithms $\{A_\epsilon\}$ along with a constant c such that A_ϵ returns a solution of value at most $(1+\epsilon)\mathrm{OPT}+c$ for minimization problems.

Note that for Set Cover or for Knapsack it makes no sense to differentiate between the notion of a PTAS or an APTAS because of scaling.

However, we will develop an APTAS for Bin Packing.

Bin Packing

Definition 65

An asymptotic polynomial-time approximation scheme (APTAS) is a family of algorithms $\{A_\epsilon\}$ along with a constant c such that A_ϵ returns a solution of value at most $(1+\epsilon)\mathrm{OPT}+c$ for minimization problems.

- Note that for Set Cover or for Knapsack it makes no sense to differentiate between the notion of a PTAS or an APTAS because of scaling.
- However, we will develop an APTAS for Bin Packing.

Bin Packing

Definition 65
An asymptotic polynomial-time approximation scheme (APTAS) is a family of algorithms $\{A_\epsilon\}$ along with a constant c such that A_ϵ returns a solution of value at most $(1+\epsilon)\mathrm{OPT}+c$ for minimization problems.

- Note that for Set Cover or for Knapsack it makes no sense to differentiate between the notion of a PTAS or an APTAS because of scaling.
- However, we will develop an APTAS for Bin Packing.

Bin Packing

Again we can differentiate between small and large items.

Lemma 66

Any packing of items into ℓ bins can be extended with items of size at most γ s.t. we use only $\max\{\ell, \frac{1}{1-\gamma}\mathrm{SIZE}(I) + 1\}$ bins, where $\mathrm{SIZE}(I) = \sum_i s_i$ is the sum of all item sizes.

- If after Greedy we use more than ℓ bins, all bins (apart from the last) must be full to at least $1-\gamma$.
- Hence, $r(1-\gamma) \le \mathrm{SIZE}(I)$ where r is the number of nearly-full bins.
- This gives the lemma.

Bin Packing

Again we can differentiate between small and large items.

Lemma 66

Any packing of items into ℓ bins can be extended with items of size at most γ s.t. we use only $\max\{\ell, \frac{1}{1-\gamma}\mathrm{SIZE}(I) + 1\}$ bins, where $\mathrm{SIZE}(I) = \sum_i s_i$ is the sum of all item sizes.

- If after Greedy we use more than ℓ bins, all bins (apart from the last) must be full to at least $1 - \gamma$.
- Hence, $r(1-\gamma) \leq \mathrm{SIZE}(I)$ where r is the number of nearly-full bins.
- This gives the lemma.

Bin Packing

Again we can differentiate between small and large items.

Lemma 66

Any packing of items into ℓ bins can be extended with items of size at most γ s.t. we use only $\max\{\ell, \frac{1}{1-\gamma}\mathrm{SIZE}(I)+1\}$ bins, where $\mathrm{SIZE}(I)=\sum_i s_i$ is the sum of all item sizes.

- If after Greedy we use more than ℓ bins, all bins (apart from the last) must be full to at least $1-\gamma$.
- Hence, $r(1-\gamma)\le \mathrm{SIZE}(I)$ where r is the number of nearly-full bins.
- This gives the lemma.

Bin Packing

Again we can differentiate between small and large items.

Lemma 66

Any packing of items into ℓ bins can be extended with items of size at most γ s.t. we use only $\max\{\ell, \frac{1}{1-\gamma}\text{SIZE}(I) + 1\}$ bins, where $\text{SIZE}(I) = \sum_i s_i$ is the sum of all item sizes.

- If after Greedy we use more than ℓ bins, all bins (apart from the last) must be full to at least $1 - \gamma$.
- Hence, $r(1-\gamma) \leq \text{SIZE}(I)$ where r is the number of nearly-full bins.
- This gives the lemma.

Choose $\gamma = \epsilon/2$. Then we either use ℓ bins or at most

$$\frac{1}{1-\epsilon/2} \cdot \mathrm{OPT} + 1 \leq (1+\epsilon) \cdot \mathrm{OPT} + 1$$

bins.

It remains to find an algorithm for the large items.

Bin Packing

Linear Grouping:

Generate an instance I' (for large items) as follows.

- Order large items according to size.
- Let the first k items belong to group 1; the following k items belong to group 2; etc.
- Delete items in the first group;
- Round items in the remaining groups to the size of the largest item in the group.

Bin Packing

Linear Grouping:

Generate an instance I' (for large items) as follows.

- Order large items according to size.
- Let the first k items belong to group 1; the following k items belong to group 2; etc.
- Delete items in the first group;
- Round items in the remaining groups to the size of the largest item in the group.

Bin Packing

Linear Grouping:

Generate an instance I' (for large items) as follows.

- Order large items according to size.
- Let the first k items belong to group 1; the following k items belong to group 2; etc.
- Delete items in the first group;
- Round items in the remaining groups to the size of the largest item in the group.

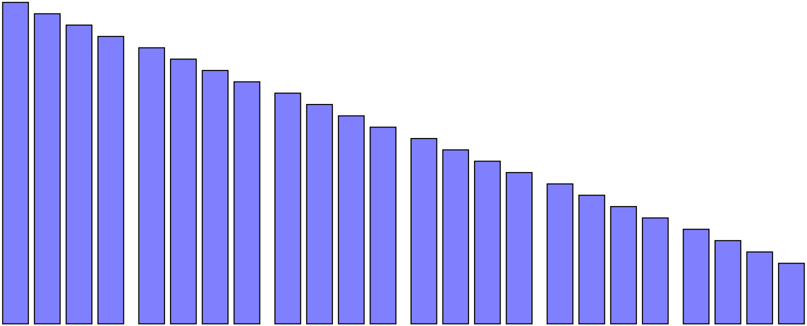
Bin Packing

Linear Grouping:

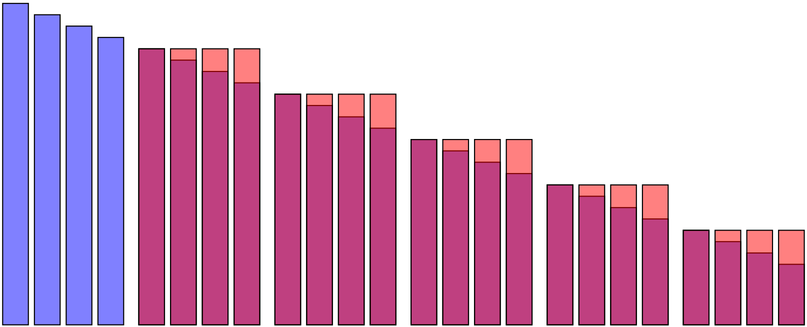
Generate an instance I' (for large items) as follows.

- Order large items according to size.
- Let the first k items belong to group 1; the following k items belong to group 2; etc.
- Delete items in the first group;
- Round items in the remaining groups to the size of the largest item in the group.

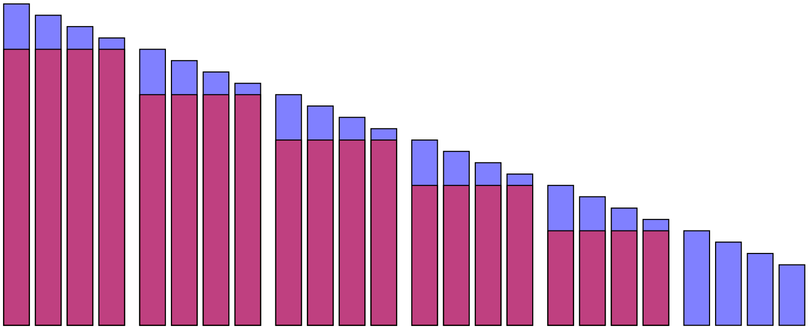
Linear Grouping



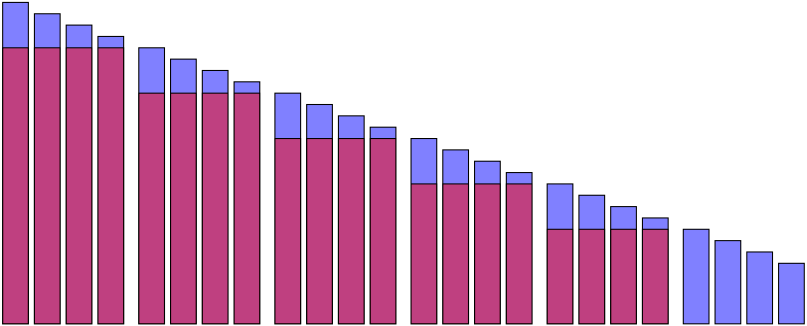
Linear Grouping



Linear Grouping



Linear Grouping



Lemma 67

$\mathrm{OPT}(I') \le \mathrm{OPT}(I) \le \mathrm{OPT}(I') + k$

Lemma 67

$\text{OPT}(I') \leq \text{OPT}(I) \leq \text{OPT}(I') + k$

Proof 1:

- Any bin packing for I gives a bin packing for I' as follows.
- Pack the items of group 2, where in the packing for I the items for group 1 have been packed;
- Pack the items of groups 3, where in the packing for I the items for group 2 have been packed;
- ...

Lemma 67

$\mathrm{OPT}(I') \leq \mathrm{OPT}(I) \leq \mathrm{OPT}(I') + k$

Proof 1:

- Any bin packing for I gives a bin packing for I' as follows.
- Pack the items of group 2, where in the packing for I the items for group 1 have been packed;
- Pack the items of groups 3, where in the packing for I the items for group 2 have been packed;
- ...

Lemma 67

$\mathrm{OPT}(I') \leq \mathrm{OPT}(I) \leq \mathrm{OPT}(I') + k$

Proof 1:

- Any bin packing for I gives a bin packing for I' as follows.
- Pack the items of group 2, where in the packing for I the items for group 1 have been packed;
- Pack the items of groups 3, where in the packing for I the items for group 2 have been packed;
- ...

Lemma 67

$\mathrm{OPT}(I') \le \mathrm{OPT}(I) \le \mathrm{OPT}(I') + k$

Proof 1:

- Any bin packing for I gives a bin packing for I' as follows.
- Pack the items of group 2, where in the packing for I the items for group 1 have been packed;
- Pack the items of groups 3, where in the packing for I the items for group 2 have been packed;
- ...

Lemma 68

$\text{OPT}(I') \leq \text{OPT}(I) \leq \text{OPT}(I') + k$

Proof 2:

- Any bin packing for I' gives a bin packing for I as follows.
- Pack the items of group 1 into k new bins;
- Pack the items of groups 2, where in the packing for I' the items for group 2 have been packed;
- ...

Lemma 68

$\mathrm{OPT}(I') \le \mathrm{OPT}(I) \le \mathrm{OPT}(I') + k$

Proof 2:

- Any bin packing for I' gives a bin packing for I as follows.
- Pack the items of group 1 into k new bins;
- Pack the items of groups 2, where in the packing for I' the items for group 2 have been packed;
- ...

Lemma 68

$\mathrm{OPT}(I') \le \mathrm{OPT}(I) \le \mathrm{OPT}(I') + k$

Proof 2:

- Any bin packing for I' gives a bin packing for I as follows.
- Pack the items of group 1 into k new bins;
- Pack the items of groups 2, where in the packing for I' the items for group 2 have been packed;
- ...

Lemma 68

$\mathrm{OPT}(I') \leq \mathrm{OPT}(I) \leq \mathrm{OPT}(I') + k$

Proof 2:

- Any bin packing for I' gives a bin packing for I as follows.
- Pack the items of group 1 into k new bins;
- Pack the items of groups 2, where in the packing for I' the items for group 2 have been packed;
- ...

Assume that our instance does not contain pieces smaller than $\epsilon/2$. Then $\mathrm{SIZE}(I) \geq \epsilon n/2$.

We set $k = \lfloor \epsilon \mathrm{SIZE}(I) \rfloor$.

Then $n/k \leq n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (here we used $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes ($4/\epsilon^2$) and at most a constant number ($2/\epsilon$) can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

- cost (for large items) at most

$$\mathrm{OPT}(I') + k \leq \mathrm{OPT}(I) + \epsilon \mathrm{SIZE}(I) \leq (1+\epsilon)\mathrm{OPT}(I)$$

- running time $\mathcal{O}((\frac{2}{\epsilon} n)^{4/\epsilon^2})$.

Assume that our instance does not contain pieces smaller than $\epsilon/2$. Then $\text{SIZE}(I) \geq \epsilon n/2$.

We set $k = \lfloor \epsilon \text{SIZE}(I) \rfloor$.

Then $n/k \leq n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (here we used $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes ($4/\epsilon^2$) and at most a constant number ($2/\epsilon$) can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

- cost (for large items) at most

$$\text{OPT}(I') + k \leq \text{OPT}(I) + \epsilon \text{SIZE}(I) \leq (1 + \epsilon)\text{OPT}(I)$$

- running time $\mathcal{O}((\frac{2}{\epsilon} n)^{4/\epsilon^2})$.

Assume that our instance does not contain pieces smaller than $\epsilon/2$. Then $\mathrm{SIZE}(I) \geq \epsilon n/2$.

We set $k = \lfloor \epsilon \mathrm{SIZE}(I) \rfloor$.

Then $n/k \leq n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (here we used $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes ($4/\epsilon^2$) and at most a constant number ($2/\epsilon$) can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

- cost (for large items) at most

$$\mathrm{OPT}(I') + k \leq \mathrm{OPT}(I) + \epsilon \mathrm{SIZE}(I) \leq (1+\epsilon)\mathrm{OPT}(I)$$

- running time $\mathcal{O}((\frac{2}{\epsilon} n)^{4/\epsilon^2})$.

Assume that our instance does not contain pieces smaller than $\epsilon/2$. Then $\text{SIZE}(I) \geq \epsilon n/2$.

We set $k = \lfloor \epsilon \text{SIZE}(I) \rfloor$.

Then $n/k \leq n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (here we used $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes $(4/\epsilon^2)$ and at most a constant number $(2/\epsilon)$ can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

- cost (for large items) at most

$$\text{OPT}(I') + k \leq \text{OPT}(I) + \epsilon \text{SIZE}(I) \leq (1+\epsilon)\text{OPT}(I)$$

- running time $\mathcal{O}((\frac{2}{\epsilon} n)^{4/\epsilon^2})$.

Assume that our instance does not contain pieces smaller than $\epsilon/2$. Then $\mathrm{SIZE}(I) \geq \epsilon n/2$.

We set $k = \lfloor \epsilon \mathrm{SIZE}(I) \rfloor$.

Then $n/k \leq n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (here we used $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes ($4/\epsilon^2$) and at most a constant number ($2/\epsilon$) can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

- cost (for large items) at most

$$\mathrm{OPT}(I') + k \leq \mathrm{OPT}(I) + \epsilon \mathrm{SIZE}(I) \leq (1+\epsilon)\mathrm{OPT}(I)$$

- running time $\mathcal{O}((\frac{2}{\epsilon} n)^{4/\epsilon^2})$.

Assume that our instance does not contain pieces smaller than $\epsilon/2$. Then $\text{SIZE}(I) \geq \epsilon n/2$.

We set $k = \lfloor \epsilon \text{SIZE}(I) \rfloor$.

Then $n/k \leq n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (here we used $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes ($4/\epsilon^2$) and at most a constant number ($2/\epsilon$) can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

- cost (for large items) at most

$$\text{OPT}(I') + k \leq \text{OPT}(I) + \epsilon \text{SIZE}(I) \leq (1+\epsilon)\text{OPT}(I)$$

- running time $\mathcal{O}((\frac{2}{\epsilon}n)^{4/\epsilon^2})$.

Can we do better?

In the following we show how to obtain a solution where the number of bins is only

$$\mathrm{OPT}(I) + \mathcal{O}(\log^2(\mathrm{SIZE}(I))) \ .$$

Note that this is usually better than a guarantee of

$$(1+\epsilon)\mathrm{OPT}(I) + 1 \ .$$

Can we do better?

In the following we show how to obtain a solution where the number of bins is only

$$\mathrm{OPT}(I) + \mathcal{O}(\log^2(\mathrm{SIZE}(I))) \ .$$

Note that this is usually better than a guarantee of

$$(1+\epsilon)\mathrm{OPT}(I) + 1 \ .$$

Can we do better?

In the following we show how to obtain a solution where the number of bins is only

$$\mathrm{OPT}(I) + \mathcal{O}(\log^2(\mathrm{SIZE}(I))) \ .$$

Note that this is usually better than a guarantee of

$$(1+\epsilon)\mathrm{OPT}(I) + 1 \ .$$

Configuration LP

Change of Notation:

- Group pieces of identical size.
- Let s_1 denote the largest size, and let b_1 denote the number of pieces of size s_1.
- s_2 is second largest size and b_2 number of pieces of size s_2;
- $\ldots$
- s_m smallest size and b_m number of pieces of size s_m.

Configuration LP

Change of Notation:

- Group pieces of identical size.
- Let s_1 denote the largest size, and let b_1 denote the number of pieces of size s_1.
- s_2 is second largest size and b_2 number of pieces of size s_2;
- ...
- s_m smallest size and b_m number of pieces of size s_m.

Configuration LP

Change of Notation:

- Group pieces of identical size.
- Let s_1 denote the largest size, and let b_1 denote the number of pieces of size s_1.
- s_2 is second largest size and b_2 number of pieces of size s_2;
- ...
- s_m smallest size and b_m number of pieces of size s_m.

Configuration LP

Change of Notation:

- Group pieces of identical size.
- Let s_1 denote the largest size, and let b_1 denote the number of pieces of size s_1.
- s_2 is second largest size and b_2 number of pieces of size s_2;
- ...
- s_m smallest size and b_m number of pieces of size s_m.

Configuration LP

Change of Notation:

- Group pieces of identical size.
- Let s_1 denote the largest size, and let b_1 denote the number of pieces of size s_1.
- s_2 is second largest size and b_2 number of pieces of size s_2;
- ...
- s_m smallest size and b_m number of pieces of size s_m.

Configuration LP

A possible packing of a bin can be described by an m-tuple $(t_1, \ldots, t_m)$, where t_i describes the number of pieces of size s_i. Clearly,

$$\sum_i t_i \cdot s_i \le 1 \ .$$

We call a vector that fulfills the above constraint a configuration.

Configuration LP

A possible packing of a bin can be described by an m-tuple $(t_1, \dots, t_m)$, where t_i describes the number of pieces of size s_i. Clearly,

$$\sum_i t_i \cdot s_i \leq 1 \ .$$

We call a vector that fulfills the above constraint a configuration.

Configuration LP

A possible packing of a bin can be described by an m-tuple $(t_1, \ldots, t_m)$, where t_i describes the number of pieces of size s_i. Clearly,

$$\sum_i t_i \cdot s_i \leq 1 \ .$$

We call a vector that fulfills the above constraint a configuration.

Configuration LP

Let N be the number of configurations (exponential).

Let $T_1, \dots, T_N$ be the sequence of all possible configurations (a configuration T_j has T_{ji} pieces of size s_i).

$$
\begin{array}{llrcl}
\min & & \sum_{j=1}^{N} x_j & & \\
\text{s.t.} & \forall i \in \{1 \dots m\} & \sum_{j=1}^{N} T_{ji} x_j & \geq & b_i \\
 & \forall j \in \{1, \dots, N\} & x_j & \geq & 0 \\
 & \forall j \in \{1, \dots, N\} & x_j & \text{integral} &
\end{array}
$$

Configuration LP

Let N be the number of configurations (exponential).

Let $T_1, \dots, T_N$ be the sequence of all possible configurations (a configuration T_j has T_{ji} pieces of size s_i).

$$
\begin{array}{llrcl}
\min & & \sum_{j=1}^{N} x_j & & \\
\text{s.t.} & \forall i \in \{1 \dots m\} & \sum_{j=1}^{N} T_{ji} x_j & \geq & b_i \\
& \forall j \in \{1, \dots, N\} & x_j & \geq & 0 \\
& \forall j \in \{1, \dots, N\} & x_j & \text{integral} &
\end{array}
$$

Configuration LP

Let N be the number of configurations (exponential).

Let $T_1, \ldots, T_N$ be the sequence of all possible configurations (a configuration T_j has T_{ji} pieces of size s_i).

$$\begin{array}{llrcl} \min & & \sum_{j=1}^{N} x_j & & \\ \text{s.t.} & \forall i \in \{1 \ldots m\} & \sum_{j=1}^{N} T_{ji} x_j & \geq & b_i \\ & \forall j \in \{1, \ldots, N\} & x_j & \geq & 0 \\ & \forall j \in \{1, \ldots, N\} & x_j & \text{integral} & \end{array}$$

Configuration LP

Let N be the number of configurations (exponential).

Let $T_1, \dots, T_N$ be the sequence of all possible configurations (a configuration T_j has T_{ji} pieces of size s_i).

$$
\begin{array}{lrcll}
\min & & \sum_{j=1}^{N} x_j & & \\
\text{s.t.} & \forall i \in \{1 \dots m\} & \sum_{j=1}^{N} T_{ji} x_j & \geq & b_i \\
 & \forall j \in \{1, \dots, N\} & x_j & \geq & 0 \\
 & \forall j \in \{1, \dots, N\} & x_j & \text{integral} &
\end{array}
$$

How to solve this LP?

later...

We can assume that each item has size at least $1/\mathrm{SIZE}(I)$.

Harmonic Grouping

- Sort items according to size (monotonically decreasing).
- Process items in this order; close the current group if size of items in the group is at least 2 (or larger). Then open new group.
- I.e., G_1 is the smallest cardinality set of largest items s.t. total size sums up to at least 2. Similarly, for $G_2, \dots, G_{r-1}$.
- Only the size of items in the last group G_r may sum up to less than 2.

Harmonic Grouping

- Sort items according to size (monotonically decreasing).
- Process items in this order; close the current group if size of items in the group is at least 2 (or larger). Then open new group.
- I.e., G_1 is the smallest cardinality set of largest items s.t. total size sums up to at least 2. Similarly, for $G_2, \dots, G_{r-1}$.
- Only the size of items in the last group G_r may sum up to less than 2.

Harmonic Grouping

- Sort items according to size (monotonically decreasing).
- Process items in this order; close the current group if size of items in the group is at least 2 (or larger). Then open new group.
- I.e., G_1 is the smallest cardinality set of largest items s.t. total size sums up to at least 2. Similarly, for $G_2, \ldots, G_{r-1}$.
- Only the size of items in the last group G_r may sum up to less than 2.

Harmonic Grouping

- Sort items according to size (monotonically decreasing).
- Process items in this order; close the current group if size of items in the group is at least 2 (or larger). Then open new group.
- I.e., G_1 is the smallest cardinality set of largest items s.t. total size sums up to at least 2. Similarly, for $G_2, \dots, G_{r-1}$.
- Only the size of items in the last group G_r may sum up to less than 2.

Harmonic Grouping

From the grouping we obtain instance I' as follows:

- Round all items in a group to the size of the largest group member.
- Delete all items from group G_1 and G_r.
- For groups $G_2, \ldots, G_{r-1}$ delete $n_i - n_{i-1}$ items.
- Observe that $n_i \geq n_{i-1}$.

Harmonic Grouping

From the grouping we obtain instance I' as follows:

- Round all items in a group to the size of the largest group member.
- Delete all items from group G_1 and G_r.
- For groups $G_2, \ldots, G_{r-1}$ delete $n_i - n_{i-1}$ items.
- Observe that $n_i \geq n_{i-1}$.

Harmonic Grouping

From the grouping we obtain instance I' as follows:

- Round all items in a group to the size of the largest group member.
- Delete all items from group G_1 and G_r.
- For groups $G_2, \ldots, G_{r-1}$ delete $n_i - n_{i-1}$ items.
- Observe that $n_i \geq n_{i-1}$.

Harmonic Grouping

From the grouping we obtain instance I' as follows:

- Round all items in a group to the size of the largest group member.
- Delete all items from group G_1 and G_r.
- For groups $G_2, \ldots, G_{r-1}$ delete $n_i - n_{i-1}$ items.
- Observe that $n_i \geq n_{i-1}$.

Lemma 69

The number of different sizes in I' is at most $\mathrm{SIZE}(I)/2$.

- Each group that survives (recall that G_1 and G_k are deleted) has total size at least 2.
- Hence, the number of surviving groups is at most $\mathrm{SIZE}(I)/2$.
- All items in a group have the same size in I'.

Lemma 69
The number of different sizes in I' is at most $\mathrm{SIZE}(I)/2$.

- Each group that survives (recall that G_1 and G_r are deleted) has total size at least 2.
- Hence, the number of surviving groups is at most $\mathrm{SIZE}(I)/2$.
- All items in a group have the same size in I'.

Lemma 69

The number of different sizes in I' is at most $\text{SIZE}(I)/2$.

- Each group that survives (recall that G_1 and G_r are deleted) has total size at least 2.
- Hence, the number of surviving groups is at most $\text{SIZE}(I)/2$.
- All items in a group have the same size in I'.

Lemma 69

The number of different sizes in I' is at most $\mathrm{SIZE}(I)/2$.

- Each group that survives (recall that G_1 and G_r are deleted) has total size at least 2.
- Hence, the number of surviving groups is at most $\mathrm{SIZE}(I)/2$.
- All items in a group have the same size in I'.

Lemma 70

The total size of deleted items is at most $\mathcal{O}(\log(\text{SIZE}(I)))$.

Lemma 70

The total size of deleted items is at most $\mathcal{O}(\log(\text{SIZE}(I)))$.

- The total size of items in G_1 and G_r is at most 6 as a group has total size at most 3.
- Consider a group G_i that has strictly more items than G_{i-1}.
- It discards $n_i - n_{i-1}$ pieces of total size at most

$$3\frac{n_i - n_{i-1}}{n_i} \le \sum_{j=n_{i-1}+1}^{n_i} \frac{3}{j}$$

  since the smallest piece has size at most $3/n_i$.
- Summing over all i that have $n_i > n_{i-1}$ gives a bound of at most

$$\sum_{j=1}^{n_{r-1}} \frac{3}{j} \le \mathcal{O}(\log(\text{SIZE}(I))) \ .$$

  (note that $n_r \le \text{SIZE}(I)$ since we assume that the size of each item is at least $1/\text{SIZE}(I)$).

Lemma 70
The total size of deleted items is at most $\mathcal{O}(\log(\text{SIZE}(I)))$.

- The total size of items in G_1 and G_r is at most 6 as a group has total size at most 3.
- Consider a group G_i that has strictly more items than G_{i-1}.
- It discards $n_i - n_{i-1}$ pieces of total size at most

$$3\frac{n_i - n_{i-1}}{n_i} \leq \sum_{j=n_{i-1}+1}^{n_i} \frac{3}{j}$$

  since the smallest piece has size at most $3/n_i$.
- Summing over all i that have $n_i > n_{i-1}$ gives a bound of at most

$$\sum_{j=1}^{n_{r-1}} \frac{3}{j} \leq \mathcal{O}(\log(\text{SIZE}(I))) \enspace .$$

  (note that $n_r \leq \text{SIZE}(I)$ since we assume that the size of each item is at least $1/\text{SIZE}(I)$).

Lemma 70

The total size of deleted items is at most $\mathcal{O}(\log(\mathrm{SIZE}(I)))$.

- The total size of items in G_1 and G_r is at most 6 as a group has total size at most 3.
- Consider a group G_i that has strictly more items than G_{i-1}.
- It discards $n_i - n_{i-1}$ pieces of total size at most

$$3\frac{n_i - n_{i-1}}{n_i} \le \sum_{j=n_{i-1}+1}^{n_i} \frac{3}{j}$$

  since the smallest piece has size at most $3/n_i$.
- Summing over all i that have $n_i > n_{i-1}$ gives a bound of at most

$$\sum_{j=1}^{n_{r-1}} \frac{3}{j} \le \mathcal{O}(\log(\mathrm{SIZE}(I))) \enspace .$$

  (note that $n_r \le \mathrm{SIZE}(I)$ since we assume that the size of each item is at least $1/\mathrm{SIZE}(I)$).

Lemma 70

The total size of deleted items is at most $\mathcal{O}(\log(\mathrm{SIZE}(I)))$.

- The total size of items in G_1 and G_r is at most 6 as a group has total size at most 3.
- Consider a group G_i that has strictly more items than G_{i-1}.
- It discards $n_i - n_{i-1}$ pieces of total size at most

$$3\frac{n_i - n_{i-1}}{n_i} \leq \sum_{j=n_{i-1}+1}^{n_i} \frac{3}{j}$$

  since the smallest piece has size at most $3/n_i$.
- Summing over all i that have $n_i > n_{i-1}$ gives a bound of at most

$$\sum_{j=1}^{n_{r-1}} \frac{3}{j} \leq \mathcal{O}(\log(\mathrm{SIZE}(I))) \ .$$

  (note that $n_r \leq \mathrm{SIZE}(I)$ since we assume that the size of each item is at least $1/\mathrm{SIZE}(I)$).

Algorithm 1 BinPack

if $\text{SIZE}(I) < 10$ **then**
    pack remaining items greedily
Apply harmonic grouping to create instance I'; pack discarded items in at most $\mathcal{O}(\log(\text{SIZE}(I)))$ bins.
Let x be optimal solution to configuration LP
Pack $\lfloor x_j \rfloor$ bins in configuration T_j for all j; call the packed instance I_1.
Let I_2 be remaining pieces from I'
Pack I_2 via BinPack(I_2)

Analysis

$$\mathrm{OPT}_{\mathrm{LP}}(I_1) + \mathrm{OPT}_{\mathrm{LP}}(I_2) \le \mathrm{OPT}_{\mathrm{LP}}(I') \le \mathrm{OPT}_{\mathrm{LP}}(I)$$

Proof:

- Each piece surviving in I' can be mapped to a piece in I of no lesser size. Hence, $\mathrm{OPT}_{\mathrm{LP}}(I') \le \mathrm{OPT}_{\mathrm{LP}}(I)$
- $\lfloor x_j \rfloor$ is feasible solution for I_1 (even integral).
- $x_j - \lfloor x_j \rfloor$ is feasible solution for I_2.

Analysis

$$\mathrm{OPT}_{\mathrm{LP}}(I_1) + \mathrm{OPT}_{\mathrm{LP}}(I_2) \leq \mathrm{OPT}_{\mathrm{LP}}(I') \leq \mathrm{OPT}_{\mathrm{LP}}(I)$$

Proof:

- Each piece surviving in I' can be mapped to a piece in I of no lesser size. Hence, $\mathrm{OPT}_{\mathrm{LP}}(I') \leq \mathrm{OPT}_{\mathrm{LP}}(I)$
- $\lfloor x_j \rfloor$ is feasible solution for I_1 (even integral).
- $x_j - \lfloor x_j \rfloor$ is feasible solution for I_2.

Analysis

$$\mathrm{OPT}_{\mathrm{LP}}(I_1) + \mathrm{OPT}_{\mathrm{LP}}(I_2) \le \mathrm{OPT}_{\mathrm{LP}}(I') \le \mathrm{OPT}_{\mathrm{LP}}(I)$$

Proof:

- Each piece surviving in I' can be mapped to a piece in I of no lesser size. Hence, $\mathrm{OPT}_{\mathrm{LP}}(I') \le \mathrm{OPT}_{\mathrm{LP}}(I)$
- $\lfloor x_j \rfloor$ is feasible solution for I_1 (even integral).
- $x_j - \lfloor x_j \rfloor$ is feasible solution for I_2.

Analysis

$$\mathrm{OPT}_{\mathrm{LP}}(I_1) + \mathrm{OPT}_{\mathrm{LP}}(I_2) \le \mathrm{OPT}_{\mathrm{LP}}(I') \le \mathrm{OPT}_{\mathrm{LP}}(I)$$

Proof:

- Each piece surviving in I' can be mapped to a piece in I of no lesser size. Hence, $\mathrm{OPT}_{\mathrm{LP}}(I') \le \mathrm{OPT}_{\mathrm{LP}}(I)$
- $\lfloor x_j \rfloor$ is feasible solution for I_1 (even integral).
- $x_j - \lfloor x_j \rfloor$ is feasible solution for I_2.

Analysis

Each level of the recursion partitions pieces into three types

1. Pieces discarded at this level.
2. Pieces scheduled because they are in I_1.
3. Pieces in I_2 are handed down to the next level.

Pieces of type 2 summed over all recursion levels are packed into at most OPT_{LP} many bins.

Pieces of type 1 are packed into at most

$$\mathcal{O}(\log(\text{SIZE}(I))) \cdot L$$

many bins where L is the number of recursion levels.

Analysis

Each level of the recursion partitions pieces into three types

1. Pieces discarded at this level.
2. Pieces scheduled because they are in I_1.
3. Pieces in I_2 are handed down to the next level.

Pieces of type 2 summed over all recursion levels are packed into at most $\mathrm{OPT}_{\mathrm{LP}}$ many bins.

Pieces of type 1 are packed into at most

$$\mathcal{O}(\log(\mathrm{SIZE}(I))) \cdot L$$

many bins where L is the number of recursion levels.

Analysis

Each level of the recursion partitions pieces into three types

1. Pieces discarded at this level.
2. Pieces scheduled because they are in I_1.
3. Pieces in I_2 are handed down to the next level.

Pieces of type 2 summed over all recursion levels are packed into at most $\mathrm{OPT}_{\mathrm{LP}}$ many bins.

Pieces of type 1 are packed into at most

$$\mathcal{O}(\log(\mathrm{SIZE}(I))) \cdot L$$

many bins where L is the number of recursion levels.

Analysis

Each level of the recursion partitions pieces into three types

1. Pieces discarded at this level.
2. Pieces scheduled because they are in I_1.
3. Pieces in I_2 are handed down to the next level.

Pieces of type 2 summed over all recursion levels are packed into at most $\mathrm{OPT}_{\mathrm{LP}}$ many bins.

Pieces of type 1 are packed into at most

$$\mathcal{O}(\log(\mathrm{SIZE}(I))) \cdot L$$

many bins where L is the number of recursion levels.

Analysis

Each level of the recursion partitions pieces into three types

1. Pieces discarded at this level.
2. Pieces scheduled because they are in I_1.
3. Pieces in I_2 are handed down to the next level.

Pieces of type 2 summed over all recursion levels are packed into at most $\mathrm{OPT}_{\mathrm{LP}}$ many bins.

Pieces of type 1 are packed into at most

$$\mathcal{O}(\log(\mathrm{SIZE}(I))) \cdot L$$

many bins where L is the number of recursion levels.

Analysis

We can show that $\mathrm{SIZE}(I_2) \le \mathrm{SIZE}(I)/2$. Hence, the number of recursion levels is only $\mathcal{O}(\log(\mathrm{SIZE}(I_{\mathrm{original}})))$ in total.

- The number of non-zero entries in the solution to the configuration LP for I' is at most the number of constraints, which is the number of different sizes ($\le \mathrm{SIZE}(I)/2$).
- The total size of items in I_2 can be at most $\sum_{j=1}^{N} x_j - \lfloor x_j \rfloor$ which is at most the number of non-zero entries in the solution to the configuration LP.

Analysis

We can show that $\text{SIZE}(I_2) \leq \text{SIZE}(I)/2$. Hence, the number of recursion levels is only $\mathcal{O}(\log(\text{SIZE}(I_{\text{original}})))$ in total.

- The number of non-zero entries in the solution to the configuration LP for I' is at most the number of constraints, which is the number of different sizes ($\leq \text{SIZE}(I)/2$).
- The total size of items in I_2 can be at most $\sum_{j=1}^{N} x_j - \lfloor x_j \rfloor$ which is at most the number of non-zero entries in the solution to the configuration LP.

Analysis

We can show that $\text{SIZE}(I_2) \leq \text{SIZE}(I)/2$. Hence, the number of recursion levels is only $\mathcal{O}(\log(\text{SIZE}(I_{\text{original}})))$ in total.

- The number of non-zero entries in the solution to the configuration LP for I' is at most the number of constraints, which is the number of different sizes ($\leq \text{SIZE}(I)/2$).
- The total size of items in I_2 can be at most $\sum_{j=1}^{N} x_j - \lfloor x_j \rfloor$ which is at most the number of non-zero entries in the solution to the configuration LP.

How to solve the LP?

Let $T_1, \dots, T_N$ be the sequence of all possible configurations (a configuration T_j has T_{ji} pieces of size s_i).

In total we have b_i pieces of size s_i.

Primal

$$\begin{array}{llrcl} \min & & \sum_{j=1}^{N} x_j & & \\ \text{s.t.} & \forall i \in \{1 \dots m\} & \sum_{j=1}^{N} T_{ji} x_j & \geq & b_i \\ & \forall j \in \{1, \dots, N\} & x_j & \geq & 0 \end{array}$$

Dual

$$\begin{array}{llrcl} \max & & \sum_{i=1}^{m} y_i b_i & & \\ \text{s.t.} & \forall j \in \{1, \dots, N\} & \sum_{i=1}^{m} T_{ji} y_i & \leq & 1 \\ & \forall i \in \{1, \dots, m\} & y_i & \geq & 0 \end{array}$$

How to solve the LP?

Let $T_1, \dots, T_N$ be the sequence of all possible configurations (a configuration T_j has T_{ji} pieces of size s_i).
In total we have b_i pieces of size s_i.

Primal

$$\begin{array}{llrcl} \min & & \sum_{j=1}^{N} x_j & & \\ \text{s.t.} & \forall i \in \{1 \dots m\} & \sum_{j=1}^{N} T_{ji} x_j & \geq & b_i \\ & \forall j \in \{1, \dots, N\} & x_j & \geq & 0 \end{array}$$

Dual

$$\begin{array}{llrcl} \max & & \sum_{i=1}^{m} y_i b_i & & \\ \text{s.t.} & \forall j \in \{1, \dots, N\} & \sum_{i=1}^{m} T_{ji} y_i & \leq & 1 \\ & \forall i \in \{1, \dots, m\} & y_i & \geq & 0 \end{array}$$

How to solve the LP?

Let $T_1, \dots, T_N$ be the sequence of all possible configurations (a configuration T_j has T_{ji} pieces of size s_i).
In total we have b_i pieces of size s_i.

Primal

$$\begin{array}{llrcl} \min & & \sum_{j=1}^{N} x_j & & \\ \text{s.t.} & \forall i \in \{1 \dots m\} & \sum_{j=1}^{N} T_{ji} x_j & \geq & b_i \\ & \forall j \in \{1, \dots, N\} & x_j & \geq & 0 \end{array}$$

Dual

$$\begin{array}{llrcl} \max & & \sum_{i=1}^{m} y_i b_i & & \\ \text{s.t.} & \forall j \in \{1, \dots, N\} & \sum_{i=1}^{m} T_{ji} y_i & \leq & 1 \\ & \forall i \in \{1, \dots, m\} & y_i & \geq & 0 \end{array}$$

Separation Oracle

Suppose that I am given variable assignment y for the dual.

How do I find a violated constraint?

I have to find a configuration $T_j = (T_{j1}, \dots, T_{jm})$ that

- is feasible, i.e.,

$$\sum_{i=1}^{m} T_{ji} \cdot s_i \le 1 ,$$

- and has a large profit

$$\sum_{i=1}^{m} T_{ji} y_i > 1$$

But this is the Knapsack problem.

Separation Oracle

Suppose that I am given variable assignment y for the dual.

How do I find a violated constraint?

I have to find a configuration $T_j = (T_{j1}, \dots, T_{jm})$ that

- is feasible, i.e.,

$$\sum_{i=1}^{m} T_{ji} \cdot s_i \leq 1 \ ,$$

- and has a large profit

$$\sum_{i=1}^{m} T_{ji} y_i > 1$$

But this is the Knapsack problem.

Separation Oracle

Suppose that I am given variable assignment y for the dual.

How do I find a violated constraint?

I have to find a configuration $T_j = (T_{j1}, \dots, T_{jm})$ that

- is feasible, i.e.,

$$\sum_{i=1}^{m} T_{ji} \cdot s_i \leq 1 \ ,$$

- and has a large profit

$$\sum_{i=1}^{m} T_{ji} y_i > 1$$

But this is the Knapsack problem.

Separation Oracle

Suppose that I am given variable assignment y for the dual.

How do I find a violated constraint?

I have to find a configuration $T_j = (T_{j1}, \ldots, T_{jm})$ that

- is feasible, i.e.,

$$\sum_{i=1}^{m} T_{ji} \cdot s_i \leq 1 \ ,$$

- and has a large profit

$$\sum_{i=1}^{m} T_{ji} y_i > 1$$

But this is the Knapsack problem.

Separation Oracle

We have FPTAS for Knapsack. This means if a constraint is violated with $1 + \epsilon' = 1 + \frac{\epsilon}{1-\epsilon}$ we find it, since we can obtain at least $(1 - \epsilon)$ of the optimal profit.

The solution we get is feasible for:

Dual′

$$\begin{array}{llrcl} \max & & \sum_{i=1}^{m} y_i b_i & & \\ \text{s.t.} & \forall j \in \{1,\dots,N\} & \sum_{i=1}^{m} T_{ji} y_i & \le & 1+\epsilon' \\ & \forall i \in \{1,\dots,m\} & y_i & \ge & 0 \end{array}$$

Primal′

$$\begin{array}{llrcl} \min & & (1+\epsilon') \sum_{j=1}^{N} x_j & & \\ \text{s.t.} & \forall i \in \{1 \dots m\} & \sum_{j=1}^{N} T_{ji} x_j & \ge & b_i \\ & \forall j \in \{1,\dots,N\} & x_j & \ge & 0 \end{array}$$

Separation Oracle

We have FPTAS for Knapsack. This means if a constraint is violated with $1+\epsilon' = 1+\frac{\epsilon}{1-\epsilon}$ we find it, since we can obtain at least $(1-\epsilon)$ of the optimal profit.

The solution we get is feasible for:

Dual′

$$\begin{array}{llrcl} \max & & \sum_{i=1}^{m} y_i b_i & & \\ \text{s.t.} & \forall j \in \{1,\dots,N\} & \sum_{i=1}^{m} T_{ji} y_i & \le & 1+\epsilon' \\ & \forall i \in \{1,\dots,m\} & y_i & \ge & 0 \end{array}$$

Primal′

$$\begin{array}{llrcl} \min & & (1+\epsilon') \sum_{j=1}^{N} x_j & & \\ \text{s.t.} & \forall i \in \{1\dots m\} & \sum_{j=1}^{N} T_{ji} x_j & \ge & b_i \\ & \forall j \in \{1,\dots,N\} & x_j & \ge & 0 \end{array}$$

Separation Oracle

We have FPTAS for Knapsack. This means if a constraint is violated with $1 + \epsilon' = 1 + \frac{\epsilon}{1-\epsilon}$ we find it, since we can obtain at least $(1 - \epsilon)$ of the optimal profit.

The solution we get is feasible for:

Dual′

$$\begin{array}{llrcl}
\max & & \sum_{i=1}^{m} y_i b_i & & \\
\text{s.t.} & \forall j \in \{1, \dots, N\} & \sum_{i=1}^{m} T_{ji} y_i & \leq & 1 + \epsilon' \\
 & \forall i \in \{1, \dots, m\} & y_i & \geq & 0
\end{array}$$

Primal′

$$\begin{array}{llrcl}
\min & & (1 + \epsilon') \sum_{j=1}^{N} x_j & & \\
\text{s.t.} & \forall i \in \{1 \dots m\} & \sum_{j=1}^{N} T_{ji} x_j & \geq & b_i \\
 & \forall j \in \{1, \dots, N\} & x_j & \geq & 0
\end{array}$$

Separation Oracle

We have FPTAS for Knapsack. This means if a constraint is violated with $1 + \epsilon' = 1 + \frac{\epsilon}{1-\epsilon}$ we find it, since we can obtain at least $(1 - \epsilon)$ of the optimal profit.

The solution we get is feasible for:

Dual′

$$\begin{array}{llrcl} \max & & \sum_{i=1}^{m} y_i b_i & & \\ \text{s.t.} & \forall j \in \{1,\dots,N\} & \sum_{i=1}^{m} T_{ji} y_i & \le & 1+\epsilon' \\ & \forall i \in \{1,\dots,m\} & y_i & \ge & 0 \end{array}$$

Primal′

$$\begin{array}{llrcl} \min & & (1+\epsilon')\sum_{j=1}^{N} x_j & & \\ \text{s.t.} & \forall i \in \{1\dots m\} & \sum_{j=1}^{N} T_{ji} x_j & \ge & b_i \\ & \forall j \in \{1,\dots,N\} & x_j & \ge & 0 \end{array}$$

Separation Oracle

If the value of the computed dual solution (which may be infeasible) is z then

$$\text{OPT} \le z \le (1+\epsilon')\text{OPT}$$

How do we get good primal solution (not just the value)?

- The constraints used when computing z certify that the solution is feasible for DUAL.
- Suppose that we drop all unused constraints in DUAL. We will compute the same solution feasible for DUAL''.
- Let DUAL'' be DUAL without unused constraints.
- The dual to DUAL'' is PRIMAL where we ignore variables for which the corresponding dual constraint has not been used.
- The optimum value for PRIMAL'' is at most $(1+\epsilon')\text{OPT}$.
- We can compute the corresponding solution in polytime.

Separation Oracle

If the value of the computed dual solution (which may be infeasible) is z then

$$\mathrm{OPT} \leq z \leq (1+\epsilon')\mathrm{OPT}$$

How do we get good primal solution (not just the value)?

- The constraints used when computing z certify that the solution is feasible for DUAL'.
- Suppose that we drop all unused constraints in DUAL. We will compute the same solution feasible for DUAL'.
- Let DUAL'' be DUAL without unused constraints.
- The dual to DUAL'' is PRIMAL where we ignore variables for which the corresponding dual constraint has not been used.
- The optimum value for PRIMAL'' is at most $(1+\epsilon')\mathrm{OPT}$.
- We can compute the corresponding solution in polytime.

Separation Oracle

If the value of the computed dual solution (which may be infeasible) is z then

$$\text{OPT} \leq z \leq (1 + \epsilon')\text{OPT}$$

How do we get good primal solution (not just the value)?

- The constraints used when computing z certify that the solution is feasible for DUAL'.
- Suppose that we drop all unused constraints in DUAL. We will compute the same solution feasible for DUAL'.
- Let DUAL'' be DUAL without unused constraints.
- The dual to DUAL'' is PRIMAL where we ignore variables for which the corresponding dual constraint has not been used.
- The optimum value for PRIMAL'' is at most $(1 + \epsilon')\text{OPT}$.
- We can compute the corresponding solution in polytime.

Separation Oracle

If the value of the computed dual solution (which may be infeasible) is z then

$$\mathrm{OPT} \le z \le (1 + \epsilon')\mathrm{OPT}$$

How do we get good primal solution (not just the value)?

- The constraints used when computing z certify that the solution is feasible for DUAL'.
- Suppose that we drop all unused constraints in DUAL. We will compute the same solution feasible for DUAL'.
- Let DUAL'' be DUAL without unused constraints.
- The dual to DUAL'' is PRIMAL where we ignore variables for which the corresponding dual constraint has not been used.
- The optimum value for PRIMAL'' is at most $(1 + \epsilon')\mathrm{OPT}$.
- We can compute the corresponding solution in polytime.

Separation Oracle

If the value of the computed dual solution (which may be infeasible) is z then

$$\mathrm{OPT} \leq z \leq (1+\epsilon')\mathrm{OPT}$$

How do we get good primal solution (not just the value)?

- The constraints used when computing z certify that the solution is feasible for DUAL'.
- Suppose that we drop all unused constraints in DUAL. We will compute the same solution feasible for DUAL'.
- Let DUAL'' be DUAL without unused constraints.
- The dual to DUAL'' is PRIMAL where we ignore variables for which the corresponding dual constraint has not been used.
- The optimum value for PRIMAL'' is at most $(1+\epsilon')\mathrm{OPT}$.
- We can compute the corresponding solution in polytime.

Separation Oracle

If the value of the computed dual solution (which may be infeasible) is z then

$$\mathrm{OPT} \leq z \leq (1 + \epsilon')\mathrm{OPT}$$

How do we get good primal solution (not just the value)?

- The constraints used when computing z certify that the solution is feasible for DUAL'.
- Suppose that we drop all unused constraints in DUAL. We will compute the same solution feasible for DUAL'.
- Let DUAL'' be DUAL without unused constraints.
- The dual to DUAL'' is PRIMAL where we ignore variables for which the corresponding dual constraint has not been used.
- The optimum value for PRIMAL'' is at most $(1 + \epsilon')\mathrm{OPT}$.
- We can compute the corresponding solution in polytime.

Separation Oracle

If the value of the computed dual solution (which may be infeasible) is z then

$$\mathrm{OPT} \leq z \leq (1 + \epsilon')\mathrm{OPT}$$

How do we get good primal solution (not just the value)?

- The constraints used when computing z certify that the solution is feasible for DUAL'.
- Suppose that we drop all unused constraints in DUAL. We will compute the same solution feasible for DUAL'.
- Let DUAL'' be DUAL without unused constraints.
- The dual to DUAL'' is PRIMAL where we ignore variables for which the corresponding dual constraint has not been used.
- The optimum value for PRIMAL'' is at most $(1 + \epsilon')\mathrm{OPT}$.
- We can compute the corresponding solution in polytime.

This gives that overall we need at most

$$(1+\epsilon')\mathrm{OPT}_{\mathrm{LP}}(I) + \mathcal{O}(\log^2(\mathrm{SIZE}(I)))$$

bins.

We can choose $\epsilon' = \frac{1}{\mathrm{OPT}}$ as $\mathrm{OPT} \leq$ #items and since we have a fully polynomial time approximation scheme (FPTAS) for knapsack.

This gives that overall we need at most

$$(1+\epsilon')\mathrm{OPT}_{\mathrm{LP}}(I) + \mathcal{O}(\log^2(\mathrm{SIZE}(I)))$$

bins.

We can choose $\epsilon' = \frac{1}{\mathrm{OPT}}$ as $\mathrm{OPT} \leq$ #items and since we have a fully polynomial time approximation scheme (FPTAS) for knapsack.

Lemma 71 (Chernoff Bounds)

Let $X_1, \ldots, X_n$ be n independent 0-1 random variables, not necessarily identically distributed. Then for $X = \sum_{i=1}^{n} X_i$ and $\mu = E[X]$, $L \leq \mu \leq U$, and $\delta > 0$

$$\Pr[X \geq (1+\delta)U] < \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{U} ,$$

and

$$\Pr[X \leq (1-\delta)L] < \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{L} ,$$

Lemma 72

For $0 \leq \delta \leq 1$ *we have that*

$$\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{U} \leq e^{-U\delta^2/3}$$

and

$$\left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{L} \leq e^{-L\delta^2/2}$$

Proof of Chernoff Bounds

Markovs Inequality:

Let X be random variable taking non-negative values.

Then

$$\Pr[X \geq a] \leq \mathrm{E}[X]/a$$

Trivial!

Proof of Chernoff Bounds

Markovs Inequality:
Let X be random variable taking non-negative values.
Then

$$\Pr[X \geq a] \leq \mathrm{E}[X]/a$$

Trivial!

Proof of Chernoff Bounds

Hence:

$$\Pr[X \geq (1+\delta)U] \leq \frac{\mathrm{E}[X]}{(1+\delta)U}$$

Proof of Chernoff Bounds

Hence:

$$\Pr[X \geq (1+\delta)U] \leq \frac{\mathrm{E}[X]}{(1+\delta)U} \approx \frac{1}{1+\delta}$$

Proof of Chernoff Bounds

Hence:

$$\Pr[X \geq (1+\delta)U] \leq \frac{\mathrm{E}[X]}{(1+\delta)U} \approx \frac{1}{1+\delta}$$

That's awfully weak :(

Proof of Chernoff Bounds

Set $p_i = \Pr[X_i = 1]$. Assume $p_i > 0$ for all i.

Proof of Chernoff Bounds

Set $p_i = \Pr[X_i = 1]$. Assume $p_i > 0$ for all i.

Cool Trick:

$$\Pr[X \geq (1+\delta)U] = \Pr[e^{tX} \geq e^{t(1+\delta)U}]$$

Proof of Chernoff Bounds

Set $p_i = \Pr[X_i = 1]$. Assume $p_i > 0$ for all i.

Cool Trick:

$$\Pr[X \geq (1+\delta)U] = \Pr[e^{tX} \geq e^{t(1+\delta)U}]$$

Now, we apply Markov:

$$\Pr[e^{tX} \geq e^{t(1+\delta)U}] \leq \frac{\mathrm{E}[e^{tX}]}{e^{t(1+\delta)U}} \enspace .$$

Proof of Chernoff Bounds

Set $p_i = \Pr[X_i = 1]$. Assume $p_i > 0$ for all i.

Cool Trick:

$$\Pr[X \geq (1+\delta)U] = \Pr[e^{tX} \geq e^{t(1+\delta)U}]$$

Now, we apply Markov:

$$\Pr[e^{tX} \geq e^{t(1+\delta)U}] \leq \frac{\mathrm{E}[e^{tX}]}{e^{t(1+\delta)U}} \enspace .$$

This may be a lot better (!?)

Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right]$$

Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right] = \mathrm{E}\left[e^{t\sum_i X_i}\right]$$

Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right] = \mathrm{E}\left[e^{t\sum_i X_i}\right] = \mathrm{E}\left[\prod_i e^{tX_i}\right]$$

Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right] = \mathrm{E}\left[e^{t\sum_i X_i}\right] = \mathrm{E}\left[\prod_i e^{tX_i}\right] = \prod_i \mathrm{E}\left[e^{tX_i}\right]$$

Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right] = \mathrm{E}\left[e^{t\sum_i X_i}\right] = \mathrm{E}\left[\prod_i e^{tX_i}\right] = \prod_i \mathrm{E}\left[e^{tX_i}\right]$$

$$\mathrm{E}\left[e^{tX_i}\right]$$

Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right] = \mathrm{E}\left[e^{t\sum_i X_i}\right] = \mathrm{E}\left[\prod_i e^{tX_i}\right] = \prod_i \mathrm{E}\left[e^{tX_i}\right]$$

$$\mathrm{E}\left[e^{tX_i}\right] = (1 - p_i) + p_i e^t$$

Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right] = \mathrm{E}\left[e^{t\sum_i X_i}\right] = \mathrm{E}\left[\prod_i e^{tX_i}\right] = \prod_i \mathrm{E}\left[e^{tX_i}\right]$$

$$\mathrm{E}\left[e^{tX_i}\right] = (1 - p_i) + p_i e^t = 1 + p_i(e^t - 1)$$

Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right] = \mathrm{E}\left[e^{t\sum_i X_i}\right] = \mathrm{E}\left[\prod_i e^{tX_i}\right] = \prod_i \mathrm{E}\left[e^{tX_i}\right]$$

$$\mathrm{E}\left[e^{tX_i}\right] = (1-p_i) + p_i e^t = 1 + p_i(e^t - 1) \leq e^{p_i(e^t-1)}$$

Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right] = \mathrm{E}\left[e^{t\sum_i X_i}\right] = \mathrm{E}\left[\prod_i e^{tX_i}\right] = \prod_i \mathrm{E}\left[e^{tX_i}\right]$$

$$\mathrm{E}\left[e^{tX_i}\right] = (1-p_i) + p_i e^t = 1 + p_i(e^t - 1) \leq e^{p_i(e^t-1)}$$

$$\prod_i \mathrm{E}\left[e^{tX_i}\right]$$

Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right] = \mathrm{E}\left[e^{t\sum_i X_i}\right] = \mathrm{E}\left[\prod_i e^{tX_i}\right] = \prod_i \mathrm{E}\left[e^{tX_i}\right]$$

$$\mathrm{E}\left[e^{tX_i}\right] = (1-p_i) + p_i e^t = 1 + p_i(e^t - 1) \le e^{p_i(e^t-1)}$$

$$\prod_i \mathrm{E}\left[e^{tX_i}\right] \le \prod_i e^{p_i(e^t-1)}$$

Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right] = \mathrm{E}\left[e^{t\sum_i X_i}\right] = \mathrm{E}\left[\prod_i e^{tX_i}\right] = \prod_i \mathrm{E}\left[e^{tX_i}\right]$$

$$\mathrm{E}\left[e^{tX_i}\right] = (1-p_i) + p_i e^t = 1 + p_i(e^t - 1) \le e^{p_i(e^t-1)}$$

$$\prod_i \mathrm{E}\left[e^{tX_i}\right] \le \prod_i e^{p_i(e^t-1)} = e^{\sum p_i(e^t-1)}$$

Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right] = \mathrm{E}\left[e^{t\sum_i X_i}\right] = \mathrm{E}\left[\prod_i e^{tX_i}\right] = \prod_i \mathrm{E}\left[e^{tX_i}\right]$$

$$\mathrm{E}\left[e^{tX_i}\right] = (1 - p_i) + p_i e^t = 1 + p_i(e^t - 1) \le e^{p_i(e^t - 1)}$$

$$\prod_i \mathrm{E}\left[e^{tX_i}\right] \le \prod_i e^{p_i(e^t-1)} = e^{\sum p_i(e^t-1)} = e^{(e^t-1)U}$$

Now, we apply Markov:

$$\Pr[X \geq (1+\delta)U] = \Pr[e^{tX} \geq e^{t(1+\delta)U}]$$
$$\leq \frac{\mathrm{E}[e^{tX}]}{e^{t(1+\delta)U}}$$

Now, we apply Markov:

$$\Pr[X \geq (1+\delta)U] = \Pr[e^{tX} \geq e^{t(1+\delta)U}]$$
$$\leq \frac{\mathrm{E}[e^{tX}]}{e^{t(1+\delta)U}} \leq \frac{e^{(e^t-1)U}}{e^{t(1+\delta)U}}$$

Now, we apply Markov:

$$\begin{aligned}\Pr[X \geq (1+\delta)U] &= \Pr[e^{tX} \geq e^{t(1+\delta)U}] \\ &\leq \frac{\mathrm{E}[e^{tX}]}{e^{t(1+\delta)U}} \leq \frac{e^{(e^t-1)U}}{e^{t(1+\delta)U}}\end{aligned}$$

We choose $t = \ln(1+\delta)$.

Now, we apply Markov:

$$\Pr[X \geq (1+\delta)U] = \Pr[e^{tX} \geq e^{t(1+\delta)U}]$$
$$\leq \frac{\mathrm{E}[e^{tX}]}{e^{t(1+\delta)U}} \leq \frac{e^{(e^t-1)U}}{e^{t(1+\delta)U}} \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^U$$

We choose $t = \ln(1+\delta)$.

Lemma 73

For $0 \leq \delta \leq 1$ *we have that*

$$\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{U} \leq e^{-U\delta^2/3}$$

and

$$\left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{L} \leq e^{-L\delta^2/2}$$

Show:

$$\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{U} \le e^{-U\delta^2/3}$$

Show:

$$\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{U} \le e^{-U\delta^2/3}$$

Take logarithms:

$$U(\delta - (1+\delta)\ln(1+\delta)) \le -U\delta^2/3$$

Show:

$$\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{U} \le e^{-U\delta^2/3}$$

Take logarithms:

$$U(\delta - (1+\delta)\ln(1+\delta)) \le -U\delta^2/3$$

True for $\delta = 0$.

Show:

$$\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{U} \leq e^{-U\delta^2/3}$$

Take logarithms:

$$U(\delta - (1+\delta)\ln(1+\delta)) \leq -U\delta^2/3$$

True for $\delta = 0$. Divide by U and take derivatives:

$$-\ln(1+\delta) \leq -2\delta/3$$

Reason:
As long as derivative of left side is smaller than derivative of right side the inequality holds.

$$f(\delta) := -\ln(1+\delta) + 2\delta/3 \le 0$$

$$f(\delta) := -\ln(1+\delta) + 2\delta/3 \le 0$$

A convex function ($f''(\delta) \ge 0$) on an interval takes maximum at the boundaries.

$$f(\delta) := -\ln(1+\delta) + 2\delta/3 \le 0$$

A convex function ($f''(\delta) \ge 0$) on an interval takes maximum at the boundaries.

$$f'(\delta) = -\frac{1}{1+\delta} + 2/3$$

$$f(\delta) := -\ln(1+\delta) + 2\delta/3 \le 0$$

A convex function ($f''(\delta) \ge 0$) on an interval takes maximum at the boundaries.

$$f'(\delta) = -\frac{1}{1+\delta} + 2/3 \qquad f''(\delta) = \frac{1}{(1+\delta)^2}$$

$$f(\delta) := -\ln(1+\delta) + 2\delta/3 \le 0$$

A convex function ($f''(\delta) \ge 0$) on an interval takes maximum at the boundaries.

$$f'(\delta) = -\frac{1}{1+\delta} + 2/3 \qquad f''(\delta) = \frac{1}{(1+\delta)^2}$$

$f(0) = 0$ and $f(1) = -\ln(2) + 2/3 < 0$

For $\delta \geq 1$ we show

$$\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{U} \leq e^{-U\delta/3}$$

For $\delta \geq 1$ we show

$$\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{U} \leq e^{-U\delta/3}$$

Take logarithms:

$$U(\delta - (1+\delta)\ln(1+\delta)) \leq -U\delta/3$$

For $\delta \geq 1$ we show

$$\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{U} \leq e^{-U\delta/3}$$

Take logarithms:

$$U(\delta - (1+\delta)\ln(1+\delta)) \leq -U\delta/3$$

True for $\delta = 0$.

For $\delta \geq 1$ we show

$$\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{U} \leq e^{-U\delta/3}$$

Take logarithms:

$$U(\delta - (1+\delta)\ln(1+\delta)) \leq -U\delta/3$$

True for $\delta = 0$. Divide by U and take derivatives:

$$-\ln(1+\delta) \leq -1/3 \iff \ln(1+\delta) \geq 1/3 \quad \text{(true)}$$

Reason:
As long as derivative of left side is smaller than derivative of right side the inequality holds.

Show:

$$\left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^L \leq e^{-L\delta^2/2}$$

Show:

$$\left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{L} \le e^{-L\delta^2/2}$$

Take logarithms:

$$L(-\delta - (1-\delta)\ln(1-\delta)) \le -L\delta^2/2$$

Show:

$$\left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^L \le e^{-L\delta^2/2}$$

Take logarithms:

$$L(-\delta-(1-\delta)\ln(1-\delta)) \le -L\delta^2/2$$

True for $\delta = 0$.

Show:

$$\left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^L \leq e^{-L\delta^2/2}$$

Take logarithms:

$$L(-\delta - (1-\delta)\ln(1-\delta)) \leq -L\delta^2/2$$

True for $\delta = 0$. Divide by L and take derivatives:

$$\ln(1-\delta) \leq -\delta$$

Reason:
As long as derivative of left side is smaller than derivative of right side the inequality holds.

$$\ln(1-\delta) \le -\delta$$

$$\ln(1-\delta) \leq -\delta$$

True for $\delta = 0$.

$$\ln(1 - \delta) \leq -\delta$$

True for $\delta = 0$. Take derivatives:

$$-\frac{1}{1 - \delta} \leq -1$$

$$\ln(1-\delta) \le -\delta$$

True for $\delta = 0$. Take derivatives:

$$-\frac{1}{1-\delta} \le -1$$

This holds for $0 \le \delta < 1$.

Integer Multicommodity Flows

- Given s_i-t_i pairs in a graph.
- Connect each pair by a path such that not too many path use any given edge.

$$\begin{array}{llrcl} \min & & & & W \\ \text{s.t.} & \forall i & \sum_{p \in \mathcal{P}_i} x_p & = & 1 \\ & & \sum_{p : e \in p} x_p & \le & W \\ & & x_p & \in & \{0,1\} \end{array}$$

Integer Multicommodity Flows

Randomized Rounding:
For each i choose one path from the set $\mathcal{P}_i$ at random according to the probability distribution given by the Linear Programming solution.

Theorem 74

If $W^* \geq c \ln n$ *for some constant* c, *then with probability at least* $n^{-c/3}$ *the total number of paths using any edge is at most* $W^* + \sqrt{cW^* \ln n}$.

Theorem 75

With probability at least $n^{-c/3}$ *the total number of paths using any edge is at most* $W^* + c \ln n$.

Integer Multicommodity Flows

Let X_e^i be a random variable that indicates whether the path for s_i-t_i uses edge e.

Then the number of paths using edge e is $Y_e = \sum_i X_e^i$.

$$E[Y_e] = \sum_i \sum_{p \in \mathcal{P}_i : e \in p} x_p^* = \sum_{p : e \in P} x_p^* \leq W^*$$

Integer Multicommodity Flows

Let X_e^i be a random variable that indicates whether the path for s_i-t_i uses edge e.

Then the number of paths using edge e is $Y_e = \sum_i X_e^i$.

$$E[Y_e] = \sum_i \sum_{p \in \mathcal{P}_i : e \in p} x_p^* = \sum_{p : e \in P} x_p^* \le W^*$$

Integer Multicommodity Flows

Let X_e^i be a random variable that indicates whether the path for s_i-t_i uses edge e.

Then the number of paths using edge e is $Y_e = \sum_i X_e^i$.

Integer Multicommodity Flows

Let X_e^i be a random variable that indicates whether the path for s_i-t_i uses edge e.

Then the number of paths using edge e is $Y_e = \sum_i X_e^i$.

$$E[Y_e] = \sum_i \sum_{p \in \mathcal{P}_i : e \in p} x_p^* = \sum_{p : e \in P} x_p^* \leq W^*$$

Integer Multicommodity Flows

Let X_e^i be a random variable that indicates whether the path for s_i-t_i uses edge e.

Then the number of paths using edge e is $Y_e = \sum_i X_e^i$.

$$E[Y_e] = \sum_i \sum_{p \in \mathcal{P}_i : e \in p} x_p^* = \sum_{p:e\in P} x_p^* \le W^*$$

Integer Multicommodity Flows

Let X_e^i be a random variable that indicates whether the path for s_i-t_i uses edge e.

Then the number of paths using edge e is $Y_e = \sum_i X_e^i$.

$$E[Y_e] = \sum_i \sum_{p \in \mathcal{P}_i : e \in p} x_p^* = \sum_{p : e \in P} x_p^* \le W^*$$

Integer Multicommodity Flows

Choose $\delta = \sqrt{(c \ln n)/W^*}$.

Then

$$\Pr[Y_e \geq (1+\delta)W^*] < e^{-W^*\delta^2/3} = \frac{1}{n^{c/3}}$$

Integer Multicommodity Flows

Choose $\delta = \sqrt{(c \ln n)/W^*}$.

Then

$$\Pr[Y_e \geq (1+\delta)W^*] < e^{-W^*\delta^2/3} = \frac{1}{n^{c/3}}$$

18 MAXSAT

Problem definition:

- n Boolean variables
- m clauses $C_1, \dots, C_m$. For example

$$C_7 = x_3 \vee \bar{x}_5 \vee \bar{x}_9$$

- Non-negative weight w_j for each clause C_j.
- Find an assignment of true/false to the variables sucht that the total weight of clauses that are satisfied is maximum.

18 MAXSAT

Problem definition:

- n Boolean variables
- m clauses $C_1, \dots, C_m$. For example

$$C_7 = x_3 \vee \bar{x}_5 \vee \bar{x}_9$$

- Non-negative weight w_j for each clause C_j.
- Find an assignment of true/false to the variables sucht that the total weight of clauses that are satisfied is maximum.

18 MAXSAT

Problem definition:

- n Boolean variables
- m clauses $C_1, \ldots, C_m$. For example

$$C_7 = x_3 \vee \bar{x}_5 \vee \bar{x}_9$$

- Non-negative weight w_j for each clause C_j.
- Find an assignment of true/false to the variables sucht that the total weight of clauses that are satisfied is maximum.

18 MAXSAT

Problem definition:

- n Boolean variables
- m clauses $C_1, \ldots, C_m$. For example

$$C_7 = x_3 \vee \bar{x}_5 \vee \bar{x}_9$$

- Non-negative weight w_j for each clause C_j.
- Find an assignment of true/false to the variables sucht that the total weight of clauses that are satisfied is maximum.

18 MAXSAT

Terminology:

- A variable x_i and its negation $\bar{x}_i$ are called literals.
- Hence, each clause consists of a set of literals (i.e., no duplications: $x_i \vee x_i \vee \bar{x}_j$ is **not** a clause).
- We assume a clause does not contain x_i and $\bar{x}_i$ for any i.
- x_i is called a positive literal while the negation $\bar{x}_i$ is called a negative literal.
- For a given clause C_j the number of its literals is called its length or size and denoted with ℓ_j.
- Clauses of length one are called unit clauses.

18 MAXSAT

Terminology:

- A variable x_i and its negation $\bar{x}_i$ are called literals.
- Hence, each clause consists of a set of literals (i.e., no duplications: $x_i \vee x_i \vee \bar{x}_j$ is **not** a clause).
- We assume a clause does not contain x_i and $\bar{x}_i$ for any i.
- x_i is called a positive literal while the negation $\bar{x}_i$ is called a negative literal.
- For a given clause C_j the number of its literals is called its length or size and denoted with ℓ_j.
- Clauses of length one are called unit clauses.

18 MAXSAT

Terminology:

- A variable x_i and its negation $\bar{x}_i$ are called literals.
- Hence, each clause consists of a set of literals (i.e., no duplications: $x_i \vee x_i \vee \bar{x}_j$ is **not** a clause).
- We assume a clause does not contain x_i and $\bar{x}_i$ for any i.
- x_i is called a positive literal while the negation $\bar{x}_i$ is called a negative literal.
- For a given clause C_j the number of its literals is called its length or size and denoted with ℓ_j.
- Clauses of length one are called unit clauses.

18 MAXSAT

Terminology:

- A variable x_i and its negation $\bar{x}_i$ are called literals.
- Hence, each clause consists of a set of literals (i.e., no duplications: $x_i \vee x_i \vee \bar{x}_j$ is **not** a clause).
- We assume a clause does not contain x_i and $\bar{x}_i$ for any i.
- x_i is called a positive literal while the negation $\bar{x}_i$ is called a negative literal.
- For a given clause C_j the number of its literals is called its length or size and denoted with ℓ_j.
- Clauses of length one are called unit clauses.

18 MAXSAT

Terminology:

- A variable x_i and its negation $\bar{x}_i$ are called literals.
- Hence, each clause consists of a set of literals (i.e., no duplications: $x_i \vee x_i \vee \bar{x}_j$ is **not** a clause).
- We assume a clause does not contain x_i and $\bar{x}_i$ for any i.
- x_i is called a positive literal while the negation $\bar{x}_i$ is called a negative literal.
- For a given clause C_j the number of its literals is called its length or size and denoted with ℓ_j.
- Clauses of length one are called unit clauses.

18 MAXSAT

Terminology:

- A variable x_i and its negation $\bar{x}_i$ are called literals.
- Hence, each clause consists of a set of literals (i.e., no duplications: $x_i \vee x_i \vee \bar{x}_j$ is **not** a clause).
- We assume a clause does not contain x_i and $\bar{x}_i$ for any i.
- x_i is called a positive literal while the negation $\bar{x}_i$ is called a negative literal.
- For a given clause C_j the number of its literals is called its length or size and denoted with ℓ_j.
- Clauses of length one are called unit clauses.

MAXSAT: Flipping Coins

Set each x_i independently to true with probability $\frac{1}{2}$ (and, hence, to false with probability $\frac{1}{2}$, as well).

Define random variable X_j with

$$X_j = \begin{cases} 1 & \text{if } C_j \text{ satisfied} \\ 0 & \text{otw.} \end{cases}$$

Then the total weight W of satisfied clauses is given by

$$W = \sum_j w_j X_j$$

Define random variable X_j with

$$X_j = \begin{cases} 1 & \text{if } C_j \text{ satisfied} \\ 0 & \text{otw.} \end{cases}$$

Then the total weight W of satisfied clauses is given by

$$W = \sum_j w_j X_j$$

$$E[W]$$

$$E[W]=\sum_j w_j E[X_j]$$

$$\begin{aligned} E[W] &= \sum_j w_j E[X_j] \\ &= \sum_j w_j \Pr[C_j \text{ is satisified}] \end{aligned}$$

$$
\begin{aligned}
E[W] &= \sum_j w_j E[X_j] \\
&= \sum_j w_j \Pr[C_j \text{ is satisified}] \\
&= \sum_j w_j \left(1 - \left(\frac{1}{2}\right)^{\ell_j}\right)
\end{aligned}
$$

$$
\begin{aligned}
E[W] &= \sum_j w_j E[X_j] \\
&= \sum_j w_j \Pr[C_j \text{ is satisified}] \\
&= \sum_j w_j \left(1 - \left(\frac{1}{2}\right)^{\ell_j}\right) \\
&\geq \frac{1}{2} \sum_j w_j
\end{aligned}
$$

$$
\begin{aligned}
E[W] &= \sum_j w_j E[X_j] \\
&= \sum_j w_j \Pr[C_j \text{ is satisified}] \\
&= \sum_j w_j \left(1 - \left(\frac{1}{2}\right)^{\ell_j}\right) \\
&\geq \frac{1}{2} \sum_j w_j \\
&\geq \frac{1}{2} \mathrm{OPT}
\end{aligned}
$$

MAXSAT: LP formulation

- Let for a clause C_j, P_j be the set of positive literals and N_j the set of negative literals.

$$C_j = \bigvee_{j\in P_j} x_i \vee \bigvee_{j\in N_j} \bar{x}_i$$

$$\begin{array}{llrcl} \max & & \sum_j w_j z_j & & \\ \text{s.t.} & \forall j & \sum_{i\in P_j} y_i + \sum_{i\in N_j}(1-y_i) & \geq & z_j \\ & \forall i & y_i & \in & \{0,1\} \\ & \forall j & z_j & \leq & 1 \end{array}$$

MAXSAT: LP formulation

- Let for a clause C_j, P_j be the set of positive literals and N_j the set of negative literals.

$$C_j = \bigvee_{j \in P_j} x_i \vee \bigvee_{j \in N_j} \bar{x}_i$$

$$\begin{array}{llrcl} \max & & \sum_j w_j z_j & & \\ \text{s.t.} & \forall j & \sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) & \geq & z_j \\ & \forall i & y_i & \in & \{0,1\} \\ & \forall j & z_j & \leq & 1 \end{array}$$

MAXSAT: Randomized Rounding

Set each x_i independently to true with probability y_i (and, hence, to false with probability $(1 - y_i)$).

Lemma 76 (Geometric Mean $\leq$ Arithmetic Mean)

For any nonnegative $a_1, \dots, a_k$

$$\left(\prod_{i=1}^{k} a_i\right)^{1/k} \leq \frac{1}{k}\sum_{i=1}^{k} a_i$$

Definition 77

A function f on an interval I is concave if for any two points s and r from I and any $\lambda \in [0,1]$ we have

$$f(\lambda s + (1-\lambda)r) \geq \lambda f(s) + (1-\lambda)f(r)$$

Definition 77

A function f on an interval I is concave if for any two points s and r from I and any $\lambda \in [0,1]$ we have

$$f(\lambda s + (1-\lambda)r) \geq \lambda f(s) + (1-\lambda)f(r)$$

Lemma 78

Let f be a concave function on the interval $[0,1]$, with $f(0) = a$ and $f(1) = a + b$. Then

$$\begin{aligned} f(\lambda) &= f((1-\lambda)0 + \lambda 1) \\ &\geq (1-\lambda)f(0) + \lambda f(1) \\ &= a + \lambda b \end{aligned}$$

for $\lambda \in [0,1]$.

Definition 77

A function f on an interval I is concave if for any two points s and r from I and any $\lambda \in [0,1]$ we have

$$f(\lambda s + (1-\lambda)r) \geq \lambda f(s) + (1-\lambda)f(r)$$

Lemma 78

Let f be a concave function on the interval $[0,1]$, with $f(0) = a$ and $f(1) = a + b$. Then

$$\begin{aligned} f(\lambda) &= f((1-\lambda)0 + \lambda 1) \\ &\geq (1-\lambda)f(0) + \lambda f(1) \\ &= a + \lambda b \end{aligned}$$

for $\lambda \in [0,1]$.

Definition 77

A function f on an interval I is concave if for any two points s and r from I and any $\lambda \in [0,1]$ we have

$$f(\lambda s + (1-\lambda)r) \geq \lambda f(s) + (1-\lambda)f(r)$$

Lemma 78

Let f be a concave function on the interval $[0,1]$, with $f(0) = a$ and $f(1) = a + b$. Then

$$\begin{aligned} f(\lambda) &= f((1-\lambda)0 + \lambda 1) \\ &\geq (1-\lambda)f(0) + \lambda f(1) \\ &= a + \lambda b \end{aligned}$$

for $\lambda \in [0,1]$.

$\Pr[C_j \text{ not satisfied}]$

$$\Pr[C_j \text{ not satisfied}] = \prod_{i \in P_j} (1 - y_i) \prod_{i \in N_j} y_i$$

$$\Pr[C_j \text{ not satisfied}] = \prod_{i\in P_j}(1-y_i)\prod_{i\in N_j} y_i$$

$$\leq \left[\frac{1}{\ell_j}\left(\sum_{i\in P_j}(1-y_i) + \sum_{i\in N_j} y_i\right)\right]^{\ell_j}$$

$$
\begin{aligned}
\Pr[C_j \text{ not satisfied}] &= \prod_{i\in P_j}(1-y_i)\prod_{i\in N_j} y_i \\
&\le \left[\frac{1}{\ell_j}\left(\sum_{i\in P_j}(1-y_i)+\sum_{i\in N_j} y_i\right)\right]^{\ell_j} \\
&= \left[1-\frac{1}{\ell_j}\left(\sum_{i\in P_j} y_i+\sum_{i\in N_j}(1-y_i)\right)\right]^{\ell_j}
\end{aligned}
$$

$$
\begin{aligned}
\Pr[C_j \text{ not satisfied}] &= \prod_{i\in P_j}(1-y_i)\prod_{i\in N_j} y_i \\
&\le \left[\frac{1}{\ell_j}\left(\sum_{i\in P_j}(1-y_i)+\sum_{i\in N_j} y_i\right)\right]^{\ell_j} \\
&= \left[1-\frac{1}{\ell_j}\left(\sum_{i\in P_j} y_i+\sum_{i\in N_j}(1-y_i)\right)\right]^{\ell_j} \\
&\le \left(1-\frac{z_j}{\ell_j}\right)^{\ell_j} \quad .
\end{aligned}
$$

The function $f(z) = 1 - (1 - \frac{z}{\ell})^{\ell}$ is concave. Hence,

$$\Pr[C_j \text{ satisfied}]$$

The function $f(z) = 1 - (1 - \frac{z}{\ell})^{\ell}$ is concave. Hence,

$$\Pr[C_j \text{ satisfied}] \geq 1 - \left(1 - \frac{z_j}{\ell_j}\right)^{\ell_j}$$

The function $f(z) = 1 - (1 - \frac{z}{\ell})^{\ell}$ is concave. Hence,

$$\begin{aligned}\Pr[C_j \text{ satisfied}] &\geq 1 - \left(1 - \frac{z_j}{\ell_j}\right)^{\ell_j} \\ &\geq \left[1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right] \cdot z_j \enspace .\end{aligned}$$

The function $f(z) = 1 - (1 - \frac{z}{\ell})^{\ell}$ is concave. Hence,

$$\begin{aligned}\Pr[C_j \text{ satisfied}] &\geq 1 - \left(1 - \frac{z_j}{\ell_j}\right)^{\ell_j} \\ &\geq \left[1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right] \cdot z_j \enspace .\end{aligned}$$

$f''(z) = -\frac{\ell-1}{\ell}\left[1 - \frac{z}{\ell}\right]^{\ell-2} \leq 0$ for $z \in [0,1]$. Therefore, f is concave.

$$E[W]$$

$$E[W] = \sum_j w_j \Pr[C_j \text{ is satisfied}]$$

$$
\begin{aligned}
E[W] &= \sum_j w_j \Pr[C_j \text{ is satisfied}] \\
&\geq \sum_j w_j z_j \left[1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right]
\end{aligned}
$$

$$
\begin{aligned}
E[W] &= \sum_j w_j \Pr[C_j \text{ is satisfied}] \\
&\geq \sum_j w_j z_j \left[1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right] \\
&\geq \left(1 - \frac{1}{e}\right) \mathrm{OPT} \enspace .
\end{aligned}
$$

MAXSAT: The better of two

Theorem 79
Choosing the better of the two solutions given by randomized rounding and coin flipping yields a $\frac{3}{4}$-approximation.

Let W_1 be the value of randomized rounding and W_2 the value obtained by coin flipping.

$$E[\max\{W_1, W_2\}]$$

Let W_1 be the value of randomized rounding and W_2 the value obtained by coin flipping.

$$E[\max\{W_1, W_2\}]$$
$$\geq E[\tfrac{1}{2}W_1 + \tfrac{1}{2}W_2]$$

Let W_1 be the value of randomized rounding and W_2 the value obtained by coin flipping.

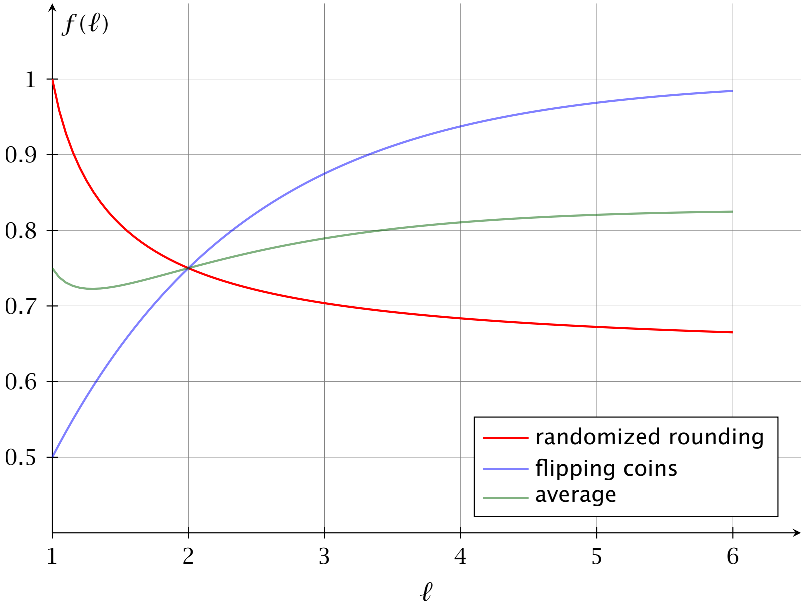
$$E[\max\{W_1, W_2\}]$$
$$\geq E[\tfrac{1}{2}W_1 + \tfrac{1}{2}W_2]$$
$$\geq \frac{1}{2}\sum_j w_j z_j \left[1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right] + \frac{1}{2}\sum_j w_j \left(1 - \left(\frac{1}{2}\right)^{\ell_j}\right)$$

Let W_1 be the value of randomized rounding and W_2 the value obtained by coin flipping.

$$
\begin{aligned}
&E[\max\{W_1, W_2\}] \\
&\qquad \geq E[\tfrac{1}{2}W_1 + \tfrac{1}{2}W_2] \\
&\qquad \geq \frac{1}{2}\sum_j w_j z_j \left[1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right] + \frac{1}{2}\sum_j w_j \left(1 - \left(\frac{1}{2}\right)^{\ell_j}\right) \\
&\qquad \geq \sum_j w_j z_j \Bigg[\underbrace{\frac{1}{2}\left(1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right) + \frac{1}{2}\left(1 - \left(\frac{1}{2}\right)^{\ell_j}\right)}_{\geq \frac{3}{4} \text{ for all integers}}\Bigg]
\end{aligned}
$$

Let W_1 be the value of randomized rounding and W_2 the value obtained by coin flipping.

$$
\begin{aligned}
E[\max\{W_1, W_2\}]
&\geq E[\tfrac{1}{2}W_1 + \tfrac{1}{2}W_2] \\
&\geq \frac{1}{2}\sum_j w_j z_j \left[1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right] + \frac{1}{2}\sum_j w_j \left(1 - \left(\frac{1}{2}\right)^{\ell_j}\right) \\
&\geq \sum_j w_j z_j \Bigg[\underbrace{\frac{1}{2}\left(1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right) + \frac{1}{2}\left(1 - \left(\frac{1}{2}\right)^{\ell_j}\right)}_{\geq \frac{3}{4} \text{ for all integers}}\Bigg] \\
&\geq \frac{3}{4}\text{OPT}
\end{aligned}
$$


$f(\ell)$
1
0.9
0.8
0.7
0.6
0.5
1
2
3
4
5
6
ℓ
randomized rounding
flipping coins
average

MAXSAT: Nonlinear Randomized Rounding

So far we used linear randomized rounding, i.e., the probability that a variable is set to 1/true was exactly the value of the corresponding variable in the linear program.

We could define a function $f : [0,1] \rightarrow [0,1]$ and set x_i to true with probability $f(y_i)$.

MAXSAT: Nonlinear Randomized Rounding

So far we used linear randomized rounding, i.e., the probability that a variable is set to 1/true was exactly the value of the corresponding variable in the linear program.

We could define a function $f : [0,1] \rightarrow [0,1]$ and set x_i to true with probability $f(y_i)$.

MAXSAT: Nonlinear Randomized Rounding

Let $f : [0,1] \to [0,1]$ be a function with

$$1 - 4^{-x} \le f(x) \le 4^{x-1}$$

Theorem 80

Rounding the LP-solution with a function f of the above form gives a $\frac{3}{4}$-approximation.

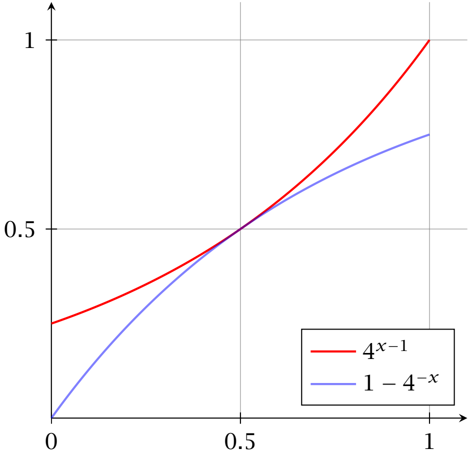
MAXSAT: Nonlinear Randomized Rounding

Let $f : [0,1] \rightarrow [0,1]$ be a function with

$$1 - 4^{-x} \leq f(x) \leq 4^{x-1}$$

Theorem 80

Rounding the LP-solution with a function f of the above form gives a $\frac{3}{4}$-approximation.


1
0.5
0
0.5
1
4^{x-1}
$1-4^{-x}$

$\Pr[C_j \text{ not satisfied}]$

$$\Pr[C_j \text{ not satisfied}] = \prod_{i \in P_j} (1 - f(y_i)) \prod_{i \in N_j} f(y_i)$$

$$\begin{aligned}\Pr[C_j \text{ not satisfied}] &= \prod_{i\in P_j}(1-f(y_i))\prod_{i\in N_j} f(y_i)\\ &\le \prod_{i\in P_j} 4^{-y_i}\prod_{i\in N_j} 4^{y_i-1}\end{aligned}$$

$$
\begin{aligned}
\Pr[C_j \text{ not satisfied}] &= \prod_{i\in P_j}(1-f(y_i))\prod_{i\in N_j} f(y_i) \\
&\le \prod_{i\in P_j} 4^{-y_i}\prod_{i\in N_j} 4^{y_i-1} \\
&= 4^{-(\sum_{i\in P_j} y_i + \sum_{i\in N_j}(1-y_i))}
\end{aligned}
$$

$$\begin{aligned}
\Pr[C_j \text{ not satisfied}] &= \prod_{i\in P_j}(1-f(y_i))\prod_{i\in N_j} f(y_i) \\
&\le \prod_{i\in P_j} 4^{-y_i}\prod_{i\in N_j} 4^{y_i-1} \\
&= 4^{-(\sum_{i\in P_j} y_i+\sum_{i\in N_j}(1-y_i))} \\
&\le 4^{-z_j}
\end{aligned}$$

The function $g(z) = 1 - 4^{-z}$ is concave on $[0, 1]$. Hence,

The function $g(z) = 1 - 4^{-z}$ is concave on $[0, 1]$. Hence,

$$\Pr[C_j \text{ satisfied}]$$

The function $g(z) = 1 - 4^{-z}$ is concave on $[0, 1]$. Hence,

$$\Pr[C_j \text{ satisfied}] \geq 1 - 4^{-z_j}$$

The function $g(z) = 1 - 4^{-z}$ is concave on $[0,1]$. Hence,

$$\Pr[C_j \text{ satisfied}] \geq 1 - 4^{-z_j} \geq \frac{3}{4} z_j \enspace .$$

The function $g(z) = 1 - 4^{-z}$ is concave on $[0, 1]$. Hence,

$$\Pr[C_j \text{ satisfied}] \geq 1 - 4^{-z_j} \geq \frac{3}{4} z_j \ .$$

The function $g(z) = 1 - 4^{-z}$ is concave on $[0, 1]$. Hence,

$$\Pr[C_j \text{ satisfied}] \geq 1 - 4^{-z_j} \geq \frac{3}{4} z_j \ .$$

Therefore,

$$E[W]$$

The function $g(z) = 1 - 4^{-z}$ is concave on $[0,1]$. Hence,

$$\Pr[C_j \text{ satisfied}] \geq 1 - 4^{-z_j} \geq \frac{3}{4} z_j \enspace .$$

Therefore,

$$E[W] = \sum_j w_j \Pr[C_j \text{ satisfied}]$$

The function $g(z) = 1 - 4^{-z}$ is concave on $[0,1]$. Hence,

$$\Pr[C_j \text{ satisfied}] \geq 1 - 4^{-z_j} \geq \frac{3}{4} z_j \enspace .$$

Therefore,

$$E[W] = \sum_j w_j \Pr[C_j \text{ satisfied}] \geq \frac{3}{4} \sum_j w_j z_j$$

The function $g(z) = 1 - 4^{-z}$ is concave on $[0, 1]$. Hence,

$$\Pr[C_j \text{ satisfied}] \geq 1 - 4^{-z_j} \geq \frac{3}{4} z_j \ .$$

Therefore,

$$E[W] = \sum_j w_j \Pr[C_j \text{ satisfied}] \geq \frac{3}{4} \sum_j w_j z_j \geq \frac{3}{4} \mathrm{OPT}$$

Can we do better?

Not if we compare ourselves to the value of an optimum LP-solution.

Definition 81 (Integrality Gap)

The integrality gap for an ILP is the worst-case ratio over all instances of the problem of the value of an optimal IP-solution to the value of an optimal solution to its linear programming relaxation.

Note that the integrality is less than one for maximization problems and larger than one for minimization problems (of course, equality is possible).

Note that an integrality gap only holds for one specific ILP formulation.

Can we do better?

Not if we compare ourselves to the value of an optimum LP-solution.

Definition 81 (Integrality Gap)

The integrality gap for an ILP is the worst-case ratio over all instances of the problem of the value of an optimal IP-solution to the value of an optimal solution to its linear programming relaxation.

Note that the integrality is less than one for maximization problems and larger than one for minimization problems (of course, equality is possible).

Note that an integrality gap only holds for one specific ILP formulation.

Can we do better?

Not if we compare ourselves to the value of an optimum LP-solution.

Definition 81 (Integrality Gap)

The integrality gap for an ILP is the worst-case ratio over all instances of the problem of the value of an optimal IP-solution to the value of an optimal solution to its linear programming relaxation.

Note that the integrality is less than one for maximization problems and larger than one for minimization problems (of course, equality is possible).

Note that an integrality gap only holds for one specific ILP formulation.

Can we do better?

Not if we compare ourselves to the value of an optimum LP-solution.

Definition 81 (Integrality Gap)

The integrality gap for an ILP is the worst-case ratio over all instances of the problem of the value of an optimal IP-solution to the value of an optimal solution to its linear programming relaxation.

Note that the integrality is less than one for maximization problems and larger than one for minimization problems (of course, equality is possible).

Note that an integrality gap only holds for one specific ILP formulation.

Can we do better?

Not if we compare ourselves to the value of an optimum LP-solution.

Definition 81 (Integrality Gap)
The integrality gap for an ILP is the worst-case ratio over all instances of the problem of the value of an optimal IP-solution to the value of an optimal solution to its linear programming relaxation.

Note that the integrality is less than one for maximization problems and larger than one for minimization problems (of course, equality is possible).

Note that an integrality gap only holds for one specific ILP formulation.

Lemma 82

Our ILP-formulation for the MAXSAT problem has integrality gap at most $\frac{3}{4}$.

$$
\begin{array}{llrcl}
\max & & \sum_j w_j z_j & & \\
\text{s.t.} & \forall j & \sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) & \geq & z_j \\
 & \forall i & y_i & \in & \{0, 1\} \\
 & \forall j & z_j & \leq & 1
\end{array}
$$

Consider: $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_2)$

- any solution can satisfy at most 3 clauses
- we can set $y_1 = y_2 = 1/2$ in the LP; this allows to set $z_1 = z_2 = z_3 = z_4 = 1$
- hence, the LP has value 4.

Lemma 82

Our ILP-formulation for the MAXSAT problem has integrality gap at most $\frac{3}{4}$.

$$
\begin{array}{llrcl}
\max & & \sum_j w_j z_j & & \\
\text{s.t.} & \forall j & \sum_{i\in P_j} y_i + \sum_{i\in N_j}(1-y_i) & \geq & z_j \\
& \forall i & y_i & \in & \{0,1\} \\
& \forall j & z_j & \leq & 1
\end{array}
$$

Consider: $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_2)$

- any solution can satisfy at most 3 clauses
- we can set $y_1 = y_2 = 1/2$ in the LP; this allows to set $z_1 = z_2 = z_3 = z_4 = 1$
- hence, the LP has value 4.

Repetition: Primal Dual for Set Cover

Primal Relaxation:

$$\begin{array}{llrcl} \min & & \sum_{i=1}^{k} w_i x_i & & \\ \text{s.t.} & \forall u \in U & \sum_{i:u\in S_i} x_i & \geq & 1 \\ & \forall i \in \{1,\dots,k\} & x_i & \geq & 0 \end{array}$$

Dual Formulation:

$$\begin{array}{llrcl} \max & & \sum_{u\in U} y_u & & \\ \text{s.t.} & \forall i \in \{1,\dots,k\} & \sum_{u:u\in S_i} y_u & \leq & w_i \\ & & y_u & \geq & 0 \end{array}$$

Repetition: Primal Dual for Set Cover

Primal Relaxation:

$$\begin{array}{llrcl} \min & & \sum_{i=1}^{k} w_i x_i & & \\ \text{s.t.} & \forall u \in U & \sum_{i:u\in S_i} x_i & \geq & 1 \\ & \forall i \in \{1,\dots,k\} & x_i & \geq & 0 \end{array}$$

Dual Formulation:

$$\begin{array}{llrcl} \max & & \sum_{u\in U} y_u & & \\ \text{s.t.} & \forall i \in \{1,\dots,k\} & \sum_{u:u\in S_i} y_u & \leq & w_i \\ & & y_u & \geq & 0 \end{array}$$

Repetition: Primal Dual for Set Cover

Algorithm:

- Start with $y = 0$ (feasible dual solution).
  Start with $x = 0$ (integral primal solution that may be infeasible).
- While x not feasible
  - Identify an element e that is not covered in current primal integral solution.
  - Increase dual variable y_e until a dual constraint becomes tight (maybe increase by 0!).
  - If this is the constraint for set S_j set $x_j = 1$ (add this set to your solution).

Repetition: Primal Dual for Set Cover

Algorithm:

- Start with $y = 0$ (feasible dual solution).
  Start with $x = 0$ (integral primal solution that may be infeasible).
- While x not feasible
  - Identify an element e that is not covered in current primal integral solution.
  - Increase dual variable y_e until a dual constraint becomes tight (maybe increase by 0!).
  - If this is the constraint for set S_j set $x_j = 1$ (add this set to your solution).

Repetition: Primal Dual for Set Cover

Algorithm:

- Start with $y = 0$ (feasible dual solution).
  Start with $x = 0$ (integral primal solution that may be infeasible).
- While x not feasible
  - Identify an element e that is not covered in current primal integral solution.
  - Increase dual variable y_e until a dual constraint becomes tight (maybe increase by 0!).
  - If this is the constraint for set S_j set $x_j = 1$ (add this set to your solution).

Repetition: Primal Dual for Set Cover

Algorithm:

- Start with $y = 0$ (feasible dual solution).
  Start with $x = 0$ (integral primal solution that may be infeasible).
- While x not feasible
  - Identify an element e that is not covered in current primal integral solution.
  - Increase dual variable y_e until a dual constraint becomes tight (maybe increase by 0!).
  - If this is the constraint for set S_j set $x_j = 1$ (add this set to your solution).

Repetition: Primal Dual for Set Cover

Algorithm:

- Start with $y = 0$ (feasible dual solution).
  Start with $x = 0$ (integral primal solution that may be infeasible).
- While x not feasible
  - Identify an element e that is not covered in current primal integral solution.
  - Increase dual variable y_e until a dual constraint becomes tight (maybe increase by 0!).
  - If this is the constraint for set S_j set $x_j = 1$ (add this set to your solution).

Repetition: Primal Dual for Set Cover

Analysis:

Repetition: Primal Dual for Set Cover

Analysis:

- For every set S_j with $x_j = 1$ we have

$$\sum_{e \in S_j} y_e = w_j$$

Repetition: Primal Dual for Set Cover

Analysis:

- For every set S_j with $x_j = 1$ we have

$$\sum_{e \in S_j} y_e = w_j$$

- Hence our cost is

Repetition: Primal Dual for Set Cover

Analysis:

- For every set S_j with $x_j = 1$ we have
$$\sum_{e \in S_j} y_e = w_j$$
- Hence our cost is
$$\sum_j w_j$$

Repetition: Primal Dual for Set Cover

Analysis:

- For every set S_j with $x_j = 1$ we have

$$\sum_{e \in S_j} y_e = w_j$$

- Hence our cost is

$$\sum_j w_j = \sum_j \sum_{e \in S_j} y_e$$

Repetition: Primal Dual for Set Cover

Analysis:

- For every set S_j with $x_j = 1$ we have

$$\sum_{e \in S_j} y_e = w_j$$

- Hence our cost is

$$\sum_j w_j = \sum_j \sum_{e \in S_j} y_e = \sum_e |\{j : e \in S_j\}| \cdot y_e$$

Repetition: Primal Dual for Set Cover

Analysis:

- For every set S_j with $x_j = 1$ we have

$$\sum_{e \in S_j} y_e = w_j$$

- Hence our cost is

$$\sum_j w_j = \sum_j \sum_{e \in S_j} y_e = \sum_e |\{j : e \in S_j\}| \cdot y_e \le f \cdot \sum_e y_e \le f \cdot \mathrm{OPT}$$

Note that the constructed pair of primal and dual solution fulfills primal slackness conditions.

Note that the constructed pair of primal and dual solution fulfills primal slackness conditions.

This means

$$x_j > 0 \Rightarrow \sum_{e \in S_j} y_e = w_j$$

Note that the constructed pair of primal and dual solution fulfills primal slackness conditions.

This means

$$x_j > 0 \Rightarrow \sum_{e \in S_j} y_e = w_j$$

If we would also fulfill dual slackness conditions

$$y_e > 0 \Rightarrow \sum_{j : e \in S_j} x_j = 1$$

then the solution would be optimal!!!

We don’t fulfill these constraint but we fulfill an approximate version:

We don't fulfill these constraint but we fulfill an approximate version:

$$y_e > 0 \Rightarrow 1 \leq \sum_{j:e \in S_j} x_j \leq f$$

We don't fulfill these constraint but we fulfill an approximate version:

$$y_e > 0 \Rightarrow 1 \leq \sum_{j:e\in S_j} x_j \leq f$$

This is sufficient to show that the solution is an f-approximation.

Suppose we have a primal/dual pair

$$\begin{array}{llrcl} \min & & \sum_j c_j x_j & & \\ \text{s.t.} & \forall i & \sum_{j:} a_{ij} x_j & \geq & b_i \\ & \forall j & x_j & \geq & 0 \end{array}$$

$$\begin{array}{llrcl} \max & & \sum_i b_i y_i & & \\ \text{s.t.} & \forall j & \sum_i a_{ij} y_i & \leq & c_j \\ & \forall i & y_i & \geq & 0 \end{array}$$

Suppose we have a primal/dual pair

min		$\sum_j c_j x_j$		
s.t.	$\forall i$	$\sum_{j:} a_{ij} x_j$	$\geq$	b_i
	$\forall j$	x_j	$\geq$	0

max		$\sum_i b_i y_i$		
s.t.	$\forall j$	$\sum_i a_{ij} y_i$	$\leq$	c_j
	$\forall i$	y_i	$\geq$	0

and solutions that fulfill approximate slackness conditions:

$$x_j > 0 \Rightarrow \sum_i a_{ij} y_i \geq \frac{1}{\alpha} c_j$$

$$y_i > 0 \Rightarrow \sum_j a_{ij} x_j \leq \beta b_i$$

Then

$$\sum_j c_j x_j$$

Then

$$\boxed{\sum_j c_j x_j}$$

primal cost

Then

right hand side of j-th dual constraint

$$\sum_j c_j x_j$$

primal cost

Then

$$\boxed{\sum_j c_j x_j} \leq \alpha \sum_j \left(\sum_i a_{ij} y_i \right) x_j$$

primal cost

Then

$$\sum_j c_j x_j \le \alpha \sum_j \left(\sum_i a_{ij} y_i \right) x_j$$

primal cost

$$= \alpha \sum_i \left(\sum_j a_{ij} x_j \right) y_i$$

Then

$$\sum_j c_j x_j \le \alpha \sum_j \left(\sum_i a_{ij} y_i \right) x_j$$

primal cost

$$= \alpha \sum_i \left(\sum_j a_{ij} x_j \right) y_i$$

$$\le \alpha\beta \cdot \sum_i b_i y_i$$

Then

$$\sum_j c_j x_j \leq \alpha \sum_j \left(\sum_i a_{ij} y_i \right) x_j$$

primal cost

$$= \alpha \sum_i \left(\sum_j a_{ij} x_j \right) y_i$$

$$\leq \alpha\beta \cdot \sum_i b_i y_i$$

dual objective

Feedback Vertex Set for Undirected Graphs

- Given a graph $G = (V, E)$ and non-negative weights $w_v \geq 0$ for vertex $v \in V$.

Feedback Vertex Set for Undirected Graphs

- Given a graph $G = (V, E)$ and non-negative weights $w_v \geq 0$ for vertex $v \in V$.
- Choose a minimum cost subset of vertices s.t. every cycle contains at least one vertex.

We can encode this as an instance of Set Cover

- Each vertex can be viewed as a set that contains some cycles.

We can encode this as an instance of Set Cover

- Each vertex can be viewed as a set that contains some cycles.
- However, this encoding gives a Set Cover instance of non-polynomial size.

We can encode this as an instance of Set Cover

- Each vertex can be viewed as a set that contains some cycles.
- However, this encoding gives a Set Cover instance of non-polynomial size.
- The $O(\log n)$-approximation for Set Cover does not help us to get a good solution.

Let C denote the set of all cycles (where a cycle is identified by its set of vertices)

Let $\mathcal{C}$ denote the set of all cycles (where a cycle is identified by its set of vertices)

Primal Relaxation:

$$
\begin{array}{llrcl}
\min & & \sum_v w_v x_v & & \\
\text{s.t.} & \forall C \in \mathcal{C} & \sum_{v \in C} x_v & \geq & 1 \\
 & \forall v & x_v & \geq & 0
\end{array}
$$

Dual Formulation:

$$
\begin{array}{llrcl}
\max & & \sum_{C \in \mathcal{C}} y_C & & \\
\text{s.t.} & \forall v \in V & \sum_{C : v \in C} y_C & \leq & w_v \\
 & \forall C & y_C & \geq & 0
\end{array}
$$

If we perform the previous dual technique for Set Cover we get the following:

- Start with $x = 0$ and $y = 0$

If we perform the previous dual technique for Set Cover we get the following:

- Start with $x = 0$ and $y = 0$
- While there is a cycle C that is not covered (does not contain a chosen vertex).

If we perform the previous dual technique for Set Cover we get the following:

- Start with $x = 0$ and $y = 0$
- While there is a cycle C that is not covered (does not contain a chosen vertex).
  - Increase y_C until dual constraint for some vertex v becomes tight.

If we perform the previous dual technique for Set Cover we get the following:

- Start with $x = 0$ and $y = 0$
- While there is a cycle C that is not covered (does not contain a chosen vertex).
  - Increase y_C until dual constraint for some vertex v becomes tight.
  - set $x_v = 1$.

Then

$$\sum_v w_v x_v$$

Then

$$\sum_v w_v x_v = \sum_v \sum_{C:v\in C} y_C x_v$$

Then

$$
\begin{aligned}
\sum_v w_v x_v &= \sum_v \sum_{C:v\in C} y_C x_v \\
&= \sum_{v\in S} \sum_{C:v\in C} y_C
\end{aligned}
$$

where S is the set of vertices we choose.

Then

$$\begin{aligned}\sum_v w_v x_v &= \sum_v \sum_{C:v\in C} y_C x_v \\ &= \sum_{v\in S} \sum_{C:v\in C} y_C \\ &= \sum_C |S\cap C|\cdot y_C\end{aligned}$$

where S is the set of vertices we choose.

Then

$$
\begin{aligned}
\sum_v w_v x_v &= \sum_v \sum_{C:v\in C} y_C x_v \\
&= \sum_{v\in S} \sum_{C:v\in C} y_C \\
&= \sum_C |S\cap C| \cdot y_C
\end{aligned}
$$

where S is the set of vertices we choose.

If every cycle is short we get a good approximation ratio, but this is unrealistic.

Algorithm 1 FeedbackVertexSet

1: $y \leftarrow 0$
2: $x \leftarrow 0$
3: **while** exists cycle C in G **do**
4: increase y_C until there is $v \in C$ s.t. $\sum_{C:v\in C} y_C = w_v$
5: $x_v = 1$
6: remove v from G
7: repeatedly remove vertices of degree 1 from G

Idea:

Always choose a short cycle that is not covered. If we always find a cycle of length at most α we get an α-approximation.

Idea:

Always choose a short cycle that is not covered. If we always find a cycle of length at most α we get an α-approximation.

Observation:

For any path P of vertices of degree 2 in G the algorithm chooses at most one vertex from P.

Observation:

If we always choose a cycle for which the number of vertices of degree at least 3 is at most α we get a 2α-approximation.

Observation:

If we always choose a cycle for which the number of vertices of degree at least 3 is at most α we get a 2α-approximation.

Theorem 83

In any graph with no vertices of degree 1, there always exists a cycle that has at most $\mathcal{O}(\log n)$ vertices of degree 3 or more. We can find such a cycle in linear time.

This means we have

$$y_C > 0 \Rightarrow |S \cap C| \le \mathcal{O}(\log n) \ .$$

Primal Dual for Shortest Path

Given a graph $G = (V, E)$ with two nodes $s, t \in V$ and edge-weights $c : E \to \mathbb{R}^+$ find a shortest path between s and t w.r.t. edge-weights c.

$$\begin{array}{llll} \min & & \sum_e c(e)x_e & \\ \text{s.t.} & \forall S \in \mathcal{S} & \sum_{e:\delta(S)} x_e & \geq & 1 \\ & \forall e \in E & x_e & \in & \{0,1\} \end{array}$$

Here $\delta(S)$ denotes the set of edges with exactly one end-point in S, and $\mathcal{S} = \{S \subseteq V : s \in S, t \notin S\}$.

Primal Dual for Shortest Path

Given a graph $G = (V, E)$ with two nodes $s, t \in V$ and edge-weights $c : E \to \mathbb{R}^+$ find a shortest path between s and t w.r.t. edge-weights c.

$$\begin{array}{llrcl} \min & & \sum_e c(e)x_e & & \\ \text{s.t.} & \forall S \in \mathcal{S} & \sum_{e:\delta(S)} x_e & \geq & 1 \\ & \forall e \in E & x_e & \in & \{0,1\} \end{array}$$

Here $\delta(S)$ denotes the set of edges with exactly one end-point in S, and $\mathcal{S} = \{S \subseteq V : s \in S, t \notin S\}$.

Primal Dual for Shortest Path

The Dual:

$$\begin{array}{llrcl} \max & & \sum_S y_S & & \\ \text{s.t.} & \forall e \in E & \sum_{S: e \in \delta(S)} y_S & \leq & c(e) \\ & \forall S \in \mathcal{S} & y_S & \geq & 0 \end{array}$$

Here $\delta(S)$ denotes the set of edges with exactly one end-point in S, and $\mathcal{S} = \{S \subseteq V : s \in S, t \notin S\}$.

Primal Dual for Shortest Path

The Dual:

$$\begin{array}{llrcl} \max & & \sum_S y_S & & \\ \text{s.t.} & \forall e \in E & \sum_{S: e \in \delta(S)} y_S & \le & c(e) \\ & \forall S \in \mathcal{S} & y_S & \ge & 0 \end{array}$$

Here $\delta(S)$ denotes the set of edges with exactly one end-point in S, and $\mathcal{S} = \{S \subseteq V : s \in S, t \notin S\}$.

Primal Dual for Shortest Path

We can interpret the value y_S as the width of a moat surounding the set S.

Each set can have its own moat but all moats must be disjoint.

An edge cannot be shorter than all the moats that it has to cross.

Primal Dual for Shortest Path

We can interpret the value y_S as the width of a moat surounding the set S.

Each set can have its own moat but all moats must be disjoint.

An edge cannot be shorter than all the moats that it has to cross.

Primal Dual for Shortest Path

We can interpret the value y_S as the width of a moat surounding the set S.

Each set can have its own moat but all moats must be disjoint.

An edge cannot be shorter than all the moats that it has to cross.

Primal Dual for Shortest Path

We can interpret the value y_S as the width of a moat surounding the set S.

Each set can have its own moat but all moats must be disjoint.

An edge cannot be shorter than all the moats that it has to cross.

Algorithm 1 PrimalDualShortestPath

1: $y \leftarrow 0$
2: $F \leftarrow \emptyset$
3: **while** there is no s-t path in (V, F) **do**
4: Let C be the connected component of (V, F) containing s
5: Increase y_C until there is an edge $e' \in \delta(C)$ such that $\sum_{S: e' \in \delta(S)} y_S = c(e')$.
6: $F \leftarrow F \cup \{e'\}$
7: Let P be an s-t path in (V, F)
8: **return** P

Lemma 84

At each point in time the set F forms a tree.

Proof:

- In each iteration we take the current connected component from (V, F) that contains s (call this component C) and add some edge from $\delta(C)$ to F.
- Since, at most one end-point of the new edge is in C the edge cannot close a cycle.

Lemma 84

At each point in time the set F forms a tree.

Proof:

- In each iteration we take the current connected component from (V,F) that contains s (call this component C) and add some edge from $\delta(C)$ to F.
- Since, at most one end-point of the new edge is in C the edge cannot close a cycle.

Lemma 84

At each point in time the set F forms a tree.

Proof:

- In each iteration we take the current connected component from (V, F) that contains s (call this component C) and add some edge from $\delta(C)$ to F.
- Since, at most one end-point of the new edge is in C the edge cannot close a cycle.

$$\sum_{e \in P} c(e)$$

$$\sum_{e\in P} c(e) = \sum_{e\in P} \sum_{S:e\in\delta(S)} y_S$$

$$\sum_{e\in P} c(e) = \sum_{e\in P} \sum_{S:e\in\delta(S)} y_S$$

$$= \sum_{S:s\in S,t\notin S} |P \cap \delta(S)| \cdot y_S \ .$$

$$\sum_{e\in P} c(e) = \sum_{e\in P} \sum_{S:e\in\delta(S)} y_S$$
$$= \sum_{S:s\in S,t\notin S} |P\cap\delta(S)|\cdot y_S \enspace .$$

If we can show that $y_S > 0$ implies $|P\cap\delta(S)| = 1$ gives

$$\sum_{e\in P} c(e) = \sum_S y_S \le \mathrm{OPT}$$

by weak duality.

$$\sum_{e\in P} c(e) = \sum_{e\in P} \sum_{S:e\in\delta(S)} y_S$$
$$= \sum_{S:s\in S,t\notin S} |P\cap\delta(S)|\cdot y_S \ .$$

If we can show that $y_S > 0$ implies $|P\cap\delta(S)| = 1$ gives

$$\sum_{e\in P} c(e) = \sum_S y_S \le \mathrm{OPT}$$

by weak duality.

Hence, we find a shortest path.

If S contains two edges from P then there must exist a subpath P' of P that starts and ends with a vertex from S (and all interior vertices are not in S).

When we increased y_S, S was a connected component of the set of edges F' that we had chosen till this point.

$F' \cup P'$ contains a cycle. Hence, also the final set of edges contains a cycle.

This is a contradiction.

If S contains two edges from P then there must exist a subpath P' of P that starts and ends with a vertex from S (and all interior vertices are not in S).

When we increased y_S, S was a connected component of the set of edges F' that we had chosen till this point.

$F' \cup P'$ contains a cycle. Hence, also the final set of edges contains a cycle.

This is a contradiction.

If S contains two edges from P then there must exist a subpath P' of P that starts and ends with a vertex from S (and all interior vertices are not in S).

When we increased y_S, S was a connected component of the set of edges F' that we had chosen till this point.

$F' \cup P'$ contains a cycle. Hence, also the final set of edges contains a cycle.

This is a contradiction.

If S contains two edges from P then there must exist a subpath P' of P that starts and ends with a vertex from S (and all interior vertices are not in S).

When we increased y_S, S was a connected component of the set of edges F' that we had chosen till this point.

$F' \cup P'$ contains a cycle. Hence, also the final set of edges contains a cycle.

This is a contradiction.

If S contains two edges from P then there must exist a subpath P' of P that starts and ends with a vertex from S (and all interior vertices are not in S).

When we increased y_S, S was a connected component of the set of edges F' that we had chosen till this point.

$F' \cup P'$ contains a cycle. Hence, also the final set of edges contains a cycle.

This is a contradiction.

Steiner Forest Problem:
Given a graph $G = (V, E)$, together with source-target pairs $s_i, t_i, i = 1, \dots, k$, and a cost function $c : E \to \mathbb{R}^+$ on the edges. Find a subset $F \subseteq E$ of the edges such that for every $i \in \{1, \dots, k\}$ there is a path between s_i and t_i only using edges in F.

$$
\begin{array}{llrcl}
\min & & \sum_e c(e) x_e & & \\
\text{s.t.} & \forall S \subseteq V : S \in S_i \text{ for some } i & \sum_{e \in \delta(S)} x_e & \geq & 1 \\
 & \forall e \in E & x_e & \in & \{0, 1\}
\end{array}
$$

Here S_i contains all sets S such that $s_i \in S$ and $t_i \notin S$.

Steiner Forest Problem:
Given a graph $G = (V, E)$, together with source-target pairs $s_i, t_i, i = 1, \dots, k$, and a cost function $c : E \to \mathbb{R}^+$ on the edges. Find a subset $F \subseteq E$ of the edges such that for every $i \in \{1, \dots, k\}$ there is a path between s_i and t_i only using edges in F.

$$\begin{array}{llrcl} \min & & \sum_e c(e) x_e & & \\ \text{s.t.} & \forall S \subseteq V : S \in \mathcal{S}_i \text{ for some } i & \sum_{e \in \delta(S)} x_e & \geq & 1 \\ & \forall e \in E & x_e & \in & \{0, 1\} \end{array}$$

Here $\mathcal{S}_i$ contains all sets S such that $s_i \in S$ and $t_i \notin S$.

Steiner Forest Problem:

Given a graph $G = (V, E)$, together with source-target pairs $s_i, t_i, i = 1, \dots, k$, and a cost function $c : E \to \mathbb{R}^+$ on the edges. Find a subset $F \subseteq E$ of the edges such that for every $i \in \{1, \dots, k\}$ there is a path between s_i and t_i only using edges in F.

$$\begin{array}{llrcl} \min & & \sum_e c(e)x_e & & \\ \text{s.t.} & \forall S \subseteq V : S \in \mathcal{S}_i \text{ for some } i & \sum_{e\in\delta(S)} x_e & \geq & 1 \\ & \forall e \in E & x_e & \in & \{0, 1\} \end{array}$$

Here $\mathcal{S}_i$ contains all sets S such that $s_i \in S$ and $t_i \notin S$.

$$\begin{array}{llrcl}
\max & & \sum_{S\,:\,\exists i \text{ s.t. } S \in S_i} y_S & & \\
\text{s.t.} & \forall e \in E & \sum_{S: e \in \delta(S)} y_S & \le & c(e) \\
 & & y_S & \ge & 0
\end{array}$$

The difference to the dual of the shortest path problem is that we have many more variables (sets for which we can generate a moat of non-zero width).

Algorithm 1 FirstTry

$y \leftarrow 0$
$F \leftarrow \emptyset$
while not all s_i-t_i pairs connected in F **do**
    Let C be some connected component of (V, F) such that $|C \cap \{s_i, t_i\}| = 1$ for some i.
    Increase y_C until there is an edge $e' \in \delta(C)$ s.t. $\sum_{S \in S_i : e' \in \delta(S)} y_S = c_{e'}$
    $F \leftarrow F \cup \{e'\}$
return $\bigcup_i P_i$

$$\sum_{e \in F} c(e)$$

$$\sum_{e \in F} c(e) = \sum_{e \in F} \sum_{S : e \in \delta(S)} y_S$$

$$\sum_{e\in F} c(e) = \sum_{e\in F} \sum_{S:e\in\delta(S)} y_S = \sum_S |\delta(S) \cap F| \cdot y_S \enspace .$$

$$\sum_{e\in F} c(e) = \sum_{e\in F}\sum_{S:e\in\delta(S)} y_S = \sum_S |\delta(S)\cap F|\cdot y_S \enspace .$$

$$\sum_{e\in F} c(e) = \sum_{e\in F}\sum_{S: e\in\delta(S)} y_S = \sum_S |\delta(S)\cap F|\cdot y_S \enspace .$$

If we show that $y_S > 0$ implies that $|\delta(S)\cap F| \le \alpha$ we are in good shape.

However, this is not true:

- Take a complete graph on $k+1$ vertices $v_0, v_1, \ldots, v_k$.

$$\sum_{e\in F} c(e) = \sum_{e\in F} \sum_{S:e\in\delta(S)} y_S = \sum_S |\delta(S) \cap F| \cdot y_S \ .$$

If we show that $y_S > 0$ implies that $|\delta(S) \cap F| \leq \alpha$ we are in good shape.

However, this is not true:

- Take a complete graph on $k+1$ vertices $v_0, v_1, \ldots, v_k$.
- The i-th pair is v_0-v_i.

$$\sum_{e\in F} c(e) = \sum_{e\in F}\sum_{S:e\in\delta(S)} y_S = \sum_S |\delta(S)\cap F|\cdot y_S\ .$$

If we show that $y_S > 0$ implies that $|\delta(S)\cap F| \le \alpha$ we are in good shape.

However, this is not true:

- Take a complete graph on $k+1$ vertices $v_0, v_1, \dots, v_k$.
- The i-th pair is v_0-v_i.
- The first component C could be $\{v_0\}$.

$$\sum_{e\in F} c(e) = \sum_{e\in F} \sum_{S:e\in\delta(S)} y_S = \sum_{S} |\delta(S)\cap F| \cdot y_S \ .$$

If we show that $y_S > 0$ implies that $|\delta(S) \cap F| \leq \alpha$ we are in good shape.

However, this is not true:

- Take a complete graph on $k+1$ vertices $v_0, v_1, \ldots, v_k$.
- The i-th pair is v_0-v_i.
- The first component C could be $\{v_0\}$.
- We only set $y_{\{v_0\}} = 1$. All other dual variables stay 0.

$$\sum_{e\in F} c(e) = \sum_{e\in F}\sum_{S:e\in\delta(S)} y_S = \sum_S |\delta(S)\cap F|\cdot y_S \enspace .$$

If we show that $y_S > 0$ implies that $|\delta(S)\cap F| \le \alpha$ we are in good shape.

However, this is not true:

- Take a complete graph on $k+1$ vertices $v_0, v_1, \dots, v_k$.
- The i-th pair is v_0-v_i.
- The first component C could be $\{v_0\}$.
- We only set $y_{\{v_0\}} = 1$. All other dual variables stay 0.
- The final set F contains all edges $\{v_0, v_i\}$, $i = 1, \dots, k$.

$$\sum_{e\in F} c(e) = \sum_{e\in F} \sum_{S:e\in\delta(S)} y_S = \sum_S |\delta(S)\cap F| \cdot y_S \ .$$

If we show that $y_S > 0$ implies that $|\delta(S)\cap F| \le \alpha$ we are in good shape.

However, this is not true:

- Take a complete graph on $k+1$ vertices $v_0, v_1, \dots, v_k$.
- The i-th pair is v_0-v_i.
- The first component C could be $\{v_0\}$.
- We only set $y_{\{v_0\}} = 1$. All other dual variables stay 0.
- The final set F contains all edges $\{v_0, v_i\}$, $i = 1, \dots, k$.
- $y_{\{v_0\}} > 0$ but $|\delta(\{v_0\})\cap F| = k$.

Algorithm 1 SecondTry

$y \leftarrow 0$; $F \leftarrow \emptyset$; $\ell \leftarrow 0$

while not all s_i-t_i pairs connected in F **do**

  $\ell \leftarrow \ell + 1$

  Let $\mathcal{C}$ be set of all connected components C of (V, F) such that $|C \cap \{s_i, t_i\}| = 1$ for some i.

  Increase y_C for all $C \in \mathcal{C}$ uniformly until for some edge $e_\ell \in \delta(C')$, $C' \in \mathcal{C}$ s.t. $\sum_{S: e_\ell \in \delta(S)} y_S = c_{e_\ell}$

  $F \leftarrow F \cup \{e_\ell\}$

$F' \leftarrow F$

for $k \leftarrow \ell$ downto 1 **do** // reverse deletion

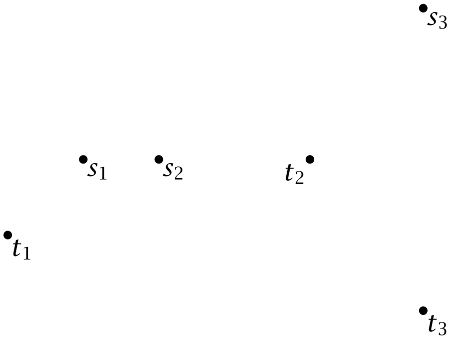
  if $F' - e_k$ is feasible solution **then**

    remove e_k from F'

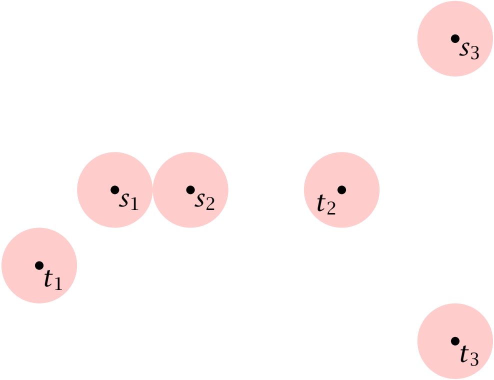
return F'

The reverse deletion step is not strictly necessary this way. It would also be sufficient to simply delete all unnecessary edges in any order.

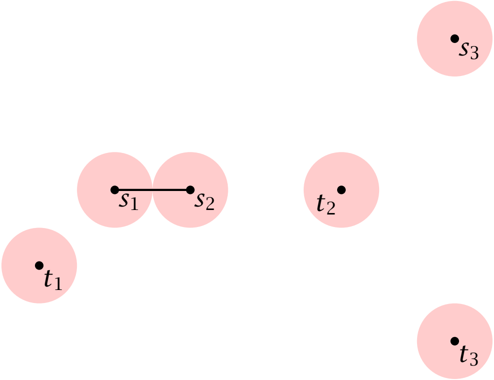
Example



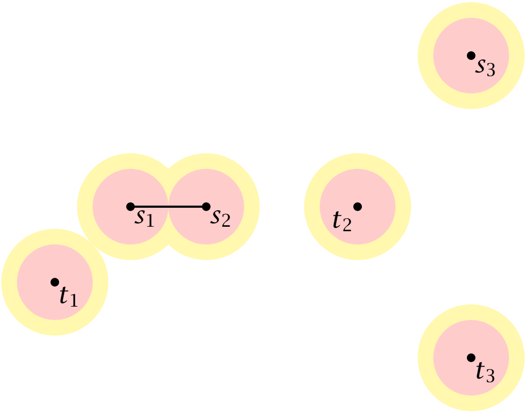
Example



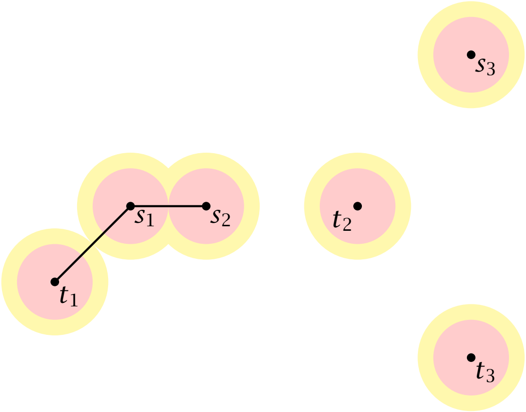
Example



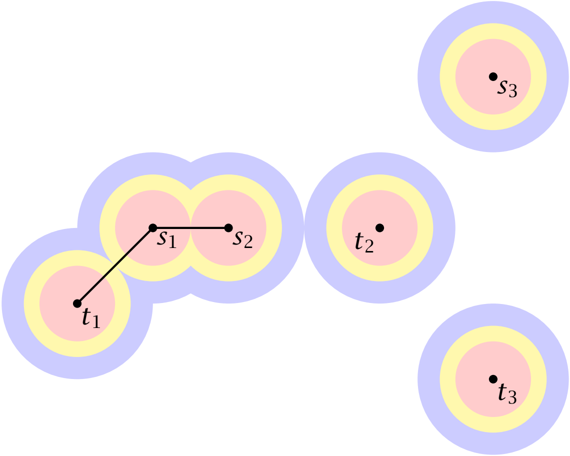
Example



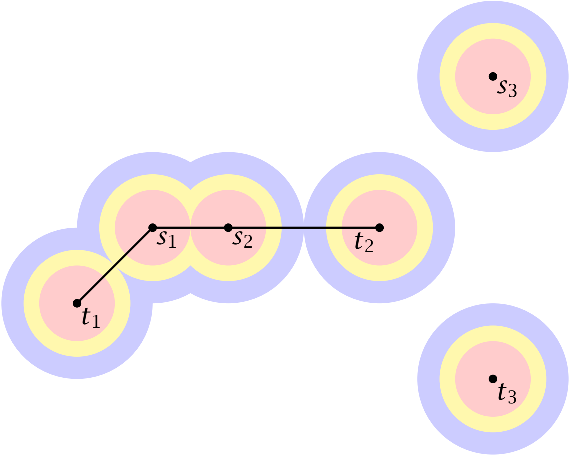
Example



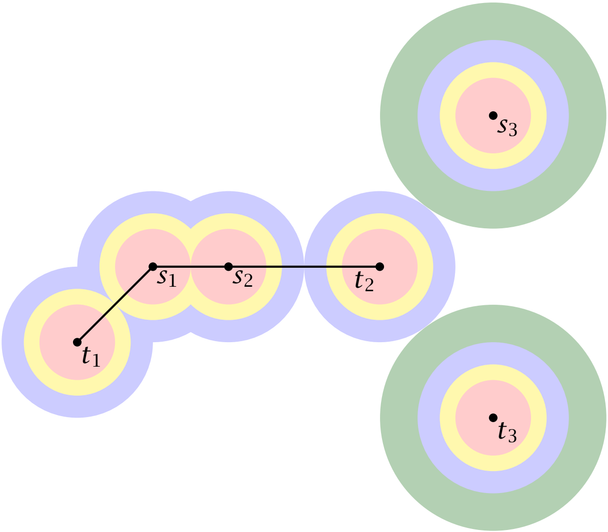
Example



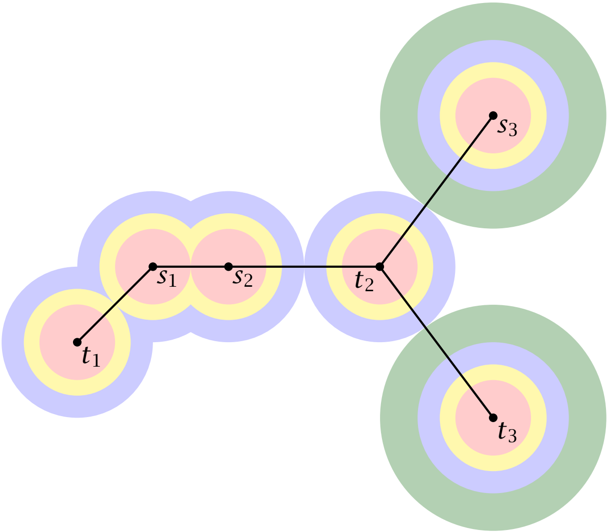
Example



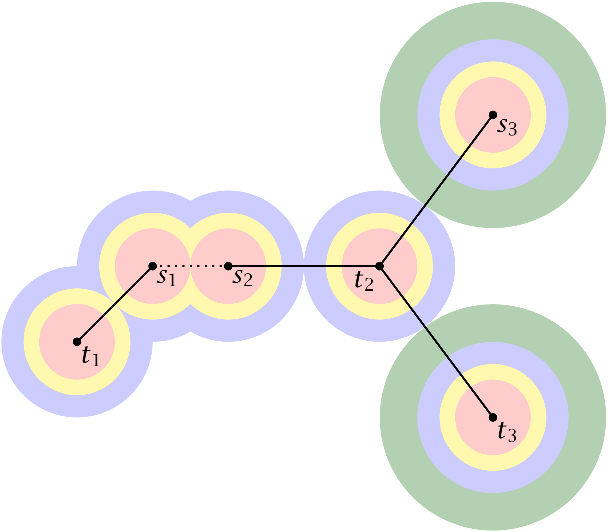
Example



Example



Example



Lemma 85

For any C in any iteration of the algorithm

$$\sum_{C \in \mathcal{C}} |\delta(C) \cap F'| \leq 2|\mathcal{C}|$$

This means that the number of times a moat from $\mathcal{C}$ is crossed in the final solution is at most twice the number of moats.

Proof: later...

$$\sum_{e \in F'} c_e = \sum_{e \in F'} \sum_{S : e \in \delta(S)} y_S = \sum_S |F' \cap \delta(S)| \cdot y_S \ .$$

We want to show that

$$\sum_S |F' \cap \delta(S)| \cdot y_S \le 2 \sum_S y_S$$

- In the i-th iteration the increase of the left hand side is

$$\epsilon \sum_{C \in \mathcal{C}} |F' \cap \delta(C)|$$

and the increase of the right hand side is $2\epsilon|\mathcal{C}|$.

- Hence, by the previous lemma the inequality holds after the iteration if it holds in the beginning of the iteration.

$$\sum_{e \in F'} c_e = \sum_{e \in F'} \sum_{S : e \in \delta(S)} y_S$$

$$\sum_{e\in F'} c_e = \sum_{e\in F'} \sum_{S:e\in\delta(S)} y_S = \sum_S |F'\cap\delta(S)|\cdot y_S \ .$$

We want to show that

$$\sum_S |F'\cap\delta(S)|\cdot y_S \le 2\sum_S y_S$$

In the i-th iteration the increase of the left-hand side is

$$\epsilon\sum_{C\in\mathcal{C}} |F'\cap\delta(C)|$$

and the increase of the right hand side is $2\epsilon|\mathcal{C}|$.

Hence, by the previous lemma the inequality holds after the iteration if it holds in the beginning of the iteration.

$$\sum_{e\in F'} c_e = \sum_{e\in F'} \sum_{S:e\in\delta(S)} y_S = \sum_S |F' \cap \delta(S)| \cdot y_S \ .$$

We want to show that

$$\sum_S |F' \cap \delta(S)| \cdot y_S \le 2 \sum_S y_S$$

$$\sum_{e\in F'} c_e = \sum_{e\in F'} \sum_{S:e\in\delta(S)} y_S = \sum_S |F' \cap \delta(S)| \cdot y_S \ .$$

We want to show that

$$\sum_S |F' \cap \delta(S)| \cdot y_S \le 2 \sum_S y_S$$

- In the i-th iteration the increase of the left-hand side is

$$\epsilon \sum_{C\in C} |F' \cap \delta(C)|$$

  and the increase of the right hand side is $2\epsilon|C|$.
- Hence, by the previous lemma the inequality holds after the iteration if it holds in the beginning of the iteration.

$$\sum_{e\in F'} c_e = \sum_{e\in F'} \sum_{S:e\in\delta(S)} y_S = \sum_S |F' \cap \delta(S)| \cdot y_S \enspace .$$

We want to show that

$$\sum_S |F' \cap \delta(S)| \cdot y_S \le 2 \sum_S y_S$$

- In the i-th iteration the increase of the left-hand side is

$$\epsilon \sum_{C\in C} |F' \cap \delta(C)|$$

  and the increase of the right hand side is $2\epsilon|C|$.
- Hence, by the previous lemma the inequality holds after the iteration if it holds in the beginning of the iteration.

Lemma 86

For any set of connected components $\mathcal{C}$ in any iteration of the algorithm

$$\sum_{C \in \mathcal{C}} |\delta(C) \cap F'| \leq 2|\mathcal{C}|$$

Proof:

- At any point during the algorithm the set of edges forms a forest (why?).
- Fix iteration i. e_i is the set we add to F. Let F_i be the set of edges in F at the beginning of the iteration.
- Let $H = F' - F_i$.
- All edges in H are necessary for the solution.

Lemma 86

For any set of connected components $\mathcal{C}$ in any iteration of the algorithm

$$\sum_{C \in \mathcal{C}} |\delta(C) \cap F'| \leq 2|\mathcal{C}|$$

Proof:

- At any point during the algorithm the set of edges forms a forest (why?).
- Fix iteration i. e_i is the set we add to F. Let F_i be the set of edges in F at the beginning of the iteration.
- Let $H = F' - F_i$.
- All edges in H are necessary for the solution.

Lemma 86

For any set of connected components $\mathcal{C}$ in any iteration of the algorithm

$$\sum_{C\in\mathcal{C}} |\delta(C)\cap F'| \le 2|\mathcal{C}|$$

Proof:

- At any point during the algorithm the set of edges forms a forest (why?).
- Fix iteration i. e_i is the set we add to F. Let F_i be the set of edges in F at the beginning of the iteration.
- Let $H = F' - F_i$.
- All edges in H are necessary for the solution.

Lemma 86

For any set of connected components $\mathcal{C}$ in any iteration of the algorithm

$$\sum_{C \in \mathcal{C}} |\delta(C) \cap F'| \leq 2|\mathcal{C}|$$

Proof:

- At any point during the algorithm the set of edges forms a forest (why?).
- Fix iteration i. e_i is the set we add to F. Let F_i be the set of edges in F at the beginning of the iteration.
- Let $H = F' - F_i$.
- All edges in H are necessary for the solution.

Lemma 86

For any set of connected components $\mathcal{C}$ in any iteration of the algorithm

$$\sum_{C \in \mathcal{C}} |\delta(C) \cap F'| \leq 2|\mathcal{C}|$$

Proof:

- At any point during the algorithm the set of edges forms a forest (why?).
- Fix iteration i. e_i is the set we add to F. Let F_i be the set of edges in F at the beginning of the iteration.
- Let $H = F' - F_i$.
- All edges in H are necessary for the solution.

- Contract all edges in F_i into single vertices V'.
- We can consider the forest H on the set of vertices V'.
- Let $\deg(v)$ be the degree of a vertex $v \in V'$ within this forest.
- Color a vertex $v \in V'$ red if it corresponds to a component from C (an active component). Otw. color it blue. (Let B the set of blue vertices (with non-zero degree) and R the set of red vertices)
- We have

$$\sum_{v \in R} \deg(v) \geq \sum_{C \in C} |\delta(C) \cap F'| \overset{?}{\leq} 2|C| = 2|R|$$

- Contract all edges in F_i into single vertices V'.
- We can consider the forest H on the set of vertices V'.
- Let $\deg(v)$ be the degree of a vertex $v \in V'$ within this forest.
- Color a vertex $v \in V'$ red if it corresponds to a component from C (an active component). Otw. color it blue. (Let B the set of blue vertices (with non-zero degree) and R the set of red vertices)
- We have

$$\sum_{v\in R} \deg(v) \ge \sum_{C\in C} |\delta(C) \cap F'| \overset{?}{\le} 2|C| = 2|R|$$

- Contract all edges in F_i into single vertices V'.
- We can consider the forest H on the set of vertices V'.
- Let $\deg(v)$ be the degree of a vertex $v \in V'$ within this forest.
- Color a vertex $v \in V'$ red if it corresponds to a component from C (an active component). Otw. color it blue. (Let B the set of blue vertices (with non-zero degree) and R the set of red vertices)
- We have

$$\sum_{v \in R} \deg(v) \geq \sum_{C \in C} |\delta(C) \cap F'| \overset{?}{\leq} 2|C| = 2|R|$$

- Contract all edges in F_i into single vertices V'.
- We can consider the forest H on the set of vertices V'.
- Let $\deg(v)$ be the degree of a vertex $v \in V'$ within this forest.
- Color a vertex $v \in V'$ red if it corresponds to a component from C (an active component). Otw. color it blue. (Let B the set of blue vertices (with non-zero degree) and R the set of red vertices)
- We have

$$\sum_{v \in R} \deg(v) \geq \sum_{C \in C} |\delta(C) \cap F'| \overset{?}{\leq} 2|C| = 2|R|$$

- Contract all edges in F_i into single vertices V'.
- We can consider the forest H on the set of vertices V'.
- Let $\deg(v)$ be the degree of a vertex $v \in V'$ within this forest.
- Color a vertex $v \in V'$ red if it corresponds to a component from C (an active component). Otw. color it blue. (Let B the set of blue vertices (with non-zero degree) and R the set of red vertices)
- We have

$$\sum_{v \in R} \deg(v) \geq \sum_{C \in C} |\delta(C) \cap F'| \stackrel{?}{\leq} 2|C| = 2|R|$$

- Suppose that no node in B has degree one.

- Suppose that no node in B has degree one.
- Then

- Suppose that no node in B has degree one.
- Then

$$\sum_{v \in R} \deg(v)$$

- Suppose that no node in B has degree one.
- Then

$$\sum_{v \in R} \deg(v) = \sum_{v \in R \cup B} \deg(v) - \sum_{v \in B} \deg(v)$$

- Suppose that no node in B has degree one.
- Then

$$\sum_{v\in R}\deg(v) = \sum_{v\in R\cup B}\deg(v) - \sum_{v\in B}\deg(v)$$
$$\leq 2(|R| + |B|) - 2|B|$$

- Suppose that no node in B has degree one.
- Then

$$\sum_{v \in R} \deg(v) = \sum_{v \in R \cup B} \deg(v) - \sum_{v \in B} \deg(v)$$
$$\le 2(|R| + |B|) - 2|B| = 2|R|$$

- Suppose that no node in B has degree one.
- Then

$$\sum_{v\in R}\deg(v) = \sum_{v\in R\cup B}\deg(v) - \sum_{v\in B}\deg(v)$$
$$\leq 2(|R| + |B|) - 2|B| = 2|R|$$

- Every blue vertex with non-zero degree must have degree at least two.

- Suppose that no node in B has degree one.
- Then

$$\begin{aligned}\sum_{v\in R}\deg(v) &= \sum_{v\in R\cup B}\deg(v) - \sum_{v\in B}\deg(v)\\ &\le 2(|R|+|B|) - 2|B| = 2|R|\end{aligned}$$

- Every blue vertex with non-zero degree must have degree at least two.
  - Suppose not. The single edge connecting $b \in B$ comes from H, and, hence, is necessary.

- Suppose that no node in B has degree one.
- Then

$$\sum_{v \in R} \deg(v) = \sum_{v \in R \cup B} \deg(v) - \sum_{v \in B} \deg(v)$$
$$\leq 2(|R| + |B|) - 2|B| = 2|R|$$

- Every blue vertex with non-zero degree must have degree at least two.
  - Suppose not. The single edge connecting $b \in B$ comes from H, and, hence, is necessary.
  - But this means that the cluster corresponding to b must separate a source-target pair.

- Suppose that no node in B has degree one.
- Then

$$\sum_{v\in R}\deg(v) = \sum_{v\in R\cup B}\deg(v) - \sum_{v\in B}\deg(v)$$
$$\le 2(|R|+|B|) - 2|B| = 2|R|$$

- Every blue vertex with non-zero degree must have degree at least two.
  - Suppose not. The single edge connecting $b \in B$ comes from H, and, hence, is necessary.
  - But this means that the cluster corresponding to b must separate a source-target pair.
  - But then it must be a red node.

20 Cuts & Metrics

Shortest Path

$$\begin{array}{llrcl} \min & & \sum_e c(e)x_e & & \\ \text{s.t.} & \forall S \in \mathcal{S} & \sum_{e:\delta(S)} x_e & \geq & 1 \\ & \forall e \in E & x_e & \in & \{0,1\} \end{array}$$

$\mathcal{S}$ is the set of subsets that separate s from t.

The Dual:

$$\begin{array}{llrcl} \max & & \sum_S y_S & & \\ \text{s.t.} & \forall e \in E & \sum_{S:e\in\delta(S)} y_S & \leq & c(e) \\ & \forall S \in \mathcal{S} & y_S & \geq & 0 \end{array}$$

The Separation Problem for the Shortest Path LP is the Minimum Cut Problem.

20 Cuts & Metrics

Shortest Path

$$\begin{array}{llrcl} \min & & \sum_e c(e) x_e & & \\ \text{s.t.} & \forall S \in \mathcal{S} & \sum_{e:\delta(S)} x_e & \geq & 1 \\ & \forall e \in E & x_e & \geq & 0 \end{array}$$

$\mathcal{S}$ is the set of subsets that separate s from t.

The Dual:

$$\begin{array}{llrcl} \max & & \sum_S y_S & & \\ \text{s.t.} & \forall e \in E & \sum_{S:e\in\delta(S)} y_S & \leq & c(e) \\ & \forall S \in \mathcal{S} & y_S & \geq & 0 \end{array}$$

The Separation Problem for the Shortest Path LP is the Minimum Cut Problem.

20 Cuts & Metrics

Shortest Path

$$\begin{array}{llrcl} \min & & \sum_e c(e) x_e & & \\ \text{s.t.} & \forall S \in \mathcal{S} & \sum_{e:\delta(S)} x_e & \geq & 1 \\ & \forall e \in E & x_e & \geq & 0 \end{array}$$

$\mathcal{S}$ is the set of subsets that separate s from t.

The Dual:

$$\begin{array}{llrcl} \max & & \sum_S y_S & & \\ \text{s.t.} & \forall e \in E & \sum_{S:e\in\delta(S)} y_S & \leq & c(e) \\ & \forall S \in \mathcal{S} & y_S & \geq & 0 \end{array}$$

The Separation Problem for the Shortest Path LP is the Minimum Cut Problem.

20 Cuts & Metrics

Minimum Cut

$$
\begin{array}{llrcl}
\min & & \sum_e c(e) x_e & & \\
\text{s.t.} & \forall P \in \mathcal{P} & \sum_{e \in P} x_e & \geq & 1 \\
& \forall e \in E & x_e & \in & \{0,1\}
\end{array}
$$

$\mathcal{P}$ is the set of path that connect s and t.

The Dual:

$$
\begin{array}{llrcl}
\max & & \sum_P y_P & & \\
\text{s.t.} & \forall e \in E & \sum_{P: e \in P} y_P & \leq & c(e) \\
& \forall P \in \mathcal{P} & y_P & \geq & 0
\end{array}
$$

The Separation Problem for the Minimum Cut LP is the Shortest Path Problem.

20 Cuts & Metrics

Minimum Cut

$$\begin{array}{llrcl} \min & & \sum_e c(e) x_e & & \\ \text{s.t.} & \forall P \in \mathcal{P} & \sum_{e \in P} x_e & \geq & 1 \\ & \forall e \in E & x_e & \geq & 0 \end{array}$$

$\mathcal{P}$ is the set of path that connect s and t.

The Dual:

$$\begin{array}{llrcl} \max & & \sum_P y_P & & \\ \text{s.t.} & \forall e \in E & \sum_{P : e \in P} y_P & \leq & c(e) \\ & \forall P \in \mathcal{P} & y_P & \geq & 0 \end{array}$$

The Separation Problem for the Minimum Cut LP is the Shortest Path Problem.

20 Cuts & Metrics

Minimum Cut

$$\begin{array}{llrcl} \min & & \sum_e c(e)x_e & & \\ \text{s.t.} & \forall P \in \mathcal{P} & \sum_{e \in P} x_e & \geq & 1 \\ & \forall e \in E & x_e & \geq & 0 \end{array}$$

$\mathcal{P}$ is the set of path that connect s and t.

The Dual:

$$\begin{array}{llrcl} \max & & \sum_P y_P & & \\ \text{s.t.} & \forall e \in E & \sum_{P: e \in P} y_P & \leq & c(e) \\ & \forall P \in \mathcal{P} & y_P & \geq & 0 \end{array}$$

The Separation Problem for the Minimum Cut LP is the Shortest Path Problem.

20 Cuts & Metrics

Minimum Cut

$$\begin{array}{llrcl} \min & & \sum_e c(e)\ell_e & & \\ \text{s.t.} & \forall P \in \mathcal{P} & \sum_{e \in P} \ell_e & \geq & 1 \\ & \forall e \in E & \ell_e & \geq & 0 \end{array}$$

$\mathcal{P}$ is the set of path that connect s and t.

The Dual:

$$\begin{array}{llrcl} \max & & \sum_P f_P & & \\ \text{s.t.} & \forall e \in E & \sum_{P: e \in P} f_P & \leq & c(e) \\ & \forall P \in \mathcal{P} & f_P & \geq & 0 \end{array}$$

The Separation Problem for the Minimum Cut LP is the Shortest Path Problem.

20 Cuts & Metrics

Observations:

Suppose that ℓ_e-values are solution to Minimum Cut LP.

- We can view ℓ_e as defining the length of an edge.
- Define $d(u,v) = \min_{\text{path } P \text{ btw. } u \text{ and } v} \sum_{e \in P} \ell_e$ as the Shortest Path Metric induced by ℓ_e.
- We have $d(u,v) = \ell_e$ for every edge $e = (u,v)$, as otw. we could reduce ℓ_e without affecting the distance between s and t.

Remark for bean-counters:

d is not a metric on V but a semimetric as two nodes u and v could have distance zero.

Observations:

Suppose that ℓ_e-values are solution to Minimum Cut LP.

- We can view ℓ_e as defining the length of an edge.
- Define $d(u,v) = \min_{\text{path } P \text{ btw. } u \text{ and } v} \sum_{e \in P} \ell_e$ as the Shortest Path Metric induced by ℓ_e.
- We have $d(u,v) = \ell_e$ for every edge $e = (u,v)$, as otw. we could reduce ℓ_e without affecting the distance between s and t.

Remark for bean-counters:

d is not a metric on V but a semimetric as two nodes u and v could have distance zero.

20 Cuts & Metrics

Observations:

Suppose that ℓ_e-values are solution to Minimum Cut LP.

- We can view ℓ_e as defining the length of an edge.
- Define $d(u,v) = \min_{\text{path } P \text{ btw. } u \text{ and } v} \sum_{e\in P} \ell_e$ as the Shortest Path Metric induced by ℓ_e.
- We have $d(u,v) = \ell_e$ for every edge $e = (u,v)$, as otw. we could reduce ℓ_e without affecting the distance between s and t.

Remark for bean-counters:

d is not a metric on V but a semimetric as two nodes u and v could have distance zero.

20 Cuts & Metrics

Observations:

Suppose that ℓ_e-values are solution to Minimum Cut LP.

- We can view ℓ_e as defining the length of an edge.
- Define $d(u,v) = \min_{\text{path } P \text{ btw. } u \text{ and } v} \sum_{e \in P} \ell_e$ as the Shortest Path Metric induced by ℓ_e.
- We have $d(u,v) = \ell_e$ for every edge $e = (u,v)$, as otw. we could reduce ℓ_e without affecting the distance between s and t.

Remark for bean-counters:
d is not a metric on V but a semimetric as two nodes u and v could have distance zero.

How do we round the LP?

- Let $B(s,r)$ be the ball of radius r around s (w.r.t. metric d). Formally:

$$B = \{v \in V \mid d(s,v) \le r\}$$

- For $0 \le r < 1$, $B(s,r)$ is an s-t-cut.

Which value of r should we choose? choose randomly!!!

Formally:

choose r u.a.r. (uniformly at random) from interval $[0,1)$

How do we round the LP?

- Let $B(s,r)$ be the ball of radius r around s (w.r.t. metric d). Formally:

$$B = \{v \in V \mid d(s,v) \leq r\}$$

- For $0 \leq r < 1$, $B(s,r)$ is an s-t-cut.

Which value of r should we choose? choose randomly!!!

Formally:
choose r u.a.r. (uniformly at random) from interval $[0,1)$

How do we round the LP?

- Let $B(s,r)$ be the ball of radius r around s (w.r.t. metric d). Formally:

$$B = \{v \in V \mid d(s,v) \leq r\}$$

- For $0 \leq r < 1$, $B(s,r)$ is an s-t-cut.

Which value of r should we choose? choose randomly!!!

Formally:

choose r u.a.r. (uniformly at random) from interval $[0,1)$

How do we round the LP?

- Let $B(s,r)$ be the ball of radius r around s (w.r.t. metric d). Formally:

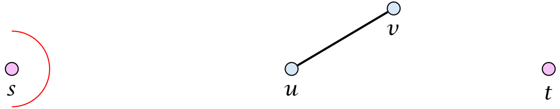
$$B = \{v \in V \mid d(s,v) \le r\}$$

- For $0 \le r < 1$, $B(s,r)$ is an s-t-cut.

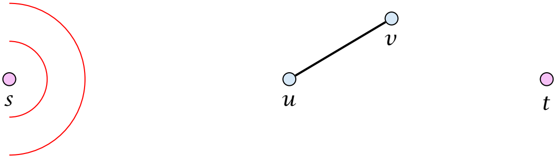
Which value of r should we choose? choose randomly!!!

Formally:
choose r u.a.r. (uniformly at random) from interval $[0,1)$

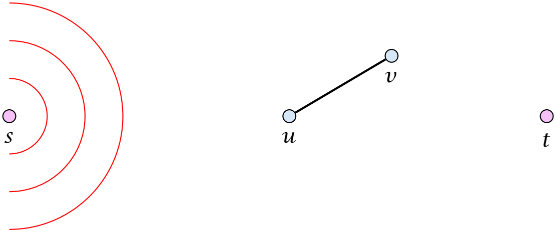
What is the probability that an edge (u, v) is in the cut?



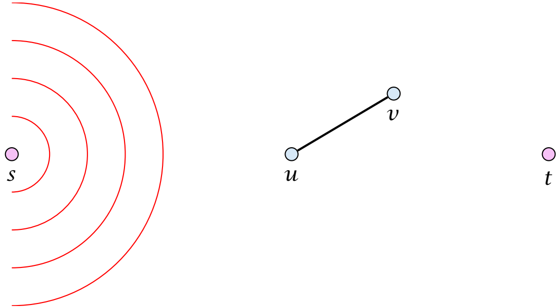
What is the probability that an edge (u, v) is in the cut?



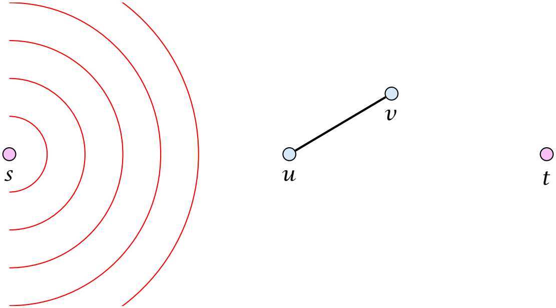
What is the probability that an edge (u, v) is in the cut?



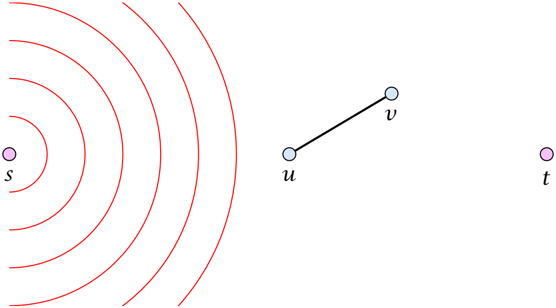
What is the probability that an edge (u, v) is in the cut?



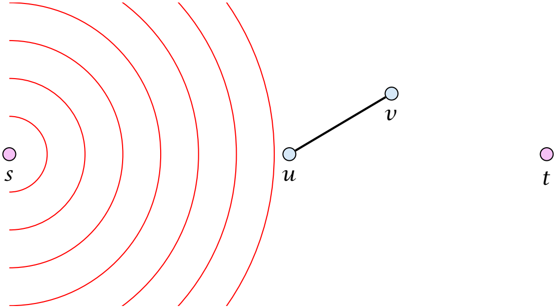
What is the probability that an edge (u, v) is in the cut?



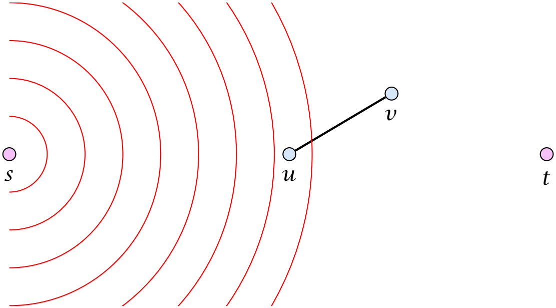
What is the probability that an edge (u, v) is in the cut?



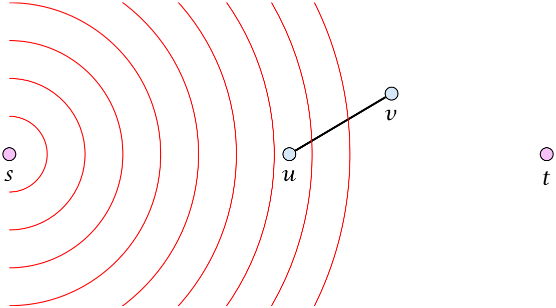
What is the probability that an edge (u, v) is in the cut?



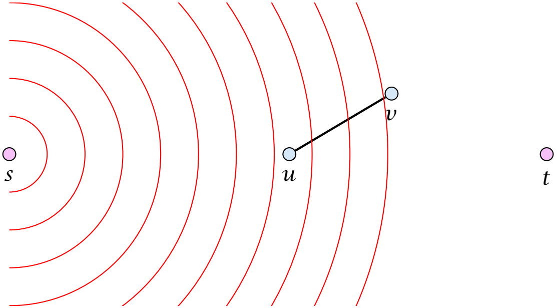
What is the probability that an edge (u, v) is in the cut?



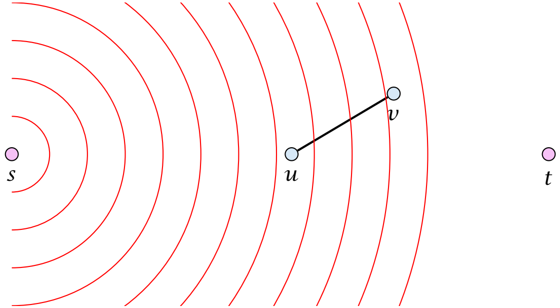
What is the probability that an edge (u, v) is in the cut?



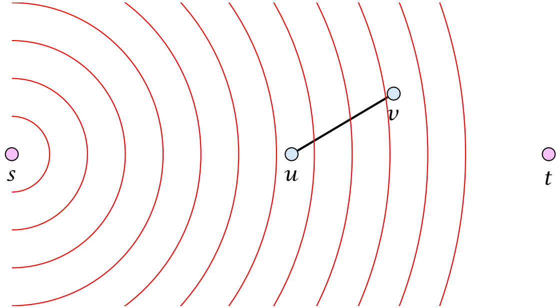
What is the probability that an edge (u, v) is in the cut?



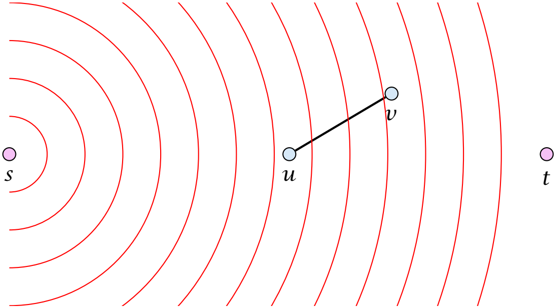
What is the probability that an edge (u, v) is in the cut?



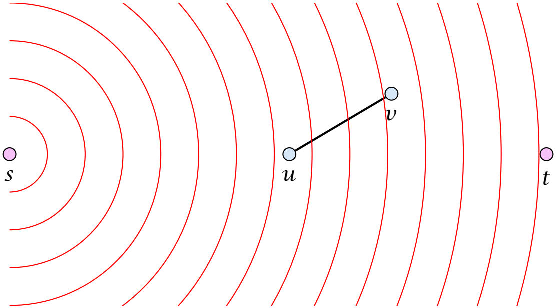
What is the probability that an edge (u, v) is in the cut?



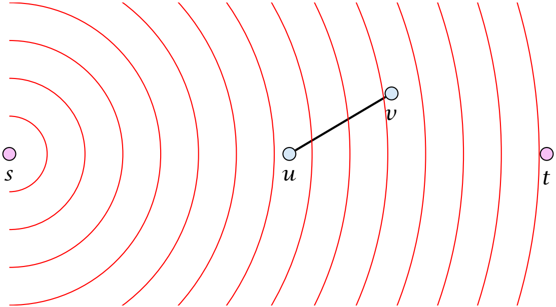
What is the probability that an edge (u, v) is in the cut?



What is the probability that an edge (u, v) is in the cut?



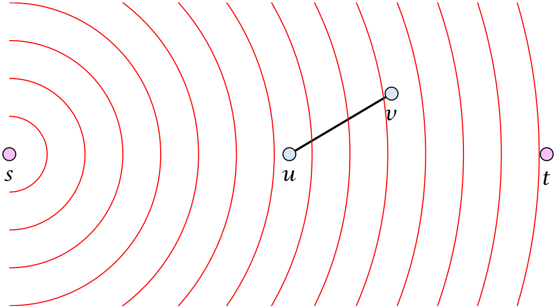
What is the probability that an edge (u, v) is in the cut?



- asssume wlog. $d(s,u) \leq d(s,v)$

$$\Pr[e \text{ is cut}]$$

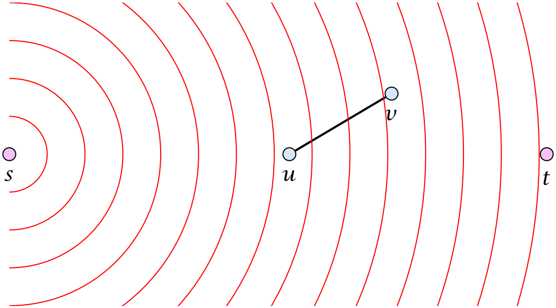
What is the probability that an edge (u, v) is in the cut?



- asssume wlog. $d(s,u) \leq d(s,v)$

$$\Pr[e \text{ is cut}] = \Pr[r \in [d(s,u), d(s,v))]$$

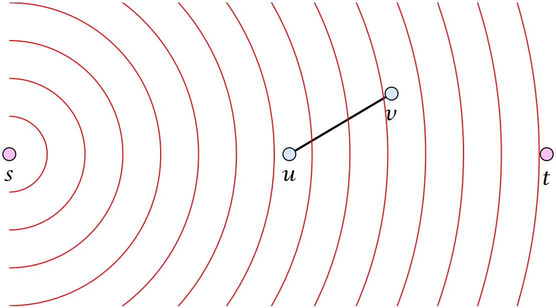
What is the probability that an edge (u, v) is in the cut?



- asssume wlog. $d(s,u) \le d(s,v)$

$$\Pr[e \text{ is cut}] = \Pr[r \in [d(s,u), d(s,v))] \le \frac{d(s,v) - d(s,u)}{1-0}$$

What is the probability that an edge (u, v) is in the cut?



- asssume wlog. $d(s,u) \le d(s,v)$

$$\Pr[e \text{ is cut}] = \Pr[r \in [d(s,u), d(s,v))] \le \frac{d(s,v) - d(s,u)}{1-0}$$

$$\le \ell_e$$

What is the expected size of a cut?

$$\begin{aligned}\mathrm{E}[\text{size of cut}] &= \mathrm{E}[\textstyle\sum_e c(e)\Pr[e \text{ is cut}]] \\ &\le \textstyle\sum_e c(e)\ell_e\end{aligned}$$

On the other hand:

$$\textstyle\sum_e c(e)\ell_e \le \text{size of mincut}$$

as the ℓ_e are the solution to the Mincut LP *relaxation*.

Hence, our rounding gives an optimal solution.

What is the expected size of a cut?

$$\begin{aligned}\mathrm{E}[\text{size of cut}] &= \mathrm{E}\big[\textstyle\sum_e c(e)\Pr[e \text{ is cut}]\big]\\ &\leq \textstyle\sum_e c(e)\ell_e\end{aligned}$$

On the other hand:

$$\textstyle\sum_e c(e)\ell_e \leq \text{size of mincut}$$

as the ℓ_e are the solution to the Mincut LP *relaxation*.

Hence, our rounding gives an optimal solution.

What is the expected size of a cut?

$$\begin{aligned} \mathrm{E}[\text{size of cut}] &= \mathrm{E}\big[\textstyle\sum_e c(e)\,\Pr[e \text{ is cut}]\big] \\ &\le \textstyle\sum_e c(e)\ell_e \end{aligned}$$

On the other hand:

$$\textstyle\sum_e c(e)\ell_e \le \text{size of mincut}$$

as the ℓ_e are the solution to the Mincut LP *relaxation*.

Hence, our rounding gives an optimal solution.

Minimum Multicut:

Given a graph $G = (V, E)$, together with source-target pairs s_i, t_i, $i = 1, \dots, k$, and a capacity function $c : E \to \mathbb{R}^+$ on the edges. Find a subset $F \subseteq E$ of the edges such that all s_i-t_i pairs lie in different components in $G = (V, E \setminus F)$.

$$\begin{array}{llcll} \min & & \sum_e c(e)\ell_e & & \\ \text{s.t.} & \forall P \in \mathcal{P}_i \text{ for some } i & \sum_{e \in P} \ell_e & \geq & 1 \\ & \forall e \in E & \ell_e & \in & \{0,1\} \end{array}$$

Here $\mathcal{P}_i$ contains all path P between s_i and t_i.

Minimum Multicut:

Given a graph $G = (V, E)$, together with source-target pairs s_i, t_i, $i = 1, \dots, k$, and a capacity function $c : E \to \mathbb{R}^+$ on the edges. Find a subset $F \subseteq E$ of the edges such that all s_i-t_i pairs lie in different components in $G = (V, E \setminus F)$.

$$\begin{array}{llrcl} \min & & \sum_e c(e)\ell_e & & \\ \text{s.t.} & \forall P \in \mathcal{P}_i \text{ for some } i & \sum_{e \in P} \ell_e & \geq & 1 \\ & \forall e \in E & \ell_e & \in & \{0, 1\} \end{array}$$

Here $\mathcal{P}_i$ contains all path P between s_i and t_i.

Minimum Multicut:
Given a graph $G = (V, E)$, together with source-target pairs s_i, t_i, $i = 1, \dots, k$, and a capacity function $c : E \to \mathbb{R}^+$ on the edges. Find a subset $F \subseteq E$ of the edges such that all s_i-t_i pairs lie in different components in $G = (V, E \setminus F)$.

$$
\begin{array}{llrcl}
\min & & \sum_e c(e)\ell_e & & \\
\text{s.t.} & \forall P \in \mathcal{P}_i \text{ for some } i & \sum_{e\in P} \ell_e & \geq & 1 \\
 & \forall e \in E & \ell_e & \in & \{0,1\}
\end{array}
$$

Here $\mathcal{P}_i$ contains all path P between s_i and t_i.

Re-using the analysis for the single-commodity case is difficult.

$$\Pr[e \text{ is cut}] \leq ?$$

- If for some R the balls $B(s_i, R)$ are disjoint between different sources, we get a $1/R$ approximation.
- However, this cannot be guaranteed.

Re-using the analysis for the single-commodity case is difficult.

$$\Pr[e \text{ is cut}] \le \textbf{?}$$

- If for some R the balls $B(s_i, R)$ are disjoint between different sources, we get a $1/R$ approximation.
- However, this cannot be guaranteed.

Re-using the analysis for the single-commodity case is difficult.

$$\Pr[e \text{ is cut}] \leq \textbf{?}$$

- If for some R the balls $B(s_i, R)$ are disjoint between different sources, we get a $1/R$ approximation.
- However, this cannot be guaranteed.

- Assume for simplicity that all edge-length ℓ_e are multiples of $\delta \ll 1$.
- Replace the graph G by a graph G', where an edge of length ℓ_e is replaced by ℓ_e/δ edges of length δ.
- Let $B(s_i, z)$ be the ball in G' that contains nodes v with distance $d(s_i, v) \le z\delta$.

Algorithm 1 RegionGrowing(s_i, p)

```
z ← 0
repeat
    flip a coin (Pr[heads] = p)
    z ← z + 1
until heads
return B(s_i, z)
```

- Assume for simplicity that all edge-length ℓ_e are multiples of $\delta \ll 1$.
- Replace the graph G by a graph G', where an edge of length ℓ_e is replaced by ℓ_e/δ edges of length δ.
- Let $B(s_i, z)$ be the ball in G' that contains nodes v with distance $d(s_i, v) \le z\delta$.

Algorithm 1 RegionGrowing(s_i, p)

```
1: z ← 0
2: repeat
3:     flip a coin (Pr[heads] = p)
4:     z ← z + 1
5: until heads
6: return B(s_i, z)
```

- Assume for simplicity that all edge-length ℓ_e are multiples of $\delta \ll 1$.
- Replace the graph G by a graph G', where an edge of length ℓ_e is replaced by ℓ_e/δ edges of length δ.
- Let $B(s_i, z)$ be the ball in G' that contains nodes v with distance $d(s_i, v) \le z\delta$.

Algorithm 1 RegionGrowing(s_i, p)

```
z ← 0
repeat
    flip a coin (Pr[heads] = p)
    z ← z + 1
until heads
return B(s_i, z)
```

- Assume for simplicity that all edge-length ℓ_e are multiples of $\delta \ll 1$.
- Replace the graph G by a graph G', where an edge of length ℓ_e is replaced by ℓ_e/δ edges of length δ.
- Let $B(s_i, z)$ be the ball in G' that contains nodes v with distance $d(s_i, v) \leq z\delta$.

Algorithm 1 RegionGrowing(s_i, p)

1: $z \leftarrow 0$
2: **repeat**
3: flip a coin $(\Pr[\text{heads}] = p)$
4: $z \leftarrow z + 1$
5: **until** heads
6: **return** $B(s_i, z)$

Algorithm 1 Multicut(G')

while $\exists s_i$-t_i pair in G' **do**
  $C \leftarrow$ RegionGrowing(s_i, p)
  $G' = G' \setminus C$ // cuts edges leaving C
return $B(s_i, z)$

- probability of cutting an edge is only p
- a source either does not reach an edge during Region Growing; then it is not cut
- if it reaches the edge then it either cuts the edge or protects the edge from being cut by other sources
- if we choose $p = \delta$ the probability of cutting an edge is only its LP-value; our expected cost are at most OPT.

Algorithm 1 Multicut(G')

while $\exists s_i$-t_i pair in G' **do**
  $C \leftarrow$ RegionGrowing(s_i, p)
  $G' = G' \setminus C$ // cuts edges leaving C
return $B(s_i, z)$

- probability of cutting an edge is only $\boldsymbol{p}$
- a source either does not reach an edge during Region Growing; then it is not cut
- if it reaches the edge then it either cuts the edge or protects the edge from being cut by other sources
- if we choose $p = \delta$ the probability of cutting an edge is only its LP-value; our expected cost are at most OPT.

Algorithm 1 Multicut(G')

while $\exists s_i$-t_i pair in G' **do**

$\quad C \leftarrow$ RegionGrowing(s_i, p)

$\quad G' = G' \setminus C$ // cuts edges leaving C

return $B(s_i, z)$

- probability of cutting an edge is only $\boldsymbol{p}$
- a source either does not reach an edge during Region Growing; then it is not cut
- if it reaches the edge then it either cuts the edge or protects the edge from being cut by other sources
- if we choose $p = \delta$ the probability of cutting an edge is only its LP-value; our expected cost are at most OPT.

Algorithm 1 Multicut(G')

```
while ∃s_i-t_i pair in G' do
    C ← RegionGrowing(s_i, p)
    G' = G' \ C // cuts edges leaving C
return B(s_i, z)
```

- probability of cutting an edge is only $\boldsymbol{p}$
- a source either does not reach an edge during Region Growing; then it is not cut
- if it reaches the edge then it either cuts the edge or protects the edge from being cut by other sources
- if we choose $p = \delta$ the probability of cutting an edge is only its LP-value; our expected cost are at most OPT.

Algorithm 1 Multicut(G')

while $\exists s_i$-t_i pair in G' **do**

  $C \leftarrow$ RegionGrowing(s_i, p)

  $G' = G' \setminus C$ // cuts edges leaving C

return $B(s_i, z)$

- probability of cutting an edge is only $\boldsymbol{p}$
- a source either does not reach an edge during Region Growing; then it is not cut
- if it reaches the edge then it either cuts the edge or protects the edge from being cut by other sources
- if we choose $p = \delta$ the probability of cutting an edge is only its LP-value; our expected cost are at most OPT.

Problem:

We may not cut all source-target pairs.

A component that we remove may contain an s_i-t_i pair.

If we ensure that we cut before reaching radius $1/2$ we are in good shape.

Problem:
We may not cut all source-target pairs.

A component that we remove may contain an s_i-t_i pair.

If we ensure that we cut before reaching radius $1/2$ we are in good shape.

Problem:

We may not cut all source-target pairs.

A component that we remove may contain an s_i-t_i pair.

If we ensure that we cut before reaching radius $1/2$ we are in good shape.

- choose $p = 6 \ln k \cdot \delta$
- we make $\frac{1}{2\delta}$ trials before reaching radius $1/2$.
- we say a Region Growing is not successful if it does not terminate before reaching radius $1/2$.

$$\Pr[\text{not successful}] \le (1-p)^{\frac{1}{2\delta}} = \left((1-p)^{1/p}\right)^{\frac{p}{2\delta}} \le e^{-\frac{p}{2\delta}} \le \frac{1}{k^3}$$

- Hence,

$$\Pr[\exists i \text{ that is not successful}] \le \frac{1}{k^2}$$

- choose $p = 6 \ln k \cdot \delta$
- we make $\frac{1}{2\delta}$ trials before reaching radius $1/2$.
- we say a Region Growing is not successful if it does not terminate before reaching radius $1/2$.

$$\Pr[\text{not successful}] \le (1-p)^{\frac{1}{2\delta}} = \left((1-p)^{1/p}\right)^{\frac{p}{2\delta}} \le e^{-\frac{p}{2\delta}} \le \frac{1}{k^3}$$

- Hence,

$$\Pr[\exists i \text{ that is not successful}] \le \frac{1}{k^2}$$

- choose $p = 6 \ln k \cdot \delta$
- we make $\frac{1}{2\delta}$ trials before reaching radius $1/2$.
- we say a Region Growing is not successful if it does not terminate before reaching radius $1/2$.

$$\Pr[\text{not successful}] \le (1-p)^{\frac{1}{2\delta}} = \left((1-p)^{1/p}\right)^{\frac{p}{2\delta}} \le e^{-\frac{p}{2\delta}} \le \frac{1}{k^3}$$

- Hence,

$$\Pr[\exists i \text{ that is not successful}] \le \frac{1}{k^2}$$

- choose $p = 6\ln k \cdot \delta$
- we make $\frac{1}{2\delta}$ trials before reaching radius $1/2$.
- we say a Region Growing is not successful if it does not terminate before reaching radius $1/2$.

$$\Pr[\text{not successful}] \le (1-p)^{\frac{1}{2\delta}} = \left((1-p)^{1/p}\right)^{\frac{p}{2\delta}} \le e^{-\frac{p}{2\delta}} \le \frac{1}{k^3}$$

- Hence,

$$\Pr[\exists i \text{ that is not successful}] \le \frac{1}{k^2}$$

What is expected cost?

$$
\begin{aligned}
\mathrm{E}[\text{cutsize}] = \Pr[\text{success}] \cdot \mathrm{E}[\text{cutsize} \mid \text{success}] \\
+ \Pr[\text{no success}] \cdot \mathrm{E}[\text{cutsize} \mid \text{no success}]
\end{aligned}
$$

Note: success means all source-target pairs separated

We assume $k \geq 2$.

What is expected cost?

$$\begin{aligned}\mathrm{E}[\text{cutsize}] &= \Pr[\text{success}] \cdot \mathrm{E}[\text{cutsize} \mid \text{success}] \\ &\quad + \Pr[\text{no success}] \cdot \mathrm{E}[\text{cutsize} \mid \text{no success}]\end{aligned}$$

$$\begin{aligned}\mathrm{E}[\text{cutsize} \mid \text{succ.}] &= \frac{\mathrm{E}[\text{cutsize}] - \Pr[\text{no succ.}] \cdot \mathrm{E}[\text{cutsize} \mid \text{no succ.}]}{\Pr[\text{success}]} \\ &\le \frac{\mathrm{E}[\text{cutsize}]}{\Pr[\text{success}]} \le \frac{1}{1-\frac{1}{k^2}} 6 \ln k \cdot \mathrm{OPT} \le 8 \ln k \cdot \mathrm{OPT}\end{aligned}$$

Note: success means all source-target pairs separated

We assume $k \ge 2$.

What is expected cost?

$$\begin{aligned}\mathrm{E}[\text{cutsize}] &= \Pr[\text{success}] \cdot \mathrm{E}[\text{cutsize} \mid \text{success}] \\ &\quad + \Pr[\text{no success}] \cdot \mathrm{E}[\text{cutsize} \mid \text{no success}]\end{aligned}$$

$$\begin{aligned}\mathrm{E}[\text{cutsize} \mid \text{succ.}] &= \frac{\mathrm{E}[\text{cutsize}] - \Pr[\text{no succ.}] \cdot \mathrm{E}[\text{cutsize} \mid \text{no succ.}]}{\Pr[\text{success}]} \\ &\le \frac{\mathrm{E}[\text{cutsize}]}{\Pr[\text{success}]} \le \frac{1}{1-\frac{1}{k^2}} 6 \ln k \cdot \mathrm{OPT} \le 8 \ln k \cdot \mathrm{OPT}\end{aligned}$$

Note: success means all source-target pairs separated

We assume $k \ge 2$.

What is expected cost?

$$\begin{aligned} \mathrm{E}[\text{cutsize}] = {} & \Pr[\text{success}] \cdot \mathrm{E}[\text{cutsize} \mid \text{success}] \\ & + \Pr[\text{no success}] \cdot \mathrm{E}[\text{cutsize} \mid \text{no success}] \end{aligned}$$

$$\begin{aligned} \mathrm{E}[\text{cutsize} \mid \text{succ.}] &= \frac{\mathrm{E}[\text{cutsize}] - \Pr[\text{no succ.}] \cdot \mathrm{E}[\text{cutsize} \mid \text{no succ.}]}{\Pr[\text{success}]} \\ &\le \frac{\mathrm{E}[\text{cutsize}]}{\Pr[\text{success}]} \le \frac{1}{1-\frac{1}{k^2}} 6 \ln k \cdot \mathrm{OPT} \le 8 \ln k \cdot \mathrm{OPT} \end{aligned}$$

Note: success means all source-target pairs separated

We assume $k \ge 2$.

What is expected cost?

$$
\begin{aligned}
\mathrm{E}[\text{cutsize}] &= \Pr[\text{success}] \cdot \mathrm{E}[\text{cutsize} \mid \text{success}] \\
&\quad + \Pr[\text{no success}] \cdot \mathrm{E}[\text{cutsize} \mid \text{no success}]
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{E}[\text{cutsize} \mid \text{succ.}] &= \frac{\mathrm{E}[\text{cutsize}] - \Pr[\text{no succ.}] \cdot \mathrm{E}[\text{cutsize} \mid \text{no succ.}]}{\Pr[\text{success}]} \\
&\le \frac{\mathrm{E}[\text{cutsize}]}{\Pr[\text{success}]} \le \frac{1}{1 - \frac{1}{k^2}} 6 \ln k \cdot \mathrm{OPT} \le 8 \ln k \cdot \mathrm{OPT}
\end{aligned}
$$

Note: success means all source-target pairs separated

We assume $k \geq 2$.

What is expected cost?

$$\begin{aligned}\mathrm{E}[\text{cutsize}] &= \Pr[\text{success}] \cdot \mathrm{E}[\text{cutsize} \mid \text{success}] \\ &\quad + \Pr[\text{no success}] \cdot \mathrm{E}[\text{cutsize} \mid \text{no success}]\end{aligned}$$

$$\begin{aligned}\mathrm{E}[\text{cutsize} \mid \text{succ.}] &= \frac{\mathrm{E}[\text{cutsize}] - \Pr[\text{no succ.}] \cdot \mathrm{E}[\text{cutsize} \mid \text{no succ.}]}{\Pr[\text{success}]} \\ &\le \frac{\mathrm{E}[\text{cutsize}]}{\Pr[\text{success}]} \le \frac{1}{1 - \frac{1}{k^2}} 6 \ln k \cdot \mathrm{OPT} \le 8 \ln k \cdot \mathrm{OPT}\end{aligned}$$

Note: success means all source-target pairs separated

We assume $k \ge 2$.

What is expected cost?

$$\begin{aligned}\text{E}[\text{cutsize}] = \Pr[\text{success}] \cdot \text{E}[\text{cutsize} \mid \text{success}] \\ + \Pr[\text{no success}] \cdot \text{E}[\text{cutsize} \mid \text{no success}]\end{aligned}$$

$$\begin{aligned}\text{E}[\text{cutsize} \mid \text{succ.}] &= \frac{\text{E}[\text{cutsize}] - \Pr[\text{no succ.}] \cdot \text{E}[\text{cutsize} \mid \text{no succ.}]}{\Pr[\text{success}]} \\ &\le \frac{\text{E}[\text{cutsize}]}{\Pr[\text{success}]} \le \frac{1}{1-\frac{1}{k^2}} 6 \ln k \cdot \text{OPT} \le 8 \ln k \cdot \text{OPT}\end{aligned}$$

Note: success means all source-target pairs separated

We assume $k \ge 2$.

If we are not successful we simply perform a trivial k-approximation.

This only increases the expected cost by at most $\frac{1}{k^2} \cdot k\mathrm{OPT} \leq \mathrm{OPT}/k$.

Hence, our final cost is $\mathcal{O}(\ln k) \cdot \mathrm{OPT}$ in expectation.

Definition 87 (NP)

A language $L \in \mathrm{NP}$ if there exists a polynomial time, deterministic verifier V (a Turing machine), s.t.

[$x \in L$] There exists a proof string y, $|y| = \mathrm{poly}(|x|)$, s.t. $V(x, y) =$ “accept”.

[$x \notin L$] For any proof string y, $V(x, y) =$ “reject”.

Note that requiring $|y| = \mathrm{poly}(|x|)$ for $x \notin L$ does not make a difference (**why?**).

Definition 87 (NP)

A language $L \in \mathrm{NP}$ if there exists a polynomial time, deterministic verifier V (a Turing machine), s.t.

[$x \in L$] There exists a proof string y, $|y| = \mathrm{poly}(|x|)$, s.t. $V(x, y) =$ "accept".

[$x \notin L$] For any proof string y, $V(x, y) =$ "reject".

Note that requiring $|y| = \mathrm{poly}(|x|)$ for $x \notin L$ does not make a difference (**why?**).

Probabilistic Proof Verification

Definition 88 (IP)

In an interactive proof system a randomized polynomial-time verifier V (with private coin tosses) interacts with an all powerful prover P in polynomially many rounds. $L \in \mathrm{IP}$ if

[$x \in L$] There exists a strategy for P s.t. V accepts with probability 1.

[$x \notin L$] Regardless of P's strategy V accepts with probability at most $1/2$.

Probabilistic Checkable Proofs

Definition 89 (PCP)

A language $L \in \mathrm{PCP}_{c(n),s(n)}(r(n), q(n))$ if there exists a polynomial time, non-adaptive, randomized verifier V (an Oracle Turing Machine), s.t.

[$x \in L$] There exists a proof string y, s.t. $V^{\pi_y}(x) =$ "accept" with proability $\geq c(n)$.

[$x \notin L$] For any proof string y, $V^{\pi_y}(x) =$ "accept" with probability $\leq s(n)$.

The verifier uses at most $r(n)$ random bits and makes at most $q(n)$ oracle queries.

Probabilistic Checkable Proofs

An Oracle Turing Machine M is a Turing machine that has access to an oracle.

Such an oracle allows M to solve some problem in a single step.

For example having access to a TSP-oracle π_{TSP} would allow M to write a TSP-instance x on a special oracle tape and obtain the answer (yes or no) in a single step.

For such TMs one looks in addition to running time also at query complexity, i.e., how often the machine queries the oracle.

For a proof string y, π_y is an oracle that upon given an index i returns the i-th character y_i of y.

$c(n)$ is called the completeness. If not specified otw. $c(n) = 1$. Probability of accepting a correct proof.

$s(n) < c(n)$ is called the soundness. If not specified otw. $s(n) = 1/2$. Probability of accepting a wrong proof.

$r(n)$ is called the randomness complexity, i.e., how many random bits the (randomized) verifier uses.

$q(n)$ is the query complexity of the verifier.

$$\mathrm{IP} \subseteq \mathrm{PCP}_{1,1/2}(\mathrm{poly}(n), \mathrm{poly}(n))$$

We can view non-adadpative $\mathrm{PCP}_{1,1/2}(\mathrm{poly}(n), \mathrm{poly}(n))$ as the version of IP in which the prover has written down his answers to all possible queries (beforehand).

This makes it harder for the prover to cheat.

The non-cheating prover does not loose power.

Note that the above is not a proof!

- $\mathrm{PCP}(0,0) = \mathrm{P}$
- $\mathrm{PCP}(\mathcal{O}(\log n), 0) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\log n)) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\mathrm{poly}(n)), 0) = \mathrm{coRP}$
  randomized polynomial time with one sided error (positive probability of accepting a false statement)
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(1)) = \mathrm{NP}$ (the PCP theorem)

- $\mathrm{PCP}(0,0) = \mathrm{P}$
- $\mathrm{PCP}(\mathcal{O}(\log n), 0) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\log n)) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\mathrm{poly}(n)), 0) = \mathrm{coRP}$
  randomized polynomial time with one sided error (positive probability of accepting a false statement)
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(1)) = \mathrm{NP}$ (the PCP theorem)

- $\mathrm{PCP}(0,0) = \mathrm{P}$
- $\mathrm{PCP}(\mathcal{O}(\log n), 0) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\log n)) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\mathrm{poly}(n)), 0) = \mathrm{coRP}$
  randomized polynomial time with one sided error (positive probability of accepting a false statement)
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(1)) = \mathrm{NP}$ (the PCP theorem)

- $\mathrm{PCP}(0,0) = \mathrm{P}$
- $\mathrm{PCP}(\mathcal{O}(\log n), 0) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\log n)) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\mathrm{poly}(n)), 0) = \mathrm{coRP}$
  randomized polynomial time with one sided error (positive probability of accepting a false statement)
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(1)) = \mathrm{NP}$ (the PCP theorem)

- $\mathrm{PCP}(0,0) = \mathrm{P}$
- $\mathrm{PCP}(\mathcal{O}(\log n), 0) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\log n)) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\mathrm{poly}(n)), 0) = \mathrm{coRP}$
  randomized polynomial time with one sided error (positive probability of accepting a false statement)
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(1)) = \mathrm{NP}$ (the PCP theorem)

- $\text{PCP}(0, 0) = \text{P}$
- $\text{PCP}(\mathcal{O}(\log n), 0) = \text{P}$
- $\text{PCP}(0, \mathcal{O}(\log n)) = \text{P}$
- $\text{PCP}(0, \mathcal{O}(\text{poly}(n))) = \text{NP}$
- $\text{PCP}(\mathcal{O}(\log n), \mathcal{O}(\text{poly}(n))) = \text{NP}$
- $\text{PCP}(\mathcal{O}(\text{poly}(n)), 0) = \text{coRP}$
  randomized polynomial time with one sided error (positive probability of accepting a false statement)
- $\text{PCP}(\mathcal{O}(\log n), \mathcal{O}(1)) = \text{NP}$ (the PCP theorem)

- $\mathrm{PCP}(0,0) = \mathrm{P}$
- $\mathrm{PCP}(\mathcal{O}(\log n),0) = \mathrm{P}$
- $\mathrm{PCP}(0,\mathcal{O}(\log n)) = \mathrm{P}$
- $\mathrm{PCP}(0,\mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\log n),\mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\mathrm{poly}(n)),0) = \mathrm{coRP}$
  randomized polynomial time with one sided error (positive probability of accepting a false statement)
- $\mathrm{PCP}(\mathcal{O}(\log n),\mathcal{O}(1)) = \mathrm{NP}$ (the PCP theorem)

$\mathrm{NP} \subseteq \mathrm{PCP}(\mathrm{poly}(n), 1)$

$\mathrm{PCP}(\mathrm{poly}(n), 1)$ means that we have a potentially exponentially long proof but we only read a constant number of bits from the proof.

The idea is to encode an NP-witness/proof (e.g. a satisfying assignment (say n bits)) by a code whose code-words have 2^n bits.

A wrong proof is either

- a code-word whose pre-image does not correspond to a satisfying assignment
- or, a sequence of bits that does not correspond to a code-word

We can detect both cases by querying a few positions.

$\text{NP} \subseteq \text{PCP}(\text{poly}(n), 1)$

$\text{PCP}(\text{poly}(n), 1)$ means that we have a potentially exponentially long proof but we only read a constant number of bits from the proof.

The idea is to encode an NP-witness/proof (e.g. a satisfying assignment (say n bits)) by a code whose code-words have 2^n bits.

A wrong proof is either

- a code-word whose pre-image does not correspond to a satisfying assignment
- or, a sequence of bits that does not correspond to a code-word

We can detect both cases by querying a few positions.

$\text{NP} \subseteq \text{PCP}(\text{poly}(n), 1)$

$\text{PCP}(\text{poly}(n), 1)$ means that we have a potentially exponentially long proof but we only read a constant number of bits from the proof.

The idea is to encode an NP-witness/proof (e.g. a satisfying assignment (say n bits)) by a code whose code-words have 2^n bits.

A wrong proof is either

- a code-word whose pre-image does not correspond to a satisfying assignment
- or, a sequence of bits that does not correspond to a code-word

We can detect both cases by querying a few positions.

The Code

$u \in \{0,1\}^n$ (satisfying assignment)

Walsh-Hadamard Code:
$\mathrm{WH}_u : \{0,1\}^n \to \{0,1\}, x \mapsto x^T u$ (over $\mathrm{GF}(2)$)

The code-word for u is WH_u. We identify this function by a bit-vector of length 2^n.

The Code

Lemma 90
If $u \neq u'$ then WH_u and $\mathrm{WH}_{u'}$ differ in at least 2^{n-1} bits.

Suppose that $u - u' \neq 0$. Then

$$\mathrm{WH}_u(x) \neq \mathrm{WH}_{u'}(x) \iff (u - u')^T x \neq 0$$

This holds for 2^{n-1} different vectors x.

The Code

Lemma 90
If $u \neq u'$ then WH_u and $\mathrm{WH}_{u'}$ differ in at least 2^{n-1} bits.

Suppose that $u - u' \neq 0$. Then

$$\mathrm{WH}_u(x) \neq \mathrm{WH}_{u'}(x) \iff (u - u')^T x \neq 0$$

This holds for 2^{n-1} different vectors x.

The Code

Suppose we are given access to a function $f : \{0,1\}^n \rightarrow \{0,1\}$ and want to check whether it is a codeword.

Since the set of codewords is the set of all linear functions $\{0,1\}^n$ to $\{0,1\}$ we can check

$$f(x+y) = f(x) + f(y)$$

for all 2^{2n} pairs x, y. But that's not very efficient.

The Code

Suppose we are given access to a function $f : \{0,1\}^n \to \{0,1\}$ and want to check whether it is a codeword.

Since the set of codewords is the set of all linear functions $\{0,1\}^n$ to $\{0,1\}$ we can check

$$f(x+y) = f(x) + f(y)$$

for all 2^{2n} pairs x, y. But that's not very efficient.

Can we just check a constant number of positions?

Definition 91

Let $\rho \in [0,1]$. We say that $f, g : \{0,1\}^n \to \{0,1\}$ are ρ-close if

$$\Pr_{x \in \{0,1\}^n}[f(x) = g(x)] \geq \rho \enspace .$$

Theorem 92

Let $f : \{0,1\}^n \to \{0,1\}$ with

$$\Pr_{x,y \in \{0,1\}^n}\Big[f(x) + f(y) = f(x+y)\Big] \geq \rho > \frac{1}{2} \enspace .$$

Then there is a linear function $\tilde{f}$ such that f and $\tilde{f}$ are ρ-close.

Definition 91

Let $\rho \in [0,1]$. We say that $f, g : \{0,1\}^n \to \{0,1\}$ are ρ-close if

$$\Pr_{x \in \{0,1\}^n}[f(x) = g(x)] \geq \rho \enspace .$$

Theorem 92

Let $f : \{0,1\}^n \to \{0,1\}$ *with*

$$\Pr_{x,y \in \{0,1\}^n}\Big[f(x) + f(y) = f(x+y)\Big] \geq \rho > \frac{1}{2} \enspace .$$

Then there is a linear function $\tilde{f}$ *such that* f *and* $\tilde{f}$ *are* ρ*-close.*

We need $\mathcal{O}(1/\delta)$ trials to be sure that f is $(1-\delta)$-close to a linear function with (arbitrary) constant probability.

Suppose for $\delta < 1/4$ f is $(1-\delta)$-close to some linear function $\tilde{f}$.

$\tilde{f}$ is uniquely defined by f, since linear functions differ on at least half their inputs.

Suppose we are given $x \in \{0,1\}^n$ and access to f. Can we compute $\tilde{f}(x)$ using only constant number of queries?

Suppose for $\delta < 1/4$ f is $(1-\delta)$-close to some linear function $\tilde{f}$.

$\tilde{f}$ is uniquely defined by f, since linear functions differ on at least half their inputs.

Suppose we are given $x \in \{0,1\}^n$ and access to f. Can we compute $\tilde{f}(x)$ using only constant number of queries?

Suppose for $\delta < 1/4$ f is $(1-\delta)$-close to some linear function $\tilde{f}$.

$\tilde{f}$ is uniquely defined by f, since linear functions differ on at least half their inputs.

Suppose we are given $x \in \{0,1\}^n$ and access to f. Can we compute $\tilde{f}(x)$ using only constant number of queries?

Suppose we are given $x \in \{0,1\}^n$ and access to f. Can we compute $\tilde{f}(x)$ using only constant number of queries?

1. Choose $x' \in \{0,1\}^n$ u.a.r.
2. Set $x'' := x + x'$.
3. Let $y' = f(x')$ and $y'' = f(x'')$.
4. Output $y' + y''$.

x' and x'' are uniformly distributed (albeit dependent). With probability at least $1 - 2\delta$ we have $f(x') = \tilde{f}(x')$ and $f(x'') = \tilde{f}(x'')$.

Then we can compute $\tilde{f}(x)$.

This technique is known as local decoding of the Walsh-Hadamard code.

Suppose we are given $x \in \{0,1\}^n$ and access to f. Can we compute $\tilde{f}(x)$ using only constant number of queries?

1. Choose $x' \in \{0,1\}^n$ u.a.r.
2. Set $x'' := x + x'$.
3. Let $y' = f(x')$ and $y'' = f(x'')$.
4. Output $y' + y''$.

x' and x'' are uniformly distributed (albeit dependent). With probability at least $1 - 2\delta$ we have $f(x') = \tilde{f}(x')$ and $f(x'') = \tilde{f}(x'')$.

Then we can compute $\tilde{f}(x)$.

This technique is known as local decoding of the Walsh-Hadamard code.

Suppose we are given $x \in \{0,1\}^n$ and access to f. Can we compute $\tilde{f}(x)$ using only constant number of queries?

1. Choose $x' \in \{0,1\}^n$ u.a.r.
2. Set $x'' := x + x'$.
3. Let $y' = f(x')$ and $y'' = f(x'')$.
4. Output $y' + y''$.

x' and x'' are uniformly distributed (albeit dependent). With probability at least $1 - 2\delta$ we have $f(x') = \tilde{f}(x')$ and $f(x'') = \tilde{f}(x'')$.

Then we can compute $\tilde{f}(x)$.

This technique is known as local decoding of the Walsh-Hadamard code.

$\mathrm{NP} \subseteq \mathrm{PCP}(\mathrm{poly}(n), 1)$

We show that $\mathrm{QUADEQ} \in \mathrm{PCP}(\mathrm{poly}(n), 1)$. The theorem follows since any PCP-class is closed under polynomial time reductions.

introduce QUADEQ...

prove NP-completeness...

Let A, b be an instance of QUADEQ. Let u be a satisfying assignment.

The correct PCP-proof will be the Walsh-Hadamard encodings of u and $u \otimes u$. The verifier will accept such a proof with probability 1.

We have to make sure that we reject proofs that do not correspond to codewords for vectors of the form u, and $u \otimes u$.

We also have to reject proofs that correspond to codewords for vectors of the form z, and $z \otimes z$, where z is not a satisfying assignment.

Step 1. Linearity Test.

The proof contains $2^n + 2^{n^2}$ bits. This is interpreted as a pair of functions $f : \{0,1\}^n \rightarrow \{0,1\}$ and $g : \{0,1\}^{n^2} \rightarrow \{0,1\}$.

We do a 0.99-linearity test for both functions (requires a constant number of queries).

We also assume that the remaining constant number of (random) accesses only hit points where $f(x) = \tilde{f}(x)$.

Hence, our proof will only see $\tilde{f}$ and therefore we use f for $\tilde{f}$, in the following (similar for g, $\tilde{g}$).

Step 1. Linearity Test.
The proof contains $2^n + 2^{n^2}$ bits. This is interpreted as a pair of functions $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^{n^2} \to \{0,1\}$.

We do a 0.99-linearity test for both functions (requires a constant number of queries).

We also assume that the remaining constant number of (random) accesses only hit points where $f(x) = \tilde{f}(x)$.

Hence, our proof will only see $\tilde{f}$ and therefore we use f for $\tilde{f}$, in the following (similar for g, $\tilde{g}$).

Step 1. Linearity Test.
The proof contains $2^n + 2^{n^2}$ bits. This is interpreted as a pair of functions $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^{n^2} \to \{0,1\}$.

We do a 0.99-linearity test for both functions (requires a constant number of queries).

We also assume that the remaining constant number of (random) accesses only hit points where $f(x) = \tilde{f}(x)$.

Hence, our proof will only see $\tilde{f}$ and therefore we use f for $\tilde{f}$, in the following (similar for g, $\tilde{g}$).

Step 1. Linearity Test.

The proof contains $2^n + 2^{n^2}$ bits. This is interpreted as a pair of functions $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^{n^2} \to \{0,1\}$.

We do a 0.99-linearity test for both functions (requires a constant number of queries).

We also assume that the remaining constant number of (random) accesses only hit points where $f(x) = \tilde{f}(x)$.

Hence, our proof will only see $\tilde{f}$ and therefore we use f for $\tilde{f}$, in the following (similar for g, $\tilde{g}$).

Step 2. Verify that g encodes $u \otimes u$ where u is string encoded by f.

$f(r) = u^T r$ and $g(z) = w^T z$ since f, g are linear.

- choose r, r' independently, u.a.r. from $\{0,1\}^n$
- if $f(r)f(r') \neq g(r \otimes r')$ reject
- repeat 3 times

A correct proof survives the test

$$f(r) \cdot f(r')$$

A correct proof survives the test

$$f(r) \cdot f(r') = u^T r \cdot u^T r'$$

A correct proof survives the test

$$f(r) \cdot f(r') = u^T r \cdot u^T r' = \Big(\sum_i u_i r_i\Big) \cdot \Big(\sum_j u_j r'_j\Big)$$

A correct proof survives the test

$$f(r) \cdot f(r') = u^T r \cdot u^T r' = \Big(\sum_i u_i r_i \Big) \cdot \Big(\sum_j u_j r'_j \Big)$$

$$= \sum_{ij} u_i u_j r_i r'_j$$

A correct proof survives the test

$$f(r) \cdot f(r') = u^T r \cdot u^T r' = \Big(\sum_i u_i r_i\Big) \cdot \Big(\sum_j u_j r'_j\Big)$$

$$= \sum_{ij} u_i u_j r_i r'_j = (u \otimes u)^T (r \otimes r')$$

A correct proof survives the test

$$f(r) \cdot f(r') = u^T r \cdot u^T r' = \Big(\sum_i u_i r_i\Big) \cdot \Big(\sum_j u_j r'_j\Big)$$
$$= \sum_{ij} u_i u_j r_i r'_j = (u \otimes u)^T (r \otimes r') = g(r \otimes r')$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let W be $n \times n$-matrix with entries from w. Let U be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let W be $n \times n$-matrix with entries from w. Let U be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r')$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let W be $n \times n$-matrix with entries from w. Let U be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r') = w^T (r \otimes r')$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let W be $n \times n$-matrix with entries from w. Let U be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r') = w^T (r \otimes r') = \sum_{ij} w_{ij} r_i r'_j$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let W be $n \times n$-matrix with entries from w. Let U be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r') = w^T (r \otimes r') = \sum_{ij} w_{ij} r_i r'_j = r^T W r'$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let W be $n \times n$-matrix with entries from w. Let U be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r') = w^T(r \otimes r') = \sum_{ij} w_{ij} r_i r'_j = r^T W r'$$

$$f(r) f(r')$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let W be $n \times n$-matrix with entries from w. Let U be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r') = w^T(r \otimes r') = \sum_{ij} w_{ij} r_i r'_j = r^T W r'$$

$$f(r) f(r') = u^T r \cdot u^T r'$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let W be $n \times n$-matrix with entries from w. Let U be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r') = w^T(r \otimes r') = \sum_{ij} w_{ij} r_i r'_j = r^T W r'$$

$$f(r) f(r') = u^T r \cdot u^T r' = r^T U r'$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let W be $n \times n$-matrix with entries from w. Let U be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r') = w^T(r \otimes r') = \sum_{ij} w_{ij} r_i r'_j = r^T W r'$$

$$f(r)f(r') = u^T r \cdot u^T r' = r^T U r'$$

If $U \neq W$ then $Wr' \neq Ur'$ with probability at least 1/2. Then $r^T W r' \neq r^T U r'$ with probability at least 1/4.

Step 3. Verify that f encodes satisfying assignment.

We need to check

$$A_k(u \otimes u) = b_k$$

where A_k is the k-th row of the constraint matrix. But the left hand side is just $g(A_k^T)$.

We can handle this by a single query but checking all constraints would take $\mathcal{O}(m)$ steps.

We compute rA, where $r \in_R \{0,1\}^m$. If u is not a satisfying assignment then with probability 1/2 the vector r will hit an odd number of violated constraint.

In this case $rA(u \otimes u) \neq rb_k$. The left hand side is equal to $g(A^T r^T)$.

Step 3. Verify that f encodes satisfying assignment.

We need to check

$$A_k(u \otimes u) = b_k$$

where A_k is the k-th row of the constraint matrix. But the left hand side is just $g(A_k^T)$.

We can handle this by a single query but checking all constraints would take $\mathcal{O}(m)$ steps.

We compute rA, where $r \in_R \{0,1\}^m$. If u is not a satisfying assignment then with probability 1/2 the vector r will hit an odd number of violated constraint.

In this case $rA(u \otimes u) \neq rb_k$. The left hand side is equal to $g(A^T r^T)$.

Step 3. Verify that f encodes satisfying assignment.

We need to check

$$A_k(u \otimes u) = b_k$$

where A_k is the k-th row of the constraint matrix. But the left hand side is just $g(A_k^T)$.

We can handle this by a single query but checking all constraints would take $\mathcal{O}(m)$ steps.

We compute rA, where $r \in_R \{0,1\}^m$. If u is not a satisfying assignment then with probability 1/2 the vector r will hit an odd number of violated constraint.

In this case $rA(u \otimes u) \neq rb_k$. The left hand side is equal to $g(A^T r^T)$.

Step 3. Verify that f encodes satisfying assignment.

We need to check

$$A_k(u \otimes u) = b_k$$

where A_k is the k-th row of the constraint matrix. But the left hand side is just $g(A_k^T)$.

We can handle this by a single query but checking all constraints would take $\mathcal{O}(m)$ steps.

We compute rA, where $r \in_R \{0,1\}^m$. If u is not a satisfying assignment then with probability 1/2 the vector r will hit an odd number of violated constraint.

In this case $rA(u \otimes u) \neq rb_k$. The left hand side is equal to $g(A^T r^T)$.

Theorem 92

Let $f : \{0,1\}^n \to \{0,1\}$ *with*

$$\Pr_{x,y\in\{0,1\}^n}\Big[f(x) + f(y) = f(x+y)\Big] \geq \rho > \frac{1}{2}\ .$$

Then there is a linear function $\tilde{f}$ *such that* f *and* $\tilde{f}$ *are* ρ*-close.*

Fourier Transform over $\mathrm{GF}(2)$

In the following we use $\{-1, 1\}$ instead of $\{0, 1\}$. We map $b \in \{0, 1\}$ to $(-1)^b$.

This turns summation into multiplication.

The set of function $f : \{-1, 1\} \to \mathbb{R}$ form a 2^n-dimensional Hilbert space.

Hilbert space

- addition $(f+g)(x) = f(x) + g(x)$
- scalar multiplication $(\alpha f)(x) = \alpha f(x)$
- inner product $\langle f, g\rangle = E_{x\in\{0,1\}^n}[f(x)g(x)]$
  (bilinear, $\langle f, f\rangle \geq 0$, and $\langle f, f\rangle = 0 \Rightarrow f = 0$)
- completeness: any sequence x_k of vectors for which

$$\sum_{k=1}^{\infty} \|x_k\| < \infty \text{ fulfills } \left\| L - \sum_{k=1}^{N} x_k \right\| \to 0$$

for some vector L.

standard basis

$$e_x(y) = \begin{cases} 1 & x = y \\ 0 & \text{otw.} \end{cases}$$

Then, $f(x) = \sum_x \alpha_x e_x$ where $\alpha_x = f(x)$, this means the functions e_x form a basis. This basis is orthonormal.

fourier basis

For $\alpha \subseteq [n]$ define

$$\chi_\alpha(x) = \prod_{i \in \alpha} x_i$$

fourier basis

For $\alpha \subseteq [n]$ define

$$\chi_\alpha(x) = \prod_{i \in \alpha} x_i$$

Note that

$$\langle \chi_\alpha, \chi_\beta \rangle$$

fourier basis

For $\alpha \subseteq [n]$ define

$$\chi_\alpha(x) = \prod_{i \in \alpha} x_i$$

Note that

$$\langle \chi_\alpha, \chi_\beta \rangle = E_x\Big[\chi_\alpha(x)\chi_\beta(x)\Big]$$

fourier basis

For $\alpha \subseteq [n]$ define

$$\chi_\alpha(x) = \prod_{i\in\alpha} x_i$$

Note that

$$\langle \chi_\alpha, \chi_\beta \rangle = E_x\Big[\chi_\alpha(x)\chi_\beta(x)\Big] = E_x\Big[\chi_{\alpha\triangle\beta}(x)\Big]$$

fourier basis

For $\alpha \subseteq [n]$ define

$$\chi_\alpha(x) = \prod_{i \in \alpha} x_i$$

Note that

$$\langle \chi_\alpha, \chi_\beta \rangle = E_x\Big[\chi_\alpha(x)\chi_\beta(x)\Big] = E_x\Big[\chi_{\alpha \triangle \beta}(x)\Big] = \begin{cases} 1 & \alpha = \beta \\ 0 & \text{otw.} \end{cases}$$

fourier basis

For $\alpha \subseteq [n]$ define

$$\chi_\alpha(x) = \prod_{i \in \alpha} x_i$$

Note that

$$\langle \chi_\alpha, \chi_\beta \rangle = E_x\left[\chi_\alpha(x)\chi_\beta(x)\right] = E_x\left[\chi_{\alpha \triangle \beta}(x)\right] = \begin{cases} 1 & \alpha = \beta \\ 0 & \text{otw.} \end{cases}$$

This means the χ_α's also define an orthonormal basis. (since we have 2^n orthonormal vectors...)

A function χ_α multiplies a set of x_i's. Back in the $\mathrm{GF}(2)$-world this means summing a set of z_i's where $x_i = (-1)^{z_i}$.

This means the function χ_α correspond to linear functions in the $\mathrm{GF}(2)$ world.

We can write any function $f : \{-1,1\}^n \to \mathbb{R}$ as

$$f = \sum_\alpha \hat{f}_\alpha \chi_\alpha$$

We call $\hat{f}_\alpha$ the α^{th} Fourier coefficient.

Lemma 93

1. $\langle f, g\rangle = \sum_\alpha f_\alpha g_\alpha$
2. $\langle f, f\rangle = \sum_\alpha f_\alpha^2$

Note that for Boolean functions $f : \{-1,1\}^n \to \{-1,1\}$, $\langle f, f\rangle = 1$.

Linearity Test

GF(2)

We want to show that if $\Pr_{x,y}[f(x) + f(y) = f(x + y)]$ is large than f has a large agreement with a linear function.

Linearity Test

GF(2)

We want to show that if $\Pr_{x,y}[f(x) + f(y) = f(x + y)]$ is large than f has a large agreement with a linear function.

Hilbert space (we prove)

Suppose that $f : \{+1, -1\}^n \to \{-1, 1\}$ satisfies $\Pr_{x,y}[f(x)f(y) = f(xy)] \geq \frac{1}{2} + \epsilon$. Then there is some $\alpha \subseteq [n]$, s.t. $\hat{f}_\alpha \geq 2\epsilon$.

Linearity Test

GF(2)
We want to show that if $\Pr_{x,y}[f(x) + f(y) = f(x+y)]$ is large than f has a large agreement with a linear function.

Hilbert space (we prove)
Suppose that $f : \{+1,-1\}^n \to \{-1,1\}$ satisfies $\Pr_{x,y}[f(x)f(y) = f(xy)] \geq \frac{1}{2} + \epsilon$. Then there is some $\alpha \subseteq [n]$, s.t. $\hat{f}_\alpha \geq 2\epsilon$.

For Boolean functions $\langle f, g\rangle$ is the fraction of inputs on which f, g agree **minus** the fraction of inputs on which they disagree.

Linearity Test

GF(2)
We want to show that if $\Pr_{x,y}[f(x) + f(y) = f(x+y)]$ is large than f has a large agreement with a linear function.

Hilbert space (we prove)
Suppose that $f : \{+1,-1\}^n \to \{-1,1\}$ satisfies $\Pr_{x,y}[f(x)f(y) = f(xy)] \geq \frac{1}{2} + \epsilon$. Then there is some $\alpha \subseteq [n]$, s.t. $\hat{f}_\alpha \geq 2\epsilon$.

For Boolean functions $\langle f, g \rangle$ is the fraction of inputs on which f, g agree **minus** the fraction of inputs on which they disagree.

$$2\epsilon \leq \hat{f}_\alpha$$

Linearity Test

GF(2)

We want to show that if $\Pr_{x,y}[f(x) + f(y) = f(x+y)]$ is large than f has a large agreement with a linear function.

Hilbert space (we prove)

Suppose that $f : \{+1,-1\}^n \to \{-1,1\}$ satisfies $\Pr_{x,y}[f(x)f(y) = f(xy)] \geq \frac{1}{2} + \epsilon$. Then there is some $\alpha \subseteq [n]$, s.t. $\hat{f}_\alpha \geq 2\epsilon$.

For Boolean functions $\langle f, g\rangle$ is the fraction of inputs on which f, g agree **minus** the fraction of inputs on which they disagree.

$$2\epsilon \leq \hat{f}_\alpha = \langle f, \chi_\alpha\rangle$$

Linearity Test

GF(2)
We want to show that if $\Pr_{x,y}[f(x) + f(y) = f(x+y)]$ is large than f has a large agreement with a linear function.

Hilbert space (we prove)
Suppose that $f : \{+1,-1\}^n \to \{-1,1\}$ satisfies $\Pr_{x,y}[f(x)f(y) = f(xy)] \geq \frac{1}{2} + \epsilon$. Then there is some $\alpha \subseteq [n]$, s.t. $\hat{f}_\alpha \geq 2\epsilon$.

For Boolean functions $\langle f, g \rangle$ is the fraction of inputs on which f, g agree **minus** the fraction of inputs on which they disagree.

$$2\epsilon \leq \hat{f}_\alpha = \langle f, \chi_\alpha \rangle = \text{agree} - \text{disagree}$$

Linearity Test

GF(2)
We want to show that if $\Pr_{x,y}[f(x) + f(y) = f(x+y)]$ is large than f has a large agreement with a linear function.

Hilbert space (we prove)
Suppose that $f : \{+1,-1\}^n \to \{-1,1\}$ satisfies $\Pr_{x,y}[f(x)f(y) = f(xy)] \geq \frac{1}{2} + \epsilon$. Then there is some $\alpha \subseteq [n]$, s.t. $\hat{f}_\alpha \geq 2\epsilon$.

For Boolean functions $\langle f, g \rangle$ is the fraction of inputs on which f, g agree **minus** the fraction of inputs on which they disagree.

$$2\epsilon \leq \hat{f}_\alpha = \langle f, \chi_\alpha \rangle = \text{agree} - \text{disagree} = 2\text{agree} - 1$$

Linearity Test

GF(2)
We want to show that if $\Pr_{x,y}[f(x) + f(y) = f(x+y)]$ is large than f has a large agreement with a linear function.

Hilbert space (we prove)
Suppose that $f : \{+1,-1\}^n \to \{-1,1\}$ satisfies $\Pr_{x,y}[f(x)f(y) = f(xy)] \geq \frac{1}{2} + \epsilon$. Then there is some $\alpha \subseteq [n]$, s.t. $\hat{f}_\alpha \geq 2\epsilon$.

For Boolean functions $\langle f, g\rangle$ is the fraction of inputs on which f, g agree **minus** the fraction of inputs on which they disagree.

$$2\epsilon \leq \hat{f}_\alpha = \langle f, \chi_\alpha\rangle = \text{agree} - \text{disagree} = 2\text{agree} - 1$$

This gives that the agreement between f and χ_α is at least $\frac{1}{2} + \epsilon$.

Linearity Test

$$\Pr_{x,y}[f(xy) = f(x)f(y)] \geq \frac{1}{2} + \epsilon$$

is equivalent to

$$E_{x,y}[f(xy)f(x)f(y)] = \text{agreement} - \text{disagreement} \geq 2\epsilon$$

$$2\epsilon \leq E_{x,y}\left[f(xy)f(x)f(y)\right]$$

$$
\begin{aligned}
2\epsilon &\leq E_{x,y}\Big[f(xy)f(x)f(y) \Big] \\
&= E_{x,y}\Big[\Big(\sum_\alpha \hat{f}_\alpha \chi_\alpha(xy)\Big) \cdot \Big(\sum_\beta \hat{f}_\beta \chi_\beta(x)\Big) \cdot \Big(\sum_\gamma \hat{f}_\gamma \chi_\gamma(y)\Big) \Big]
\end{aligned}
$$

$$
\begin{aligned}
2\epsilon &\le E_{x,y}\Big[f(xy)f(x)f(y)\Big] \\
&= E_{x,y}\Big[\Big(\textstyle\sum_\alpha \hat{f}_\alpha \chi_\alpha(xy)\Big)\cdot\Big(\sum_\beta \hat{f}_\beta \chi_\beta(x)\Big)\cdot\Big(\sum_\gamma \hat{f}_\gamma \chi_\gamma(y)\Big)\Big] \\
&= E_{x,y}\Big[\textstyle\sum_{\alpha,\beta,\gamma} \hat{f}_\alpha \hat{f}_\beta \hat{f}_\gamma \chi_\alpha(x)\chi_\alpha(y)\chi_\beta(x)\chi_\gamma(y)\Big]
\end{aligned}
$$

$$
\begin{aligned}
2\epsilon &\leq E_{x,y}\Big[f(xy)f(x)f(y)\Big] \\
&= E_{x,y}\Big[\Big(\sum_\alpha \hat{f}_\alpha \chi_\alpha(xy)\Big)\cdot\Big(\sum_\beta \hat{f}_\beta \chi_\beta(x)\Big)\cdot\Big(\sum_\gamma \hat{f}_\gamma \chi_\gamma(y)\Big)\Big] \\
&= E_{x,y}\Big[\sum_{\alpha,\beta,\gamma} \hat{f}_\alpha \hat{f}_\beta \hat{f}_\gamma \chi_\alpha(x)\chi_\alpha(y)\chi_\beta(x)\chi_\gamma(y)\Big] \\
&= \sum_{\alpha,\beta,\gamma} \hat{f}_\alpha \hat{f}_\beta \hat{f}_\gamma \cdot E_x\Big[\chi_\alpha(x)\chi_\beta(x)\Big]E_y\Big[\chi_\alpha(y)\chi_\gamma(y)\Big]
\end{aligned}
$$

$$
\begin{aligned}
2\epsilon &\le E_{x,y}\Big[f(xy)f(x)f(y)\Big] \\
&= E_{x,y}\Big[\Big(\sum\nolimits_{\alpha}\hat{f}_{\alpha}\chi_{\alpha}(xy)\Big)\cdot\Big(\sum\nolimits_{\beta}\hat{f}_{\beta}\chi_{\beta}(x)\Big)\cdot\Big(\sum\nolimits_{\gamma}\hat{f}_{\gamma}\chi_{\gamma}(y)\Big)\Big] \\
&= E_{x,y}\Big[\sum\nolimits_{\alpha,\beta,\gamma}\hat{f}_{\alpha}\hat{f}_{\beta}\hat{f}_{\gamma}\chi_{\alpha}(x)\chi_{\alpha}(y)\chi_{\beta}(x)\chi_{\gamma}(y)\Big] \\
&= \sum\nolimits_{\alpha,\beta,\gamma}\hat{f}_{\alpha}\hat{f}_{\beta}\hat{f}_{\gamma}\cdot E_x\Big[\chi_{\alpha}(x)\chi_{\beta}(x)\Big]E_y\Big[\chi_{\alpha}(y)\chi_{\gamma}(y)\Big] \\
&= \sum_{\alpha}\hat{f}_{\alpha}^{3}
\end{aligned}
$$

$$
\begin{aligned}
2\epsilon &\le E_{x,y}\Big[f(xy)f(x)f(y)\Big] \\
&= E_{x,y}\Big[\Big(\sum\nolimits_\alpha \hat{f}_\alpha\chi_\alpha(xy)\Big)\cdot\Big(\sum\nolimits_\beta \hat{f}_\beta\chi_\beta(x)\Big)\cdot\Big(\sum\nolimits_\gamma \hat{f}_\gamma\chi_\gamma(y)\Big)\Big] \\
&= E_{x,y}\Big[\sum\nolimits_{\alpha,\beta,\gamma}\hat{f}_\alpha\hat{f}_\beta\hat{f}_\gamma\chi_\alpha(x)\chi_\alpha(y)\chi_\beta(x)\chi_\gamma(y)\Big] \\
&= \sum\nolimits_{\alpha,\beta,\gamma}\hat{f}_\alpha\hat{f}_\beta\hat{f}_\gamma\cdot E_x\Big[\chi_\alpha(x)\chi_\beta(x)\Big]E_y\Big[\chi_\alpha(y)\chi_\gamma(y)\Big] \\
&= \sum_\alpha \hat{f}_\alpha^3 \\
&\le \max_\alpha \hat{f}_\alpha\cdot\sum_\alpha \hat{f}_\alpha^2 = \max_\alpha \hat{f}_\alpha
\end{aligned}
$$

Probabilistic proof for Graph NonIsomorphism

GNI is the language of pairs of non-isomorphic graphs

Probabilistic proof for Graph NonIsomorphism

GNI is the language of pairs of non-isomorphic graphs

Verifier gets input (G_0, G_1) (two graphs with n-nodes)

Probabilistic proof for Graph NonIsomorphism

GNI is the language of pairs of non-isomorphic graphs

Verifier gets input (G_0, G_1) (two graphs with n-nodes)

It expects a proof of the following form:

- For any labeled n-node graph H the H's bit $P[H]$ of the proof fulfills

$$G_0 \equiv H \implies P[H] = 0$$
$$G_1 \equiv H \implies P[H] = 1$$
$$G_0, G_1 \not\equiv H \implies P[H] = \text{arbitrary}$$

Probabilistic proof for Graph NonIsomorphism

Verifier:

- choose $b \in \{0,1\}$ at random
- take graph G_b and apply a random permutation to obtain a labeled graph H
- check whether $P[H] = b$

Probabilistic proof for Graph NonIsomorphism

Verifier:

- choose $b \in \{0, 1\}$ at random
- take graph G_b and apply a random permutation to obtain a labeled graph H
- check whether $P[H] = b$

If $G_0 \not\equiv G_1$ then by using the obvious proof the verifier will always accept.

Probabilistic proof for Graph NonIsomorphism

Verifier:

- choose $b \in \{0, 1\}$ at random
- take graph G_b and apply a random permutation to obtain a labeled graph H
- check whether $P[H] = b$

If $G_0 \not\equiv G_1$ then by using the obvious proof the verifier will always accept.

If $G_0 \not\equiv G_1$ a proof only accepts with probability $1/2$.

- suppose $\pi(G_0) = G_1$
- if we accept for $b = 1$ and permutation π_{rand} we reject for permutation $b = 0$ and $\pi_{\text{rand}} \circ \pi$

How to show Harndess of Approximation?

Decision version of optimization problems:

Suppose we have some maximization problem.

How to show Harndess of Approximation?

Decision version of optimization problems:
Suppose we have some maximization problem.

The corresponding decision problem equips each instance with a parameter k and asks whether we can obtain a solution value of at least k. (where infeasible solutions are assumed to have value $-\infty$)

How to show Harndess of Approximation?

Decision version of optimization problems:
Suppose we have some maximization problem.

The corresponding decision problem equips each instance with a parameter k and asks whether we can obtain a solution value of at least k. (where infeasible solutions are assumed to have value $-\infty$)

(Analogous for minimization problems.)

How to show Harndess of Approximation?

Decision version of optimization problems:
Suppose we have some maximization problem.

The corresponding decision problem equips each instance with a parameter k and asks whether we can obtain a solution value of at least k. (where infeasible solutions are assumed to have value $-\infty$)

(Analogous for minimization problems.)

This is the standard way to show that some optimization problem is e.g. NP-hard.

How to show Harndess of Approximation?

Gap version of optimization problems:

Suppose we have some maximization problem.

How to show Harndess of Approximation?

Gap version of optimization problems:

Suppose we have some maximization problem.

The corresponding (α, β)-gap problem asks the following:

How to show Harndess of Approximation?

Gap version of optimization problems:
Suppose we have some maximization problem.

The corresponding (α, β)-gap problem asks the following:

Suppose we are given an instance I and a promise that either $\text{opt}(I) \geq \beta$ or $\text{opt}(I) \leq \alpha$. Can we differentiate between these two cases?

How to show Harndess of Approximation?

Gap version of optimization problems:
Suppose we have some maximization problem.

The corresponding (α, β)-gap problem asks the following:

Suppose we are given an instance I and a promise that either $\text{opt}(I) \geq \beta$ or $\text{opt}(I) \leq \alpha$. Can we differentiate between these two cases?

An algorithm A has to output

- $A(I) = 1$ if $\text{opt}(I) \geq \beta$
- $A(I) = 0$ if $\text{opt}(I) \leq \alpha$
- $A(I) =$ arbitrary, otw

How to show Harndess of Approximation?

Gap version of optimization problems:
Suppose we have some maximization problem.

The corresponding (α, β)-gap problem asks the following:

Suppose we are given an instance I and a promise that either $\text{opt}(I) \geq \beta$ or $\text{opt}(I) \leq \alpha$. Can we differentiate between these two cases?

An algorithm A has to output

- $A(I) = 1$ if $\text{opt}(I) \geq \beta$
- $A(I) = 0$ if $\text{opt}(I) \leq \alpha$
- $A(I) =$ arbitrary, otw

Note that this is not a decision problem

An approximation algorithm with approximation guarantee $c \leq \beta/\alpha$ can solve an (α, β)-gap problem.

Constraint Satisfaction Problem

A qCSP ϕ consists of m n-ary Boolean functions $\phi_1, \dots, \phi_m$ (constraints), where each function only depends on q inputs. The goal is to maximize the number of satisifed constraints.

- $u \in \{0,1\}^n$ satsifies constraint ϕ_i if $\phi_i(u) = 1$
- $r(u) := \sum_i \phi_i(u)/m$ is fraction of satisfied constraints
- $\text{value}(\phi) = \max_u r(u)$
- ϕ is satisfiable if $\text{value}(\phi) = 1$.

3SAT is a constraint satsifaction problem with $q = 3$.

Constraint Satisfaction Problem

GAP version:

A ρGAPqCSP ϕ consists of m n-ary Boolean functions $\phi_1, \dots, \phi_m$ (constraints), where each function only depends on q inputs. We know that either ϕ is satisfiable or $\text{value}(\phi) < \rho$, and want to differentiate between these cases.

ρGAPqCSP is NP-hard if for any $L \in \text{NP}$ there is a polytime computable function f mapping strings to instances of qCSP s.t.

- $x \in L \Longrightarrow \text{value}(f(x)) = 1$
- $x \notin L \Longrightarrow \text{value}(f(x)) < \rho$

Theorem 94

There exists constants q, ρ such that ρGAPqCSP is NP-hard.

We know that $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$.

We reduce 3SAT to ρGAPqCSP.

3SAT has a PCP system in which the verifier makes a constant number of queries (q), and uses $c \log n$ random bits (for some c).

For input x and $r \in \{0,1\}^{c \log n}$ define

- $V_{x,r}$ as function that maps a proof π to the result (0/1) computed by the verifier when using proof π, instance x and random coins r.
- $V_{x,r}$ only depends on q bits of the proof

For any x the collection ϕ of the $V_{x,r}$'s over all r is polynomial size qCSP.

ϕ can be computed in polynomial time.

We know that $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$.

We reduce 3SAT to ρGAPqCSP.

3SAT has a PCP system in which the verifier makes a constant number of queries (q), and uses $c \log n$ random bits (for some c).

For input x and $r \in \{0,1\}^{c \log n}$ define

- $V_{x,r}$ as function that maps a proof π to the result (0/1) computed by the verifier when using proof π, instance x and random coins r.
- $V_{x,r}$ only depends on q bits of the proof

For any x the collection ϕ of the $V_{x,r}$'s over all r is polynomial size qCSP.

ϕ can be computed in polynomial time.

We know that $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$.

We reduce 3SAT to ρGAPqCSP.

3SAT has a PCP system in which the verifier makes a constant number of queries (q), and uses $c \log n$ random bits (for some c).

For input x and $r \in \{0,1\}^{c \log n}$ define

- $V_{x,r}$ as function that maps a proof π to the result (0/1) computed by the verifier when using proof π, instance x and random coins r.
- $V_{x,r}$ only depends on q bits of the proof

For any x the collection ϕ of the $V_{x,r}$'s over all r is polynomial size qCSP.

ϕ can be computed in polynomial time.

We know that $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$.

We reduce 3SAT to ρGAPqCSP.

3SAT has a PCP system in which the verifier makes a constant number of queries (q), and uses $c \log n$ random bits (for some c).

For input x and $r \in \{0,1\}^{c \log n}$ define

- $V_{x,r}$ as function that maps a proof π to the result (0/1) computed by the verifier when using proof π, instance x and random coins r.
- $V_{x,r}$ only depends on q bits of the proof

For any x the collection ϕ of the $V_{x,r}$'s over all r is polynomial size qCSP.

ϕ can be computed in polynomial time.

We know that $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$.

We reduce 3SAT to ρGAPqCSP.

3SAT has a PCP system in which the verifier makes a constant number of queries (q), and uses $c \log n$ random bits (for some c).

For input x and $r \in \{0,1\}^{c \log n}$ define

- $V_{x,r}$ as function that maps a proof π to the result (0/1) computed by the verifier when using proof π, instance x and random coins r.
- $V_{x,r}$ only depends on q bits of the proof

For any x the collection ϕ of the $V_{x,r}$'s over all r is polynomial size qCSP.

ϕ can be computed in polynomial time.

We know that $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$.

We reduce 3SAT to ρGAPqCSP.

3SAT has a PCP system in which the verifier makes a constant number of queries (q), and uses $c \log n$ random bits (for some c).

For input x and $r \in \{0,1\}^{c \log n}$ define

- $V_{x,r}$ as function that maps a proof π to the result (0/1) computed by the verifier when using proof π, instance x and random coins r.
- $V_{x,r}$ only depends on q bits of the proof

For any x the collection ϕ of the $V_{x,r}$'s over all r is polynomial size qCSP.

ϕ can be computed in polynomial time.

$$x \in \text{3SAT} \Longrightarrow \phi \text{ is satisfiable}$$

$$x \notin \text{3SAT} \Longrightarrow \text{value}(\phi) \leq \frac{1}{2}$$

This means that ρGAPqCSP is NP-hard.

$$x \in \text{3SAT} \implies \phi \text{ is satisfiable}$$

$$x \notin \text{3SAT} \implies \text{value}(\phi) \leq \frac{1}{2}$$

This means that ρGAPqCSP is NP-hard.

Suppose that ρGAPqCSP is NP-hard for some constants q, ρ ($\rho < 1$).

Suppose you get an input x, and have to decide whether $x \in L$.

We get a verifier as follows.

We use the reduction to map an input x into an instance ϕ of qCSP.

The proof is considered to be an assignment to the variables.

We can check a random constraint ϕ_i by making q queries. If $x \in L$ the verifier accepts with probability 1.

Otw. at most a ρ fraction of constraints are satisfied by the proof, and the verifier accepts with probability at most ρ.

Hence, $L \in \mathrm{PCP}_{1,\rho}(\log_2 m, q)$, where m is the number of constraints.

Suppose that ρGAPqCSP is NP-hard for some constants q, ρ ($\rho < 1$).

Suppose you get an input x, and have to decide whether $x \in L$.

We get a verifier as follows.

We use the reduction to map an input x into an instance ϕ of qCSP.

The proof is considered to be an assignment to the variables.

We can check a random constraint ϕ_i by making q queries. If $x \in L$ the verifier accepts with probability 1.

Otw. at most a ρ fraction of constraints are satisfied by the proof, and the verifier accepts with probability at most ρ.

Hence, $L \in \mathrm{PCP}_{1,\rho}(\log_2 m, q)$, where m is the number of constraints.

Suppose that ρGAPqCSP is NP-hard for some constants q, ρ ($\rho < 1$).

Suppose you get an input x, and have to decide whether $x \in L$.

We get a verifier as follows.

We use the reduction to map an input x into an instance ϕ of qCSP.

The proof is considered to be an assignment to the variables.

We can check a random constraint ϕ_i by making q queries. If $x \in L$ the verifier accepts with probability 1.

Otw. at most a ρ fraction of constraints are satisfied by the proof, and the verifier accepts with probability at most ρ.

Hence, $L \in \mathrm{PCP}_{1,\rho}(\log_2 m, q)$, where m is the number of constraints.

Suppose that ρGAPqCSP is NP-hard for some constants q, ρ ($\rho < 1$).

Suppose you get an input x, and have to decide whether $x \in L$.

We get a verifier as follows.

We use the reduction to map an input x into an instance ϕ of qCSP.

The proof is considered to be an assignment to the variables.

We can check a random constraint ϕ_i by making q queries. If $x \in L$ the verifier accepts with probability 1.

Otw. at most a ρ fraction of constraints are satisfied by the proof, and the verifier accepts with probability at most ρ.

Hence, $L \in \mathrm{PCP}_{1,\rho}(\log_2 m, q)$, where m is the number of constraints.

Suppose that ρGAPqCSP is NP-hard for some constants q, ρ ($\rho < 1$).

Suppose you get an input x, and have to decide whether $x \in L$.

We get a verifier as follows.

We use the reduction to map an input x into an instance ϕ of qCSP.

The proof is considered to be an assignment to the variables.

We can check a random constraint ϕ_i by making q queries. If $x \in L$ the verifier accepts with probability 1.

Otw. at most a ρ fraction of constraints are satisfied by the proof, and the verifier accepts with probability at most ρ.

Hence, $L \in \mathrm{PCP}_{1,\rho}(\log_2 m, q)$, where m is the number of constraints.

Suppose that ρGAPqCSP is NP-hard for some constants q, ρ ($\rho < 1$).

Suppose you get an input x, and have to decide whether $x \in L$.

We get a verifier as follows.

We use the reduction to map an input x into an instance ϕ of qCSP.

The proof is considered to be an assignment to the variables.

We can check a random constraint ϕ_i by making q queries. If $x \in L$ the verifier accepts with probability 1.

Otw. at most a ρ fraction of constraints are satisfied by the proof, and the verifier accepts with probability at most ρ.

Hence, $L \in \mathrm{PCP}_{1,\rho}(\log_2 m, q)$, where m is the number of constraints.

Suppose that ρGAPqCSP is NP-hard for some constants q, ρ ($\rho < 1$).

Suppose you get an input x, and have to decide whether $x \in L$.

We get a verifier as follows.

We use the reduction to map an input x into an instance ϕ of qCSP.

The proof is considered to be an assignment to the variables.

We can check a random constraint ϕ_i by making q queries. If $x \in L$ the verifier accepts with probability 1.

Otw. at most a ρ fraction of constraints are satisfied by the proof, and the verifier accepts with probability at most ρ.

Hence, $L \in \mathrm{PCP}_{1,\rho}(\log_2 m, q)$, where m is the number of constraints.

Theorem 95

For any positive constants $\epsilon, \delta > 0$*, it is the case that* $\mathrm{NP} \subseteq \mathrm{PCP}_{1-\epsilon,1/2+\delta}(\log n, 3)$*, and the verifier is restricted to use only the functions odd and even.*

It is NP-hard to approximate an ODD/EVEN constraint satisfaction problem by a factor better than $1/2 + \delta$, for any constant δ.

Theorem 96

For any positive constant $\delta > 0$*,* $\mathrm{NP} \subseteq \mathrm{PCP}_{1,7/8+\delta}(\mathcal{O}(\log n), 3)$ *and the verifier is restricted to use only functions that check the OR of three bits or their negations.*

It is NP-hard to approximate 3SAT better than $7/8 + \delta$.

The following GAP-problem is NP-hard for any $\epsilon > 0$.

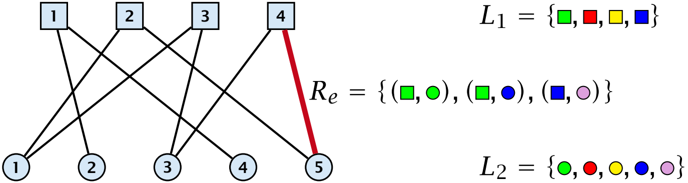
Given a graph $G = (V, E)$ composed of m independent sets of size 3 ($|V| = 3m$). Distinguish between

- the graph has a CLIQUE of size m
- the largest CLIQUE has size at most $(7/8 + \epsilon)m$

Label Cover

Input:

- bipartite graph $G = (V_1, V_2, E)$
- label sets L_1, L_2
- for every edge $(u, v) \in E$ a relation $R_{u,v} \subseteq L_1 \times L_2$ that describe assignments that make the edge *happy*.
- maximize number of happy edges



Label Cover

- an instance of label cover is (d_1, d_2)-regular if every vertex in L_1 has degree d_1 and every vertex in L_2 has degree d_2.
- if every vertex has the same degree d the instance is called d-regular

Minimization version:

- assign a set $L_x \subseteq L_1$ of labels to every node $x \in L_1$ and a set $L_y \subseteq L_2$ to every node $x \in L_2$
- make sure that for every edge (x, y) there is $\ell_x \in L_x$ and $\ell_y \in L_y$ s.t. $(\ell_x, \ell_y) \in R_{x,y}$
- minimize $\sum_{x \in L_1} |L_x| + \sum_{y \in L_2} |L_y|$ (total labels used)

MAX E3SAT via Label Cover

instance:

$$\Phi(x) = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_4 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4)$$

corresponding graph:

label sets: $L_1 = \{T, F\}^3, L_2 = \{T, F\}$ (T=true, F=false)

relation: $R_{C,x_i} = \{((u_i, u_j, u_k), u_i)\}$, where the clause C is over variables x_i, x_j, x_k and assignment (u_i, u_j, u_k) satisfies C

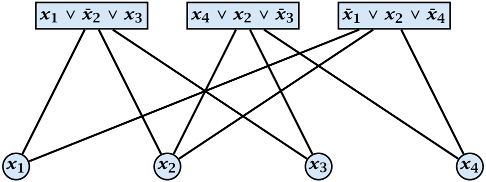
$$R = \{((F,F,F),F), ((F,T,F),F), ((F,F,T),T), ((F,T,T),T),$$
$$((T,T,T),T), ((T,T,F),F), ((T,F,F),F)\}$$

MAX E3SAT via Label Cover

instance:

$\Phi(x) = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_4 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4)$

corresponding graph:



label sets: $L_1 = \{T,F\}^3, L_2 = \{T,F\}$ (T=true, F=false)

relation: $R_{C,x_i} = \{((u_i,u_j,u_k),u_i)\}$, where the clause C is over variables x_i,x_j,x_k and assignment (u_i,u_j,u_k) satisfies C

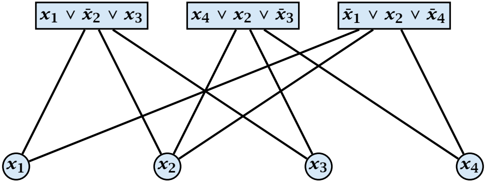
$$R = \{((F,F,F),F),((F,T,F),F),((F,F,T),T),((F,T,T),T),\\ ((T,T,T),T),((T,T,F),F),((T,F,F),F)\}$$

MAX E3SAT via Label Cover

instance:

$\Phi(x) = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_4 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4)$

corresponding graph:



label sets: $L_1 = \{T, F\}^3, L_2 = \{T, F\}$ (T=true, F=false)

relation: $R_{C,x_i} = \{((u_i, u_j, u_k), u_i)\}$, where the clause C is over variables x_i, x_j, x_k and assignment (u_i, u_j, u_k) satisfies C

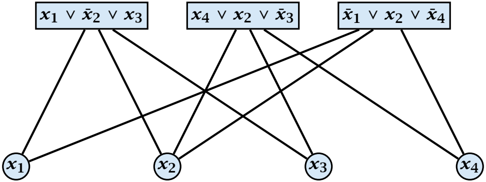
$$R = \{((F,F,F),F), ((F,T,F),F), ((F,F,T),T), ((F,T,T),T), ((T,T,T),T), ((T,T,F),F), ((T,F,F),F)\}$$

MAX E3SAT via Label Cover

instance:

$\Phi(x) = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_4 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4)$

corresponding graph:



label sets: $L_1 = \{T, F\}^3, L_2 = \{T, F\}$ (T=true, F=false)

relation: $R_{C,x_i} = \{((u_i, u_j, u_k), u_i)\}$, where the clause C is over variables x_i, x_j, x_k and assignment (u_i, u_j, u_k) satisfies C

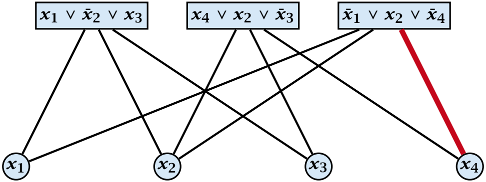
$$R = \{((F,F,F),F), ((F,T,F),F), ((F,F,T),T), ((F,T,T),T), ((T,T,T),T), ((T,T,F),F), ((T,F,F),F)\}$$

MAX E3SAT via Label Cover

instance:

$\Phi(x) = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_4 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4)$

corresponding graph:



label sets: $L_1 = \{T, F\}^3, L_2 = \{T, F\}$ (T=true, F=false)

relation: $R_{C,x_i} = \{((u_i, u_j, u_k), u_i)\}$, where the clause C is over variables x_i, x_j, x_k and assignment (u_i, u_j, u_k) satisfies C

$$R = \{((F,F,F),F), ((F,T,F),F), ((F,F,T),T), ((F,T,T),T), ((T,T,T),T), ((T,T,F),F), ((T,F,F),F)\}$$

MAX E3SAT via Label Cover

Lemma 97

If we can satisfy k out of m clauses in ϕ we can make at least $3k + 2(m - k)$ edges happy.

Proof:

- for V_2 use the setting of the assignment that satisfies k clauses
- for satisfied clauses in V_1 use the corresponding assignment to the clause-variables (gives $3k$ happy edges)
- for unsatisfied clauses flip assignment of one of the variables; this makes one incident edge unhappy (gives $2(m - k)$ happy edges)

MAX E3SAT via Label Cover

Lemma 97

If we can satisfy k out of m clauses in ϕ we can make at least $3k + 2(m - k)$ edges happy.

Proof:

- for V_2 use the setting of the assignment that satisfies k clauses
- for satisfied clauses in V_1 use the corresponding assignment to the clause-variables (gives $3k$ happy edges)
- for unsatisfied clauses flip assignment of one of the variables; this makes one incident edge unhappy (gives $2(m - k)$ happy edges)

MAX E3SAT via Label Cover

Lemma 97

If we can satisfy k out of m clauses in ϕ we can make at least $3k + 2(m - k)$ edges happy.

Proof:

- for V_2 use the setting of the assignment that satisfies k clauses
- for satisfied clauses in V_1 use the corresponding assignment to the clause-variables (gives $3k$ happy edges)
- for unsatisfied clauses flip assignment of one of the variables; this makes one incident edge unhappy (gives $2(m - k)$ happy edges)

MAX E3SAT via Label Cover

Lemma 97

If we can satisfy k out of m clauses in ϕ we can make at least $3k + 2(m - k)$ edges happy.

Proof:

- for V_2 use the setting of the assignment that satisfies k clauses
- for satisfied clauses in V_1 use the corresponding assignment to the clause-variables (gives $3k$ happy edges)
- for unsatisfied clauses flip assignment of one of the variables; this makes one incident edge unhappy (gives $2(m - k)$ happy edges)

MAX E3SAT via Label Cover

Lemma 98

If we can satisfy at most k clauses in Φ we can make at most $3k + 2(m - k) = 2m + k$ edges happy.

Proof:

MAX E3SAT via Label Cover

Lemma 98

If we can satisfy at most k clauses in Φ we can make at most $3k + 2(m - k) = 2m + k$ edges happy.

Proof:

- the labeling of nodes in V_2 gives an assignment
- every unsatisfied clause in this assignment cannot be assigned a label that satisfies all 3 incident edges
- hence at most $3m - (m - k) = 2m + k$ edges are happy

MAX E3SAT via Label Cover

Lemma 98
If we can satisfy at most k clauses in Φ we can make at most $3k + 2(m - k) = 2m + k$ edges happy.

Proof:

- the labeling of nodes in V_2 gives an assignment
- every unsatisfied clause in this assignment cannot be assigned a label that satisfies all 3 incident edges
- hence at most $3m - (m - k) = 2m + k$ edges are happy

MAX E3SAT via Label Cover

Lemma 98

If we can satisfy at most k clauses in Φ we can make at most $3k + 2(m - k) = 2m + k$ edges happy.

Proof:

- the labeling of nodes in V_2 gives an assignment
- every unsatisfied clause in this assignment cannot be assigned a label that satisfies all 3 incident edges
- hence at most $3m - (m - k) = 2m + k$ edges are happy

Hardness for Label Cover

We cannot distinguish between the following two cases

- all $3m$ edges can be made happy
- at most $2m + (7/8 + \epsilon)m \approx (\frac{23}{8} + \epsilon)m$ out of the $3m$ edges can be made happy

Hence, we cannot obtain an approximation constant $\alpha > \frac{23}{24}$.

Here α is a constant!!! Maybe a guarantee of the form $\frac{23}{8} + \frac{1}{m}$ is possible.

Hardness for Label Cover

We cannot distinguish between the following two cases

- all $3m$ edges can be made happy
- at most $2m + (7/8 + \epsilon)m \approx (\frac{23}{8} + \epsilon)m$ out of the $3m$ edges can be made happy

Hence, we cannot obtain an approximation constant $\alpha > \frac{23}{24}$.

Here α is a constant!!! Maybe a guarantee of the form $\frac{23}{8} + \frac{1}{m}$ is possible.

Hardness for Label Cover

We cannot distinguish between the following two cases

- all $3m$ edges can be made happy
- at most $2m + (7/8 + \epsilon)m \approx (\frac{23}{8} + \epsilon)m$ out of the $3m$ edges can be made happy

Hence, we cannot obtain an approximation constant $\alpha > \frac{23}{24}$.

Here α is a constant!!! Maybe a guarantee of the form $\frac{23}{8} + \frac{1}{m}$ is possible.

$(3, 5)$-regular instances

Theorem 99
There is a constant ρ s.t. MAXE3SAT is hard to approximate with a factor of ρ even if restricted to instances where a variable appears in exactly 5 clauses.

Then our reduction has the following properties:

- the resulting Label Cover instance is $(3, 5)$-regular
- it is hard to approximate for a constant $\alpha < 1$
- given a label ℓ_1 for x there is at most one label ℓ_2 for y that makes edge (x, y) happy (uniqueness property)

$(3, 5)$-regular instances

Theorem 99

There is a constant ρ s.t. MAXE3SAT is hard to approximate with a factor of ρ even if restricted to instances where a variable appears in exactly 5 clauses.

Then our reduction has the following properties:

- the resulting Label Cover instance is $(3, 5)$-regular
- it is hard to approximate for a constant $\alpha < 1$
- given a label ℓ_1 for x there is at most one label ℓ_2 for y that makes edge (x, y) happy (uniqueness property)

Regular instances

Theorem 100
If for a particular constant $\alpha < 1$ there is an α-approximation algorithm for Label Cover on 15-regular instances than P=NP.

Given a label ℓ_1 for $x \in V_1$ there is at most one label ℓ_2 for y that makes (x, y) happy. (uniqueness property)

Regular instances

proof...

Boosting

Given Label Cover instance I with $G = (V_1, V_2, E)$, label sets L_1 and L_2 we construct a new instance I':

- $V_1' = V_1^k = V_1 \times \cdots \times V_1$
- $V_2' = V_2^k = V_2 \times \cdots \times V_2$
- $L_1' = L_1^k = L_1 \times \cdots \times L_1$
- $L_2' = L_2^k = L_2 \times \cdots \times L_2$
- $E' = E^k = E \times \cdots \times E$

An edge $((x_1, \dots, x_k), (y_1, \dots, y_k))$ whose end-points are labelled by $(\ell_1^x, \dots, \ell_k^x)$ and $(\ell_1^y, \dots, \ell_k^y)$ is happy if $(\ell_i^x, \ell_i^y) \in R_{x_i, y_i}$ for all i.

Boosting

If I is regular than also I'.

If I has the uniqueness property than also I'.

Theorem 101
There is a constant $c > 0$ such if $\mathrm{OPT}(I) = |E|(1-\delta)$ then $\mathrm{OPT}(I') \leq |E'|(1-\delta)^{\frac{ck}{\log L}}$, where $L = |L_1| + |L_2|$ denotes total number of labels in I.

proof is highly non-trivial

Boosting

If I is regular than also I'.

If I has the uniqueness property than also I'.

Theorem 101

There is a constant $c > 0$ such if $\mathrm{OPT}(I) = |E|(1-\delta)$ then $\mathrm{OPT}(I') \leq |E'|(1-\delta)^{\frac{ck}{\log L}}$, where $L = |L_1| + |L_2|$ denotes total number of labels in I.

proof is highly non-trivial

Boosting

If I is regular than also I'.

If I has the uniqueness property than also I'.

Theorem 101
There is a constant $c > 0$ such if $\mathrm{OPT}(I) = |E|(1-\delta)$ then $\mathrm{OPT}(I') \leq |E'|(1-\delta)^{\frac{ck}{\log L}}$, where $L = |L_1| + |L_2|$ denotes total number of labels in I.

proof is highly non-trivial

Boosting

If I is regular than also I'.

If I has the uniqueness property than also I'.

Theorem 101

There is a constant $c > 0$ *such if* $\mathrm{OPT}(I) = |E|(1-\delta)$ *then* $\mathrm{OPT}(I') \leq |E'|(1-\delta)^{\frac{ck}{\log L}}$*, where* $L = |L_1| + |L_2|$ *denotes total number of labels in* I*.*

proof is highly non-trivial

Boosting

If I is regular than also I'.

If I has the uniqueness property than also I'.

Theorem 101

There is a constant $c > 0$ *such if* $\mathrm{OPT}(I) = |E|(1-\delta)$ *then* $\mathrm{OPT}(I') \leq |E'|(1-\delta)^{\frac{ck}{\log L}}$, *where* $L = |L_1| + |L_2|$ *denotes total number of labels in* I.

proof is highly non-trivial

Theorem 102

There are constants $c > 0$, $\delta < 1$ s.t. for any k we cannot distinguish regular instances for Label Cover in which either

- $\mathrm{OPT}(I) = |E|$, *or*
- $\mathrm{OPT}(I) = |E|(1-\delta)^{\frac{ck}{\log 10}}$

unless each problem in NP has an algorithm running in time $\mathcal{O}(n^{\mathcal{O}(k)})$.

Corollary 103

There is no α-approximation for Label Cover for any constant α.

Set Cover

Theorem 104
There exist regular Label Cover instances s.t. we cannot distinguish whether

- *all edges are satisfiable, or*
- *at most a* $1/\log^2(|L_2||E|)$*-fraction is satisfiable*

unless NP-problems have algorithms with running time $\mathcal{O}(n^{\mathcal{O}(\log\log n)})$.

choose $k = \frac{2\log 10}{c}\log_{1/(1-\delta)}(\log(|L_2||E|)) = \mathcal{O}(\log\log n)$.

Set Cover

Partition System (s, t, h)

- universe U of size s
- t pairs of sets $(A_1, \bar{A}_1), \dots, (A_t, \bar{A}_t)$;
  $A_i \subseteq U, \bar{A}_i = U \setminus A_i$
- choosing from any h pairs only one of A_i, $\bar{A}_i$ we do not cover the whole set U

For any h, t with $h \leq t$ there exist systems with $s = |U| \leq 2^{2h+2}t^2$.

Set Cover

Given a Label Cover instance we construct a Set Cover instance;

The universe is $E \times U$, where U is the universe of some partition system; ($t = |L_2|$, $h = (\log |E||L_2|)$)

for all $v \in V_2, j \in L_2$

$$S_{v,j} = \{((u,v),a) \mid (u,v) \in E, a \in A_j\}$$

for all $u \in V_1, i \in L_1$

$$S_{u,i} = \{((u,v),a) \mid (u,v) \in E, a \in \bar{A}_j, \text{where } (i,j) \in R_{(u,v)}\}$$

note that $S_{u,i}$ is well-defined because of the uniqueness property

Set Cover

Given a Label Cover instance we construct a Set Cover instance;

The universe is $E \times U$, where U is the universe of some partition system; $(t = |L_2|, h = (\log |E||L_2|))$

for all $v \in V_2, j \in L_2$

$$S_{v,j} = \{((u,v),a) \mid (u,v) \in E, a \in A_j\}$$

for all $u \in V_1, i \in L_1$

$$S_{u,i} = \{((u,v),a) \mid (u,v) \in E, a \in \bar{A}_j, \text{ where } (i,j) \in R_{(u,v)}\}$$

note that $S_{u,i}$ is well-defined because of the uniqueness property

Set Cover

Given a Label Cover instance we construct a Set Cover instance;

The universe is $E \times U$, where U is the universe of some partition system; ($t = |L_2|$, $h = (\log |E||L_2|)$)

for all $v \in V_2, j \in L_2$

$$S_{v,j} = \{((u,v),a) \mid (u,v) \in E, a \in A_j\}$$

for all $u \in V_1, i \in L_1$

$$S_{u,i} = \{((u,v),a) \mid (u,v) \in E, a \in \bar{A}_j, \text{where } (i,j) \in R_{(u,v)}\}$$

note that $S_{u,i}$ is well-defined because of the uniqueness property

Set Cover

Given a Label Cover instance we construct a Set Cover instance;

The universe is $E \times U$, where U is the universe of some partition system; ($t = |L_2|$, $h = (\log |E||L_2|)$)

for all $v \in V_2, j \in L_2$

$$S_{v,j} = \{((u,v),a) \mid (u,v) \in E, a \in A_j\}$$

for all $u \in V_1, i \in L_1$

$$S_{u,i} = \{((u,v),a) \mid (u,v) \in E, a \in \bar{A}_j, \text{ where } (i,j) \in R_{(u,v)}\}$$

note that $S_{u,i}$ is well-defined because of the uniqueness property

Set Cover

Given a Label Cover instance we construct a Set Cover instance;

The universe is $E \times U$, where U is the universe of some partition system; ($t = |L_2|$, $h = (\log |E||L_2|)$)

for all $v \in V_2, j \in L_2$

$$S_{v,j} = \{((u,v),a) \mid (u,v) \in E, a \in A_j\}$$

for all $u \in V_1, i \in L_1$

$$S_{u,i} = \{((u,v),a) \mid (u,v) \in E, a \in \bar{A}_j, \text{ where } (i,j) \in R_{(u,v)}\}$$

note that $S_{u,i}$ is well-defined because of the uniqueness property

Set Cover

Given a Label Cover instance we construct a Set Cover instance;

The universe is $E \times U$, where U is the universe of some partition system; ($t = |L_2|$, $h = (\log |E||L_2|)$)

for all $v \in V_2, j \in L_2$

$$S_{v,j} = \{((u,v),a) \mid (u,v) \in E, a \in A_j\}$$

for all $u \in V_1, i \in L_1$

$$S_{u,i} = \{((u,v),a) \mid (u,v) \in E, a \in \bar{A}_j, \text{ where } (i,j) \in R_{(u,v)}\}$$

note that $S_{u,i}$ is well-defined because of the uniqueness property

Suppose that we can make all edges happy.

Choose sets $S_{u,i}$'s and $S_{v,j}$'s, where i is the label we assigned to u, and j the label for v. ($|V_1|+|V_2|$ sets)

For an edge (u,v), $S_{v,j}$ contains $\{(u,v)\} \times A_j$. For a happy edge $S_{u,i}$ contains $\{(u,v)\} \times \bar{A}_j$.

Since all edges are happy we have covered the whole universe.

Suppose that we can make all edges happy.

Choose sets $S_{u,i}$'s and $S_{v,j}$'s, where i is the label we assigned to u, and j the label for v. ($|V_1|+|V_2|$ sets)

For an edge (u,v), $S_{v,j}$ contains $\{(u,v)\} \times A_j$. For a happy edge $S_{u,i}$ contains $\{(u,v)\} \times \bar{A}_j$.

Since all edges are happy we have covered the whole universe.

Suppose that we can make all edges happy.

Choose sets $S_{u,i}$'s and $S_{v,j}$'s, where i is the label we assigned to u, and j the label for v. ($|V_1|+|V_2|$ sets)

For an edge (u,v), $S_{v,j}$ contains $\{(u,v)\}\times A_j$. For a happy edge $S_{u,i}$ contains $\{(u,v)\}\times \bar{A}_j$.

Since all edges are happy we have covered the whole universe.

Suppose that we can make all edges happy.

Choose sets $S_{u,i}$'s and $S_{v,j}$'s, where i is the label we assigned to u, and j the label for v. ($|V_1|$+$|V_2|$ sets)

For an edge (u,v), $S_{v,j}$ contains $\{(u,v)\} \times A_j$. For a happy edge $S_{u,i}$ contains $\{(u,v)\} \times \bar{A}_j$.

Since all edges are happy we have covered the whole universe.

Suppose that we can make all edges happy.

Choose sets $S_{u,i}$'s and $S_{v,j}$'s, where i is the label we assigned to u, and j the label for v. ($|V_1|+|V_2|$ sets)

For an edge (u,v), $S_{v,j}$ contains $\{(u,v)\} \times A_j$. For a happy edge $S_{u,i}$ contains $\{(u,v)\} \times \bar{A}_j$.

Since all edges are happy we have covered the whole universe.

Lemma 105

Given a solution to the set cover instance using at most $\frac{h}{8}(|V_1| + |V_2|)$ sets we can find a solution to the Label Cover instance satisfying at least $\frac{2}{h^2}|E|$ edges.

- n_u: number of $S_{u,i}$'s in cover
- n_v: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge (u,v) we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i,j) \in R_{u,v}$ (making (u,v) happy)
- we choose a random label for u from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for v from the (at most $h/2$) $S_{v,j}$-sets
- (u,v) gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

- n_u: number of $S_{u,i}$'s in cover
- n_v: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge (u,v) we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i,j) \in R_{u,v}$ (making (u,v) happy)
- we choose a random label for u from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for v from the (at most $h/2$) $S_{v,j}$-sets
- (u,v) gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

- n_u: number of $S_{u,i}$'s in cover
- n_v: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge (u, v) we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i, j) \in R_{u,v}$ (making (u, v) happy)
- we choose a random label for u from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for v from the (at most $h/2$) $S_{v,j}$-sets
- (u, v) gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

- n_u: number of $S_{u,i}$'s in cover
- n_v: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge (u,v) we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i,j) \in R_{u,v}$ (making (u,v) happy)
- we choose a random label for u from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for v from the (at most $h/2$) $S_{v,j}$-sets
- (u,v) gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

- n_u: number of $S_{u,i}$'s in cover
- n_v: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge (u, v) we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i, j) \in R_{u,v}$ (making (u, v) happy)
- we choose a random label for u from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for v from the (at most $h/2$) $S_{v,j}$-sets
- (u, v) gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

- n_u: number of $S_{u,i}$'s in cover
- n_v: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge (u, v) we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i, j) \in R_{u,v}$ (making (u, v) happy)
- we choose a random label for u from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for v from the (at most $h/2$) $S_{v,j}$-sets
- (u, v) gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

- n_u: number of $S_{u,i}$'s in cover
- n_v: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge (u, v) we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i, j) \in R_{u,v}$ (making (u, v) happy)
- we choose a random label for u from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for v from the (at most $h/2$) $S_{v,j}$-sets
- (u, v) gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

- n_u: number of $S_{u,i}$'s in cover
- n_v: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge (u, v) we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i, j) \in R_{u,v}$ (making (u, v) happy)
- we choose a random label for u from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for v from the (at most $h/2$) $S_{v,j}$-sets
- (u, v) gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

Set Cover

Theorem 106
There is no $\frac{1}{32}\log N$-approximation for the unweighted Set Cover problem unless problems in NP can be solved in time $\mathcal{O}(n^{\mathcal{O}(\log\log n)})$.

Given label cover instance (V_1, V_2, E), label sets L_1 and L_2;

Set $h = \log(|E||L_2|)$ and $t = |L_2|$; Size of partition system is

$$s = |U| = 2^{2h+2}t^2 = 4(|E||L_2|)^2|L_2|^2 = 4|E|^2|L_2|^4$$

The size of the ground set is then

$$N = |E||U| = 4|E|^3|L_2|^4 \le (|E||L_2|)^4$$

for sufficiently large $|E|$. Then $h \ge \frac{1}{4}\log N$.

If we get an instance where all edges are satisfiable there exists a cover of size only $|V_1| + |V_2|$.

If we find a cover of size at most $\frac{h}{8}(|V_1| + |V_2|)$ we can use this to satisfy at least a fraction of $2/h^2 \ge 1/\log^2(|E||L_2|)$ of the edges. this is not possible...

Given label cover instance (V_1, V_2, E), label sets L_1 and L_2;

Set $h = \log(|E||L_2|)$ and $t = |L_2|$; Size of partition system is

$$s = |U| = 2^{2h+2}t^2 = 4(|E||L_2|)^2|L_2|^2 = 4|E|^2|L_2|^4$$

The size of the ground set is then

$$N = |E||U| = 4|E|^3|L_2|^4 \le (|E||L_2|)^4$$

for sufficiently large $|E|$. Then $h \ge \frac{1}{4}\log N$.

If we get an instance where all edges are satisfiable there exists a cover of size only $|V_1| + |V_2|$.

If we find a cover of size at most $\frac{h}{8}(|V_1| + |V_2|)$ we can use this to satisfy at least a fraction of $2/h^2 \ge 1/\log^2(|E||L_2|)$ of the edges. this is not possible...

Given label cover instance (V_1, V_2, E), label sets L_1 and L_2;

Set $h = \log(|E||L_2|)$ and $t = |L_2|$; Size of partition system is

$$s = |U| = 2^{2h+2}t^2 = 4(|E||L_2|)^2|L_2|^2 = 4|E|^2|L_2|^4$$

The size of the ground set is then

$$N = |E||U| = 4|E|^3|L_2|^4 \leq (|E||L_2|)^4$$

for sufficiently large $|E|$. Then $h \geq \frac{1}{4}\log N$.

If we get an instance where all edges are satisfiable there exists a cover of size only $|V_1| + |V_2|$.

If we find a cover of size at most $\frac{h}{8}(|V_1| + |V_2|)$ we can use this to satisfy at least a fraction of $2/h^2 \geq 1/\log^2(|E||L_2|)$ of the edges. this is not possible...

Given label cover instance (V_1, V_2, E), label sets L_1 and L_2;

Set $h = \log(|E||L_2|)$ and $t = |L_2|$; Size of partition system is

$$s = |U| = 2^{2h+2}t^2 = 4(|E||L_2|)^2|L_2|^2 = 4|E|^2|L_2|^4$$

The size of the ground set is then

$$N = |E||U| = 4|E|^3|L_2|^4 \le (|E||L_2|)^4$$

for sufficiently large $|E|$. Then $h \ge \frac{1}{4}\log N$.

If we get an instance where all edges are satisfiable there exists a cover of size only $|V_1| + |V_2|$.

If we find a cover of size at most $\frac{h}{8}(|V_1| + |V_2|)$ we can use this to satisfy at least a fraction of $2/h^2 \ge 1/\log^2(|E||L_2|)$ of the edges. this is not possible...

Given label cover instance (V_1, V_2, E), label sets L_1 and L_2;

Set $h = \log(|E||L_2|)$ and $t = |L_2|$; Size of partition system is

$$s = |U| = 2^{2h+2}t^2 = 4(|E||L_2|)^2|L_2|^2 = 4|E|^2|L_2|^4$$

The size of the ground set is then

$$N = |E||U| = 4|E|^3|L_2|^4 \le (|E||L_2|)^4$$

for sufficiently large $|E|$. Then $h \ge \frac{1}{4}\log N$.

If we get an instance where all edges are satisfiable there exists a cover of size only $|V_1| + |V_2|$.

If we find a cover of size at most $\frac{h}{8}(|V_1| + |V_2|)$ we can use this to satisfy at least a fraction of $2/h^2 \ge 1/\log^2(|E||L_2|)$ of the edges. this is not possible...

Partition Systems

Lemma 107
Given h *and* t *there is a partition system of size* $s = 2^h h \ln(4t) \le 2^{2h+2}t^2$.

We pick t sets at random from the possible $2^{|U|}$ subsets of U.

Fix a choice of h of these sets, and a choice of h bits (whether we choose A_i or $\bar{A}_i$). There are $2^h \cdot \binom{t}{h}$ such choices.

Partition Systems

Lemma 107

Given h and t there is a partition system of size $s = 2^h h \ln(4t) \le 2^{2h+2} t^2$.

We pick t sets at random from the possible $2^{|U|}$ subsets of U.

Fix a choice of h of these sets, and a choice of h bits (whether we choose A_i or $\bar{A}_i$). There are $2^h \cdot \binom{t}{h}$ such choices.

Partition Systems

Lemma 107
Given h and t there is a partition system of size $s = 2^h h \ln(4t) \leq 2^{2h+2}t^2$.

We pick t sets at random from the possible $2^{|U|}$ subsets of U.

Fix a choice of h of these sets, and a choice of h bits (whether we choose A_i or $\bar{A}_i$). There are $2^h \cdot \binom{t}{h}$ such choices.

What is the probability that a given choice covers U?

The probability that an element $u \in A_i$ is $1/2$ (same for $\bar{A}_i$).

The probability that u is covered is $1 - \frac{1}{2^h}$.

The probability that all u are covered is $(1 - \frac{1}{2^h})^s$

The probability that there exists a choice such that all u are covered is at most

$$\binom{t}{h} 2^h \left(1 - \frac{1}{2^h}\right)^s \leq (2t)^h e^{-s/2^h} = (2t)^h \cdot e^{-h \ln(4t)} \leq \frac{1}{2^h}$$

What is the probability that a given choice covers U?

The probability that an element $u \in A_i$ is $1/2$ (same for $\bar{A}_i$).

The probability that u is covered is $1 - \frac{1}{2^h}$.

The probability that all u are covered is $(1 - \frac{1}{2^h})^s$

The probability that there exists a choice such that all u are covered is at most

$$\binom{t}{h} 2^h \left(1 - \frac{1}{2^h}\right)^s \leq (2t)^h e^{-s/2^h} = (2t)^h \cdot e^{-h \ln(4t)} \leq \frac{1}{2^h}$$

What is the probability that a given choice covers U?

The probability that an element $u \in A_i$ is $1/2$ (same for $\bar{A}_i$).

The probability that u is covered is $1 - \frac{1}{2^h}$.

The probability that all u are covered is $(1 - \frac{1}{2^h})^s$

The probability that there exists a choice such that all u are covered is at most

$$\binom{t}{h} 2^h \left(1 - \frac{1}{2^h}\right)^s \leq (2t)^h e^{-s/2^h} = (2t)^h \cdot e^{-h \ln(4t)} \leq \frac{1}{2^h}$$

What is the probability that a given choice covers U?

The probability that an element $u \in A_i$ is $1/2$ (same for $\bar{A}_i$).

The probability that u is covered is $1 - \frac{1}{2^h}$.

The probability that all u are covered is $(1 - \frac{1}{2^h})^s$

The probability that there exists a choice such that all u are covered is at most

$$\binom{t}{h} 2^h \left(1 - \frac{1}{2^h}\right)^s \leq (2t)^h e^{-s/2^h} = (2t)^h \cdot e^{-h \ln(4t)} \leq \frac{1}{2^h}$$

What is the probability that a given choice covers U?

The probability that an element $u \in A_i$ is $1/2$ (same for $\bar{A}_i$).

The probability that u is covered is $1 - \frac{1}{2^h}$.

The probability that all u are covered is $(1 - \frac{1}{2^h})^s$

The probability that there exists a choice such that all u are covered is at most

$$\binom{t}{h} 2^h \left(1 - \frac{1}{2^h}\right)^s \le (2t)^h e^{-s/2^h} = (2t)^h \cdot e^{-h\ln(4t)} \le \frac{1}{2^h}$$

What is the probability that a given choice covers U?

The probability that an element $u \in A_i$ is $1/2$ (same for $\bar{A}_i$).

The probability that u is covered is $1 - \frac{1}{2^h}$.

The probability that all u are covered is $(1 - \frac{1}{2^h})^s$

The probability that there exists a choice such that all u are covered is at most

$$\binom{t}{h} 2^h \left(1 - \frac{1}{2^h}\right)^s \leq (2t)^h e^{-s/2^h} = (2t)^h \cdot e^{-h \ln(4t)} \leq \frac{1}{2^h}$$