SS 2012

# Efficient Algorithms and Data Structures II


Harald Räcke

Fakultät für Informatik
TU München



http://www14.in.tum.de/lehre/2012SS/ea/

Summer Term 2012




to be filled...



**Algorithm 1** Pivot$(N, B, A, b, c, v, \ell, e)$

let $\hat{A}$ be the new $m \times n$-matrix
$\hat{b}_e \leftarrow b_\ell / a_{\ell e}$
**for** $j \in N - \{e\}$ **do** $\hat{a}_{ej} \leftarrow a_{\ell j} / a_{\ell e}$
$\hat{a}_{e\ell} \leftarrow 1/a_{\ell e}$
**for** $i \in B - \{\ell\}$ **do**
    $\hat{b}_i \leftarrow b_i - a_{ie}\hat{b}_e$
    **for** $j \in N - \setminus\{e\}$ **do** $\hat{a}_{ij} = a_{ij} - a_{ie}\hat{a}_{ej}$
    $\hat{a}_{i\ell} \leftarrow -a_{ie}\hat{a}_{e\ell}$
$\hat{v} \leftarrow v + c_e\hat{b}_e$
**for** $j \in N - \{e\}$ **do** $\hat{c}_j \leftarrow c_j - c_e\hat{a}_{ej}$
$\hat{c}_\ell \leftarrow -c_e\hat{a}_{e\ell}$
$\hat{N} \leftarrow N - \{e\} \cup \{\ell\}$; $\hat{B} \leftarrow B - \{\ell\} \cup \{e\}$

**Algorithm 2** Simplex$(A, b, c)$

$(N, B, A, b, c, v) \leftarrow \text{InitializeSimplex(A, b, c)}$
let $\Delta$ be new $n$-dimensional vector
**while** some index $j \in N$ has $c_j > 0$ **do**
    choose index $e \in N$ with $c_e > 0$
    **for** each $i \in B$ **do**
        **if** $a_{ie} > 0$ **then** $\Delta_i \leftarrow b_i / a_{ie}$
        **else** $\Delta_i \leftarrow \infty$
    choose index $\ell \in B$ that minimizes $\Delta_i$
    **if** $\Delta_\ell = \infty$ **return** "'unbounded"'
    **else**$(N, B, A, b, c, v) = \text{Pivot}(N, B, A, b, c, v, \ell, e)$
**for** $i \in B$ **do** $\bar{x}_i \leftarrow b_i$;
**for** $i \in N$ **do** $\bar{x}_i \leftarrow 0$;
**return** $\bar{x}$

# Simplex Algorithm

Questions/Observations:

- How do we find the initial feasible solution?
- The final solution will be feasible, since each pivot-step guarantees that no variable becomes negative (no problem);
- Do we terminate?
- Is the final solution optimal?



# Simplex Algorithm

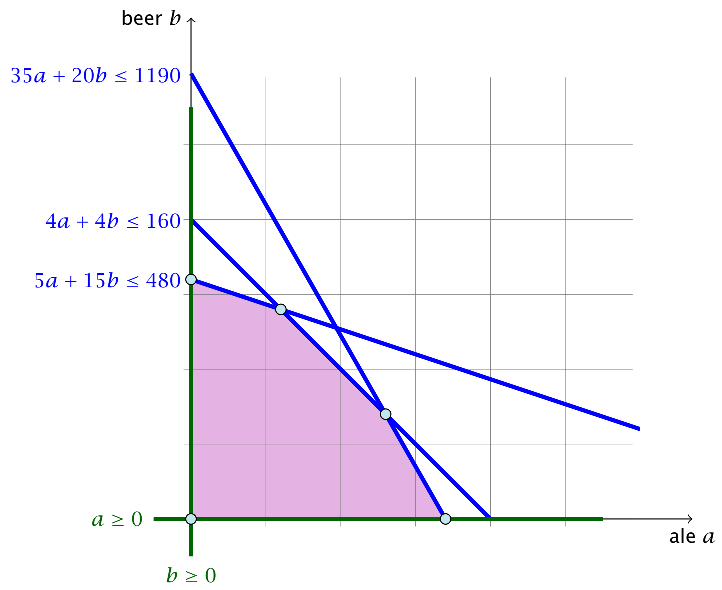
**The simplex algorithm only considers basic feasible solutions!**

### Lemma 1

*If a given linear program LP is bounded then there is a basic feasible solution that gives the optimum value.*

**Basic feasible solutions correspond to corner points of the feasible region!**








Let $P = \{x \mid Ax = b, x \ge 0\} \subseteq \mathbb{R}^d$.

### Definition 2

$x$ is a vertex of $P$ if there is no $y$ with $x + y \in P$ and $x - y \in P$.

Let $P = \{x \mid Ax = b, x \geq 0\} \subseteq \mathbb{R}^d$.

**Lemma 3**

*Then for each $x \in P$ there exists a vertex $x' \in P$ with $c^t x' \geq c^t x$.*

This means that also the maximum is obtained at a vertex of $P$.



Let $P = \{x \mid Ax = b, x \geq 0\}$, and let $x \in P$. If $x$ is a vertex of $P$ there is nothing to prove.

Otw. there exist $y \neq 0$ with $x \pm y \in P$.

Since $A(x - y) = A(x + y)$ (equal to $b$) we have $Ay = 0$

Since, $c^t(x \pm y) = c^t x \pm c^t y$ we have $c^t y = 0$ since $x$ is maximal.

Wlog. we can assume that there is a $j \in \{1 \dots d\}$ with $y_j < 0$ (otw. redefine $y$ as $-y$).



Define

- $\lambda = \min\{-\frac{x_j}{y_j} \mid y_j < 0\}$.
- That's the largest $\lambda$ s.t. $x + \lambda y \geq 0$.
- $A(x + \lambda y) = b$.
- $(x + \lambda y)_k = 0$ but $x_k > 0$.
- Replace $x$ by $x + \lambda y$. We have reduced the number of non-zero components.



Let $P = \{x \mid Ax = b, x \geq 0\}$ and $x \in P$. Let $A_x$ denote the sub-matrix of $A$ that contains columns $j$ with $x_j > 0$.

**Lemma 4**

*$x$ is a vertex of $P$ if and only if the columns of $A_x$ are linearly independent.*

**Proof: (⇐)**

Assume for contradiction that $x$ is not a vertex. Then there exists $y \neq 0$ with $x \pm y \in P$. Let $A_y$ denote the sub-matrix corresponding to the non-zero components of $y$.

As before we get $Ay = 0$ (from $A(x - y) = A(x + y)$). Since $y \neq 0$ $A_y$ has linearly dependent columns

$x_j = 0 \Rightarrow y_j = 0$, since $x + y \geq 0$ and $x - y \geq 0$. Therefore, $A_y$ contains a subset of the columns of $x$.

Hence, $A_x$ contains linearly dependent columns.



**Proof: (⇐)**

Suppose $A_x$, has linearly dependent rows. Then there is $y \neq 0$ with $A_x y = 0$.

By adding zero-components to $y$ we get $y \neq 0$ with $Ay = 0$ and

$$x_j = 0 \Rightarrow y_j = 0$$

For small enough $\epsilon > 0$ this gives $x + \epsilon y \in P$ and $x - \epsilon y \in P$.

Hence, $x$ is not a vertex.



A vertex/corner-point is defined by choosing a set of linearly independent columns.

We can assume wlog. that the row-rank of $A$ (in the slack form) is $m$ (otw. we can remove a constraint).

If $x$ is a vertex then $A_x$ has full column-rank ($\leq m$).

$A_x$ can be extended to a quadratic ($m \times m$-matrix) with full column rank.

A quadratic matrix $A_B$ with full rank is called basis.



## Termination

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

Does it always increase?

## Termination

The objective function may not decrease!

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

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

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.



**How to choose the pivot-elements:**

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



**How do we come up with an initial solution?**

- $Ax \leq b, x \geq 0$, and $b \geq 0$.
- The standard slack from for this problem is $Ax + E_m s = b, x \geq 0, s \geq 0$, where $s$ denotes the vector of slack variables.
- Then $s = b$, $x = 0$ is a basic feasible solution.
- 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. $Ax + E_m v = b$, $x \geq 0$, $v \geq 0$ using Simplex. $x = 0$, $v = b$ is initial feasible.
3. If $\sum_i s_i > 0$ 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

In the end we have an LP of the form $\max\{v + c^t x \mid Ax = b, x \geq 0\}$ (recall that $A$ is not the original matrix), with $c_i^t \leq 0$ for all $i$. Furthermore, each basic variable only appears in one equation with coefficient $+1$.

Of course, LP$' = \max\{c^t x \mid Ax = b, x \geq 0\}$ has the same optimum solution (with different objective function value).

The best we can hope for (for LP$'$) is an objective function value of $0$ as $c^t \leq 0$ and $x \geq 0$ is required.

The basic feasible solution achieves that and is therefore optimal.



## Duality

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

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

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 $\sum_i y_i a_{ij} \geq c_i$ then $\sum_i y_i b_i$ will be an upper bound.



## Duality

**Definition 5**
Let $z = \max\{c^t x \mid Ax \geq b, x \geq 0\}$ be a linear program $P$ (called the primal linear program).

The linear program $D$ defined by

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

is called the dual problem.



## Duality

**Lemma 6**
*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

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

## 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 7 (Weak Duality)**

*Let $\hat{x}$ be a 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

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

$A\hat{x} \le b \Rightarrow y^t A\hat{x} \le \hat{y}^t b$ $(\hat{y} \ge 0)$

This gives

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

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

If $P$ is unbounded then $D$ is infeasible.



The following linear programs form a primal dual pair:

$$z = \max\{c^t x \mid Ax = b, x \ge 0\}$$
$$w = \min\{b^t y \mid A^t y \le c\}$$



proof...

## Strong Duality

### Theorem 8 (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 9 (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 $(y - x^*)^t(x - x^*) \le 0$.*

### Lemma 10 (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.*



**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 $X' = \{x \in X \mid \|y - x\| \le \|y - x'\|\}$. This set is closed and bounded.
- Applying Weierstrass gives the existence.



**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 11 (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 $(y - x^*)^t(x - x^*) \leq 0$ 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 12 (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, $b \notin S$.

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 13 (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* $A^t y \ge 0$, $b^t y < 0$, $y \ge 0$



## Proof of Farkas Lemma II

proof...



## Proof of Strong Duality

$$P\colon\ z = \max\{c^t x \mid Ax \le b, x \ge 0\}$$

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

### Theorem 14 (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

$\boldsymbol{z \le w}$: follows from weak duality

$\boldsymbol{z \ge w}$:
We show $z < \alpha$ implies $w < \alpha$.

$$\begin{array}{lrcl} \exists x \in \mathbb{R}^n & & & \\ \text{s.t.} & Ax & \le & b \\ & -c^t x & \le & -\alpha \\ & x & \ge & 0 \end{array}$$

$$\begin{array}{lrcl} \exists y \in \mathbb{R}^m; z \in \mathbb{R} & & & \\ \text{s.t.} & A^t y - cz & \ge & 0 \\ & yb^t - \alpha z & < & 0 \\ & y, z & \ge & 0 \end{array}$$

From the definition of $\alpha$ we know that the first system is infeasible; hence the second must be feasible.

## Proof of Strong Duality

$$\begin{array}{rrcl} \exists y \in \mathbb{R}^m; z \in \mathbb{R} & & & \\ \text{s.t.} & A^t y - cz & \geq & 0 \\ & yb^t - \alpha z & < & 0 \\ & y, z & \geq & 0 \end{array}$$

If the solution $y, z$ has $z = 0$ we have that

$$\begin{array}{rrcl} \exists y \in \mathbb{R}^m & & & \\ \text{s.t.} & A^t y & \geq & 0 \\ & yb^t & < & 0 \\ & y & \geq & 0 \end{array}$$

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 $y, z$ with $z > 0$.

We can rescale this solution (scaling both $y$ and $z$) s.t. $z = 1$.

Then $y$ is feasible for the dual but $b^t y < \alpha$. This means that $w < \alpha$.



## Simplex in Matrix Notation

Given a linear program in slack form

$$z = \max\{c^t x \mid Ax = b; x \geq 0\} \ .$$

The simplex algorithm (for a given basis $B$) writes the equations in the following form:

$$\begin{array}{rcl} z & = & \hat{v} + \hat{c}_N^t x_N \\ x_B & = & \hat{b} - \hat{A}_N x_N \end{array}$$

Here $\hat{A}_N$ is a matrix that contains one column for every non-basis variable $x_i$, $i \in N$. Similarly, $\hat{c}_N^t$ is an $|N|$-dimensional vector.



## Simplex in Matrix Notation

We can directly compute the matrix $\hat{A}_N$, the vector $\hat{b}$, and the constant term $\hat{v}$ for a given basis $B$.

We have

$$z = Ax + b = A_B x_B + A_N x_N + b$$

This gives

$$A_B x_B = b - A_N x_N$$

and hence

$$x_B = \underbrace{\boxed{A_B^{-1} b}}_{\hat{b}} - \underbrace{\boxed{A_B^{-1} A_N}}_{\hat{A}_N} x_N$$

since the matrix $A_B$ is linearly independent.

## Simplex in Matrix Notation

The objective function is given by $z = c^t x = c_B^t x_B + c_N^t x_N$.

Plugging in $x_B = A_B^{-1}b - A_B^{-1}A_N x_N$ gives

$$\begin{aligned} z &= c_B^t(A_B^{-1}b - A_B^{-1}A_N x_N) + c_N^t x_N \\ &= \underbrace{\boxed{c_B^t A_B^{-1} b}}_{\hat{v}} + \underbrace{\boxed{(c_N^t - c_B^t A_B^{-1} A_N)}}_{\hat{c}_N} x_N \end{aligned}$$



## Simplex in Matrix Notation

The non-constant part of the objective function in any iteration is of the form

$$(c^t - y^t A)x$$

this means the optimization direction is given by the inital direction plus a linear combination of the rows of $A$.

To see this observe that

$$\begin{aligned} &(c^t - \overbrace{\boxed{c_B^t A_B^{-1}}}^{y^t} A)x \\ &\quad = c_B^t x_B + c_N^t x_N - c_B^t A_B^{-1} A_B x_B - c_B^t A_B^{-1} A_N x_N \\ &\quad = c_N^t x_N - c_B^t A_B^{-1} A_N x_N \\ &\quad = \underbrace{\boxed{(c_N^t - c_B^t A_B^{-1} A_N)}}_{\hat{c}_N^t} x_N \end{aligned}$$



## Simplex in Matrix Notation

When Simplex terminates we have

$$\hat{c}_N^t = c_N^t - c_B^t A_B^{-1} A_N \le 0$$

**$y$ is a feasible solution to the dual:**

$$\begin{aligned} y^t A &= \begin{bmatrix} y^t A_B & y^t A_N \end{bmatrix} \\ &= \begin{bmatrix} c_B^t A_B^{-1} A_B & c_B^t A_B^{-1} A_N \end{bmatrix} \\ &\ge \begin{bmatrix} c_B^t & c_N^t \end{bmatrix} \\ &= c^t \end{aligned}$$

(Here we assumed that $B = \{1, \dots, m\}$ which can be obtained by renaming variables; without this assumption the notation becomes much more cumbersome)



## Simplex in Matrix Notation

The profit of the primal basic feasible solution ($x_N = 0$; $x_B = \hat{b} = A_B^{-1}b$) is equal to the cost of the dual solution $y$.

$$\begin{aligned} y^t b &= c_B^t A_B^{-1} b \\ &= c_B^t x_B \\ &= c_N^t x_N + c_B^t x_B \\ &= c^t x \end{aligned}$$

## Complementary Slackness

**Lemma 15**

*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.



## Complementary Slackness

**Proof:**

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.

$$\sum_j x_j^* ((y^t A)_j - c_j) = 0$$

From the constraint of the dual it follows that $y^t A \ge 0$. Hence the left hand side is a sum over the product of non-negative number. Hence, if e.g. $(y^t A)_j - c_j > 0$ (the $j$-th constraint in the dual is not tight) then $x_j = 0$ (2.). The result for (1./3./4.) follows similarly.



- 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 $\varepsilon_C$, $\varepsilon_H$, and $\varepsilon_M$, 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

If $\epsilon$ is small enough then the optimum dual solution $y^*$ does 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.



## Flows

### Definition 16

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 17

The value of an $(s,t)$-flow $f$ is defined as

$$\mathrm{val}(f) = \sum_x f_{sx} - \sum_x f_{xs} \ .$$

**Maximum Flow Problem:** Find an $(s,t)$-flow with maximum value.



## LP-Formulation of Maxflow

$$\begin{array}{llrcll}
\max & & \sum_z f_{sz} - \sum_z f_{zs} & & & \\
\text{s.t.} & \forall (z,w) \in V \times V & f_{zw} & \leq & c_{zw} & \ell_{zw} \\
 & \forall w \neq s,t & \sum_z f_{wz} - \sum_z f_{zw} & = & 0 & p_w \\
 & & f_{zw} & \geq & 0 &
\end{array}$$

$$\begin{array}{lllcl}
\min & & \sum_{(xy)} c_{xy} \ell_{xy} & & \\
\text{s.t.} & f_{xy}\ (x,y \neq s,t): & 1\ell_{xy} + 1p_x - 1p_y & \geq & 0 \\
 & f_{sy}\ (y \neq s,t): & 1\ell_{sy} \quad\quad - 1p_y & \geq & 1 \\
 & f_{xs}\ (x \neq s,t): & 1\ell_{xs} + 1p_x & \geq & -1 \\
 & f_{ty}\ (y \neq s,t): & 1\ell_{ty} \quad\quad - 1p_y & \geq & 0 \\
 & f_{xt}\ (x \neq s,t): & 1\ell_{xt} + 1p_x & \geq & 0 \\
 & f_{st}: & 1\ell_{st} & \geq & 1 \\
 & f_{ts}: & 1\ell_{ts} & \geq & -1 \\
 & & \ell_{xy} & \geq & 0
\end{array}$$

## LP-Formulation of Maxflow

$$
\begin{array}{lllrcl}
\min & & & \multicolumn{3}{c}{\sum_{(xy)} c_{xy} f_{xy}} \\
\text{s.t.} & f_{xy} & (x,y \neq s,t): & 1\ell_{xy} + 1p_x - 1p_y & \geq & 0 \\
 & f_{sy} & (y \neq s,t): & 1\ell_{sy} + (-1) - 1p_y & \geq & 0 \\
 & f_{xs} & (x \neq s,t): & 1\ell_{xs} + 1p_x - (-1) & \geq & 0 \\
 & f_{ty} & (y \neq s,t): & 1\ell_{ty} + 0 - 1p_y & \geq & 0 \\
 & f_{xt} & (x \neq s,t): & 1\ell_{xt} + 1p_x - 0 & \geq & 0 \\
 & f_{st}: & & 1\ell_{st} + (-1) - 0 & \geq & 0 \\
 & f_{ts}: & & 1\ell_{ts} + 0 - (-1) & \geq & 0 \\
 & & & \ell_{xy} & \geq & 0
\end{array}
$$



## LP-Formulation of Maxflow

$$
\begin{array}{lllrcl}
\min & & & \multicolumn{3}{c}{\sum_{(xy)} c_{xy} \ell_{xy}} \\
\text{s.t.} & f_{xy} & (x,y \neq s,t): & 1\ell_{xy} + 1p_x - 1p_y & \geq & 0 \\
 & f_{sy} & (y \neq s,t): & 1\ell_{sy} + p_s - 1p_y & \geq & 0 \\
 & f_{xs} & (x \neq s,t): & 1\ell_{xs} + 1p_x - p_s & \geq & 0 \\
 & f_{ty} & (y \neq s,t): & 1\ell_{ty} + p_t - 1p_y & \geq & 0 \\
 & f_{xt} & (x \neq s,t): & 1\ell_{xt} + 1p_x - p_t & \geq & 0 \\
 & f_{st}: & & 1\ell_{st} + p_s - p_t & \geq & 0 \\
 & f_{ts}: & & 1\ell_{ts} + p_t - p_s & \geq & 0 \\
 & & & \ell_{xy} & \geq & 0
\end{array}
$$

with $p_t = 0$ and $p_s = -1$.



## LP-Formulation of Maxflow

$$
\begin{array}{llrcl}
\min & & \multicolumn{3}{c}{\sum_{(xy)} c_{xy} \ell_{xy}} \\
\text{s.t.} & f_{xy}: & 1\ell_{xy} + 1p_x - 1p_y & \geq & 0 \\
 & & \ell_{xy} & \geq & 0 \\
 & & p_s & = & -1 \\
 & & p_t & = & 0
\end{array}
$$



We can interpret the $\ell_{xy}$ 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 a triangle inequality $(d(x,t) \leq d(x,y) + d(y,t) \Rightarrow d(x,t) \leq \ell_{xy} + d(y,t))$.

**If we would have formulated the primal differently by multiplying the equality-constraint by $-1$ we would have had an easier interpretation of dual variables. Set $p_s = 1; p_t = 0$ and interpret $p_x$ as the distance to $t$.**

One can show that the optimum LP-solution for the Maxflow problem 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.



**How do we come up with an initial solution?**

- $Ax \leq b, x \geq 0$, and $b \geq 0$.
- The standard slack from for this problem is $Ax + E_m s = b, x \geq 0, s \geq 0$, where $s$ denotes the vector of slack variables.
- Then $s = b$, $x = 0$ is a basic feasible solution.
- 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. $Ax + E_m v = b$, $x \geq 0$, $v \geq 0$ 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.



# 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:**
Change $\mathrm{LP} := \max\{c^t x, Ax = b; x \geq 0\}$ 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 $\mathrm{LP}'$ (note that columns in $A_B$ are linearly independent).
III. LP has no degenerate basic solutions

## Pertubation

Let $B$ be index set of a basis with basic solution

$$x_B^* = A_B^{-1} b \geq 0 \quad \text{(i.e. } B \text{ is feasible)}$$

Fix

$$b' := b + A_B \begin{pmatrix} \varepsilon \\ \vdots \\ \varepsilon^m \end{pmatrix} \text{ for } \varepsilon > 0 \ .$$

This is the pertubation 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 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 $\varepsilon$ 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.

## Lexicographic Pivoting

Doing calculations with perturbed instances may be costly. Also the right choice of $\varepsilon$ is difficult.

**Idea:**
Simulate behaviour of LP' without explicitly doing a perturbation.



## Lexicographic Pivoting

We choose the entering variable arbitrarily as before ($\hat{c}_e > 0$, 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

LP chooses an arbitrary leaving variable that has $\hat{a}_{\ell e} < 0$ and minimizes

$$\theta_\ell = -\frac{\hat{b}_\ell}{\hat{a}_{\ell e}} = -\frac{(A_B^{-1}b)_\ell}{(A_B^{-1}A_{*e})_\ell} \ .$$

$\ell$ 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 18**
$u \leq_{\text{lex}} v$ 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

$$\theta_\ell = -\frac{\left(A_B^{-1}\left(b+\begin{pmatrix}\varepsilon\\ \vdots\\ \varepsilon^m\end{pmatrix}\right)\right)_\ell}{(A_B^{-1}A_{*e})_\ell} = -\frac{\left(A_B^{-1}(b \mid E_m)\begin{pmatrix}1\\ \varepsilon\\ \vdots\\ \varepsilon^m\end{pmatrix}\right)_\ell}{(A_B^{-1}A_{*e})_\ell}$$

$$= -\frac{\ell\text{-th row of } A_B^{-1}(b \mid E_m)}{(A_B^{-1}A_{*e})_\ell}\begin{pmatrix}1\\ \varepsilon\\ \vdots\\ \varepsilon^m\end{pmatrix}$$



## Lexicographic Pivoting

This means you can choose the variable/row $\ell$ for which the vector

$$-\frac{\ell\text{-th row of } A_B^{-1}(b \mid E_m)}{(A_B^{-1}A_{*e})_\ell}$$

is lexicographically minimal.

Of course only including rows with $(A_B^{-1}A_{*e})_\ell < 0$.

This technique guarantees that in each step of the simplex algorithm the objective function will increase.



## 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)
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.



## 5 Seidels LP-algorithm

- Suppose we want to solve $\max\{c^t x \mid Ax \le b; x \ge 0\}$, where $x \in \mathbb{R}^d$ and we have $m$ constraints.
- In the worst-case Simplex runs in time roughly $\mathcal{O}(m(m+d)\binom{m+d}{m}) \approx (m+d)^m$. (better bounds on the running time exist, but will not be discussed here).
- The following algorithm runs in time $\mathcal{O}(m(d+1)!)$.
- It solves $\max\{c^t x \mid Ax \le b; -M \le x_i \le M\}$. Here we added so-called bounding box constraints for the variables $x_i$ to simplify the description.
- We use $\mathcal{H}$ to denote the set of constraints (a set of half-spaces of $\mathbb{R}^d$). $\mathcal{H}$ does not include the bounding box constraints.

## 5 Seidels LP-algorithm

**Algorithm 3** SeidelLP($\mathcal{H}, d$)

**if** $d = 1$ **then** solve 1-dimensional problem and return;
**if** $\mathcal{H} = \emptyset$ **then** solve problem on bounding box and return;
choose random constraint $h \in \mathcal{H}$
$\hat{\mathcal{H}} \leftarrow \mathcal{H} \setminus \{h\}$
$\hat{x}^* \leftarrow$ SeidelLP($\hat{\mathcal{H}}, d$)
**if** $\hat{x}^*$ fulfills $h$ **then**
    return $\hat{x}^*$
// optimal solution fulfills $h$ with equality, i.e., $a_h^t x = b_h$
solve $a_h^t x = b_h$ for some variable $x_\ell$;
eliminate this variable in all constraints from $\hat{\mathcal{H}}$.
Transform box constraints for $x_\ell$ into normal constraints and add them to $\hat{\mathcal{H}}$.
$\hat{x}^* \leftarrow$ SeidelLP($\hat{\mathcal{H}}, d-1$)
add the value of $x_\ell$ to $\hat{x}^*$ and return the solution

## 5 Seidels LP-algorithm

- If $d = 1$ we can solve the 1-dimensional problem in time $\mathcal{O}(m)$.
- If $d > 1$ and $m = 0$ there are only the box constraints. We select $x_j^* = M$ if $c_j \geq 0$, otw. we choose $x_j^* = -M$. This takes time $\mathcal{O}(d)$.
- 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 $\mathcal{O}(dm)$ to eliminate $x_\ell$. 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 increase the objective function.



## 5 Seidels LP-algorithm

This gives the recurrence

$$T(m,d) = \begin{cases} \mathcal{O}(m) & \text{if } d = 1 \\ \mathcal{O}(d) & \text{if } d > 1 \text{ and } m = 0 \\ \mathcal{O}(d) + T(m-1,d) + & \\ \frac{d}{m}(\mathcal{O}(dm) + T(m+1,d-1)) & \text{otw.} \end{cases}$$



## 5 Seidels LP-algorithm

- Let $C$ be the constant in the $\mathcal{O}$-notation.
- We show $T(m,d) \leq Cf(d)\max(1, m-1)$.
- $\boldsymbol{d = 1}$**:**

$$T(m,1) \leq Cm \leq Cf(1)\max(1,m-1) \text{ for } f(1) \geq 2$$

- $\boldsymbol{d > 1; m = 0}$ **:**

$$T(m,d) \leq \mathcal{O}(d) \leq Cd \leq Cf(d)\max(1,m-1) \text{ for } f(d) \geq d$$

## 5 Seidels LP-algorithm

- $d > 1; m = 1:$

$$
\begin{aligned}
T(1,d) &= \mathcal{O}(d) + T(m-1,d) + \frac{d}{m}(\mathcal{O}(dm) + T(m+1,d-1)) \\
&= \mathcal{O}(d) + T(0,d) + \frac{d}{m}(\mathcal{O}(d) + T(2,d-1)) \\
&\le C(d + d + d^2 + df(d-1)\max\{1,1\}) \\
&\le C(3d^2 + df(d-1)) \\
&\le Cf(d)\max\{1,1-1\}
\end{aligned}
$$

if $f(d) \ge df(d-1) + 3d^2$.



## 5 Seidels LP-algorithm

- $d > 1; m = 2:$

$$
\begin{aligned}
T(2,d) &= \mathcal{O}(d) + T(1,d) + \frac{d}{2}(\mathcal{O}(2d) + T(3,d-1)) \\
&\le \mathcal{O}(d) + [\mathcal{O}(d) + T(0,d) + d(\mathcal{O}(d) + T(2,d-1))] \\
&\quad + \frac{d}{2}(2Cd + Cf(d-1)2) \\
&\le 5Cd^2 + Cdf(d-1) + Cf(d-1) \\
&\le Cf(d)\max\{1,2-1\}
\end{aligned}
$$

if $f(d) \ge (d+1)f(d-1) + 5d^2$.



## 5 Seidels LP-algorithm

- $d > 1; m > 2:$

$$
\begin{aligned}
T(m,d) &= \mathcal{O}(d) + T(m-1,d) + \frac{d}{m}(\mathcal{O}(dm) + T(m+1,d-1)) \\
&\le \mathcal{O}(d) + Cf(d)(m-2) + \frac{d}{m}(Cdm + Cf(d-1)m) \\
&\le 2Cd^2 + Cf(d)(m-2) + Cdf(d-1) \\
&\le Cf(d)(m-1) \\
&\le Cf(d)\max\{1,m-1\}
\end{aligned}
$$

if $f(d) \ge df(d-1) + 2d^2$.



## 5 Seidels LP-algorithm

- Define $f(1) = 5 \cdot 1^2$ and $f(d) = (d+1)f(d-1) + 5d^2$ for $d > 1$.

Then

$$
\begin{aligned}
f(d) &\le 5d^2 + (d+1)f(d-1) \\
&= 5d^2 + (d+1)\Big[5(d-1)^2 + df(d-2)\Big] \\
&= 5d^2 + (d+1)\Big[5(d-1)^2 + d\Big[5(d-2)^2 + (d-1)f(d-3)\Big]\Big] \\
&= 5d^2 + 5(d+1)(d-1)^2 + 5(d+1)d(d-2)^2 + \dots \\
&\quad + 5(d+1)d(d-1)\cdot \ldots \cdot 4 \cdot 3 \cdot 1^2 \\
&= 5(d+1)!\left(\frac{d^2}{(d+1)!} + \frac{(d-1)^2}{d!} + \frac{(d-2)^2}{(d-1)!} + \dots + \frac{1^2}{(d-(d-2))!}\right) \\
&= \mathcal{O}((d+1)!)
\end{aligned}
$$

since $\sum_{i\ge 1} \frac{i^2}{(i+1)!}$ is a constant.

## Complexity

**LP Decision Problem (LP decision)**

- 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...

**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

$$\lceil \log_2(|a| + 1) \rceil + 1$$

- Let for a matrix $M$,

$$L(M) = \sum_{i,j} (\lceil \log_2(|m_{ij} + 1|) \rceil + 1)$$

- 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 the input size $L([A|b])$).



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 $0$.



## Size of a Basic Feasible Solution

**Lemma 19**

*Let $A \in \mathbb{Z}^{m \times m}$ be an invertable matrix and let $b \in \mathbb{Z}^m$. Further define $L' = L([A \mid b]) + m \log_2 m$. Then a solution to $Ax = b$ has rational components $x_j$ of the form $\frac{D_j}{D}$, where $|D_j| \leq 2^{L'}$ and $|D| \leq 2^{L'}$.*

**Proof:**
Cramers rules says that we can compute $x_j$ as

$$x_j = \frac{\det(B_j)}{\det(A)}$$

where $B_j$ is the matrix obtained from $A$ by replacing the $j$-th column by the vector $b$.

## Bounding the Determinant

Observe that

$$\begin{aligned}|\det(A)| &= \left|\sum_{\pi\in S_m}\prod_{1\le i\le m}\operatorname{sgn}(\pi)a_{i\pi(i)}\right|\\ &\le \sum_{\pi\in S_m}\prod_{1\le i\le m}|a_{i\pi(i)}|\\ &\le m!\cdot 2^{L([A|b])}\le m^m 2^L\le 2^{L'}\ .\end{aligned}$$

Analogously for $\det(B_j)$.



## Bounding the Determinant

Since we only require a bound polynomial in the input length we could also argue that the largest entry $Z$ in the matrix is at most $2^{L([A|b])}$.

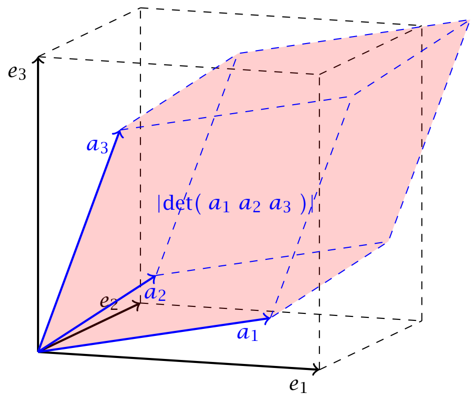
Then, Hadamards inequality gives

$$\begin{aligned}|\det(A)| &\le \prod_{i=1}^{m}\|A_{*i}\|\le\prod_{i=1}^{m}(\sqrt{m}Z)\\ &\le m^{m/2}Z^m\le 2^{mL([A|b])+m\log_2 m}\end{aligned}$$

which also gives an encoding length polynomial in the input length $L([A\mid b])$.



## Hadamards Inequality




Hadamards inequality says that the red volume is smaller than the volume in the black cube (if $\|e_1\|=\|a_1\|$, $\|e_2\|=\|a_2\|$, $\|e_3\|=\|a_3\|$).



This means if $Ax=b$, $x\ge 0$ 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 decision 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

$$\log_2\left(\frac{2n2^{2L'}}{1/2^{L'}}\right) = \mathcal{O}(L') \ ,$$

as the range of the search is at most $-n2^{2L'}, \ldots, n2^{2L'}$ and the distance between two adjacent values is at least $\frac{1}{\det(A)} \geq \frac{1}{2^{L'}}$.

Here we use $L' = L([A \mid b \mid c]) + n\log_2 n$ (the input size plus $n\log_2 n$).



**How do we detect whether the LP is unbounded?**

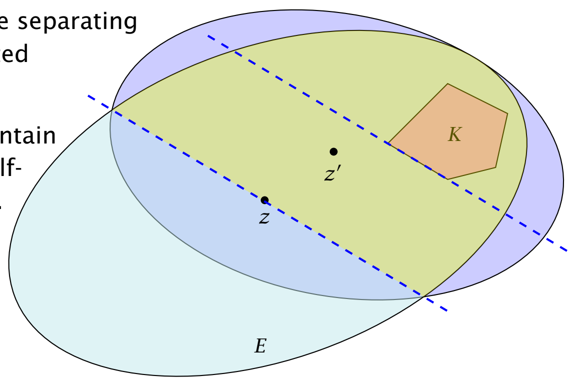
Let $M_{\max} = n2^{2L'}$ be an upper bound on the objective value of a basic feasible solution.

We can add a constraint $c^t x \geq M_{\max} + 1$ and check for feasibility.



## Ellipsoid Method

- 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 20

A mapping $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ with $f(x) = Lx + t$, where $L$ is an invertible matrix is called an affine transformation.



## Definition 21

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 22

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.

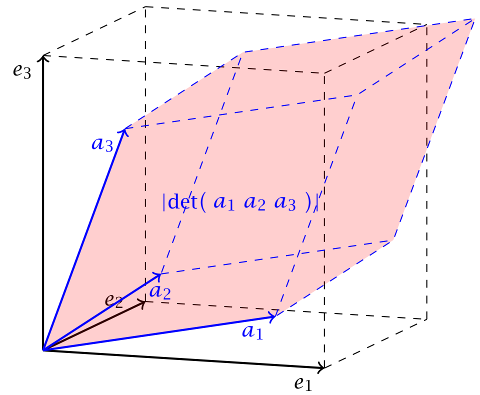


## Lemma 23

*Let $L$ be an affine transformation and $K \subseteq \mathbb{R}^n$. Then*

$$\mathrm{vol}(L(K)) = |\det(L)|\mathrm{vol}(K) \ .$$

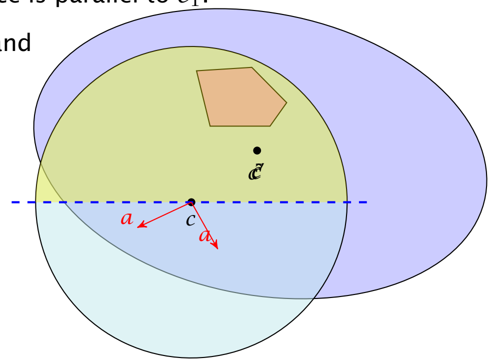
# n-dimensional volume





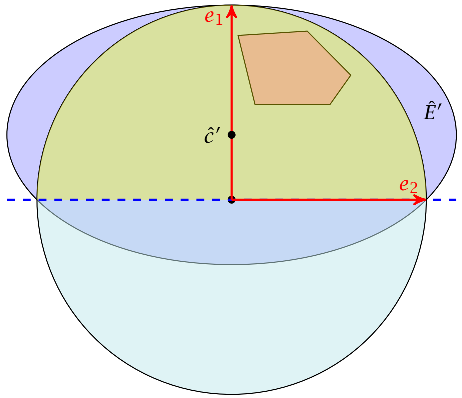

# How to Compute the New Ellipsoid

- Use $f^{-1}$ (recall that $f = Lx + t$ is the transformation function for the Ellipsoid) 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 ellipsoid $\hat{E}'$ 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 $\hat{Q}'^{-1}$ is of the form

$$\hat{Q}'^{-1} = \begin{pmatrix} \frac{1}{a^2} & 0 & \ldots & 0 \\ 0 & \frac{1}{b^2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & \frac{1}{b^2} \end{pmatrix}$$

## 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

$$\frac{1}{b^2} = 1 - \frac{t^2}{a^2} = 1 - \frac{t^2}{(1-t)^2} = \frac{1-2t}{(1-t)^2}$$



## The Easy Case

- We want to choose $t$ such that the volume of $\hat{E}'$ is minimal.

$$\mathrm{vol}(\hat{E}') = \mathrm{vol}(B(0,1)) \cdot |\det(\hat{L})| \ ,$$

where $\hat{Q}' = \hat{L}'^t \hat{L}'$.

- This gives

$$\hat{L}'^{-1} = \begin{pmatrix} \frac{1}{a} & 0 & \dots & 0 \\ 0 & \frac{1}{b} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{1}{b} \end{pmatrix} \text{ and } \hat{L}' = \begin{pmatrix} a & 0 & \dots & 0 \\ 0 & b & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b \end{pmatrix}$$



## 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

$$
\begin{aligned}
\frac{\mathrm{d}\,\mathrm{vol}(\hat{E})}{\mathrm{d}t} &= \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{(1-t)^n}{(\sqrt{1-2t})^{n-1}}\right)\\
&= \frac{1}{N^2}\cdot\Bigg((-1)\cdot n(1-t)^{n-1}\cdot(\sqrt{1-2t})^{n-1}\\
&\qquad -(n-1)(\sqrt{1-2t})^{n-2}\cdot\frac{1}{2\sqrt{1-2t}}\cdot(-2)\cdot(1-t)^n\Bigg)\\
&= \frac{1}{N^2}\cdot(\sqrt{1-2t})^{n-3}\cdot(1-t)^{n-1}\\
&\qquad\cdot\Big((n-1)(1-t)-n(1-2t)\Big)\\
&= \frac{1}{N^2}\cdot(\sqrt{1-2t})^{n-3}\cdot(1-t)^{n-1}\cdot\Big((n+1)t-1\Big)
\end{aligned}
$$



## The Easy Case

- We obtain the minimum for $t=\frac{1}{n+1}$.
- For this value we obtain

$$
a=\sqrt{1-t}=\frac{n}{n+1}\quad\text{and}\quad b=\frac{1-t}{\sqrt{1-2t}}=\frac{n}{\sqrt{n^2-1}}
$$



## 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:

$$
\begin{aligned}
\gamma_n^2 &= \left(\frac{n}{n+1}\right)^2\left(\frac{n^2}{n^2-1}\right)^{n-1}\\
&= \left(1-\frac{1}{n+1}\right)^2\left(1+\frac{1}{(n-1)(n+1)}\right)^{n-1}\\
&\le e^{-2\frac{1}{n+1}}\cdot e^{\frac{1}{n+1}}\\
&= e^{-\frac{1}{n+1}}
\end{aligned}
$$

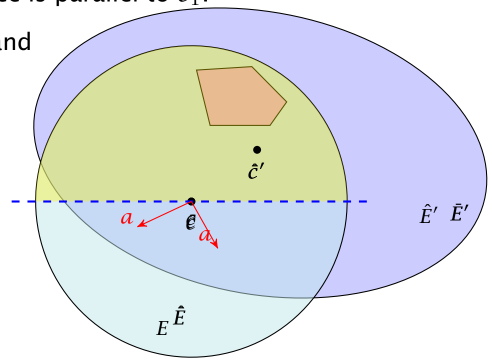
where we used $(1+x)^a\le e^{ax}$ for $x\in\mathbb{R}$ and $a>0$.

This gives $\gamma_n\le e^{-\frac{1}{2(n+1)}}$.



## How to Compute the New Ellipsoid

- 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 $E$.



$$e^{-\frac{1}{2(n+1)}} \geq \frac{\mathrm{vol}(\hat{E}')}{\mathrm{vol}(B(0,1))} = \frac{\mathrm{vol}(\hat{E}')}{\mathrm{vol}(\hat{E})} = \frac{\mathrm{vol}(R(\hat{E}'))}{\mathrm{vol}(R(\hat{E}))}$$
$$= \frac{\mathrm{vol}(\bar{E}')}{\mathrm{vol}(\bar{E})} = \frac{\mathrm{vol}(f(\bar{E}'))}{\mathrm{vol}(f(\bar{E}))} = \frac{\mathrm{vol}(E')}{\mathrm{vol}(E)}$$



# The Ellipsoid Algorithm

### How to Compute The New Parameters?

The transformation function of the ellipsoid: $f(x) = Lx + c$;

The halfspace: $H = \{x \mid a^t(x - c) \leq 0\}$;

$$\begin{aligned} f^{-1}(H) &= \{f^{-1}(x) \mid a^t(x - c) \leq 0\} \\ &= \{f^{-1}(f(y)) \mid a^t(f(y)) - c) \leq 0\} \\ &= \{y \mid a^t(f(y)) - c) \leq 0\} \\ &= \{y \mid a^t(Ly + c) - c) \leq 0\} \\ &= \{y \mid (a^t L)y \leq 0\} \end{aligned}$$

This means $\bar{a} = L^t a$.



# The Ellipsoid Algorithm

$$R^{-1}\Big(\frac{L^t a}{\|L^t a\|}\Big) = -e_1 \quad \Rightarrow \quad -\frac{L^t a}{\|L^t a\|} = R \cdot e_1$$

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 ellipsoids centered in the origin.

Note that

$$\hat{Q}' = \frac{n^2}{n^2 - 1}\Big(I - \frac{2}{n+1} e_1 e_1^t\Big)$$

## 6 The Ellipsoid Algorithm

$$
\begin{aligned}
\bar{E}' &= R(\hat{E}') \\
&= \{R(x) \mid x^t \hat{Q}'^{-1} x \le 1\} \\
&= \{y \mid (R^{-1}y)^t \hat{Q}'^{-1} R^{-1} y \le 1\} \\
&= \{y \mid (y^t (R^t)^{-1} \hat{Q}'^{-1} R^{-1} y \le 1\} \\
&= \{y \mid (y^t (\underbrace{R\hat{Q}'R^t}_{\bar{Q}'})^{-1} y \le 1\}
\end{aligned}
$$



## 6 The Ellipsoid Algorithm

Hence,

$$
\begin{aligned}
\bar{Q}' &= R\hat{Q}'R^t \\
&= R \cdot \frac{n^2}{n^2-1}\Big(I - \frac{2}{n+1} e_1 e_1^t\Big) \cdot R^t \\
&= \frac{n^2}{n^2-1}\Big(R \cdot R^t - \frac{2}{n+1}(Re_1)(Re_1)^t\Big) \\
&= \frac{n^2}{n^2-1}\Big(I - \frac{2}{n+1}\frac{L^t a a^t L}{\|L^t a\|^2}\Big)
\end{aligned}
$$



## 6 The Ellipsoid Algorithm

Hence,

$$
\begin{aligned}
\bar{Q}' &= R\hat{Q}'R^t \\
&= R \cdot \frac{n^2}{n^2-1}\Big(I - \frac{2}{n+1} e_1 e_1^t\Big) \cdot R^t \\
&= \frac{n^2}{n^2-1}\Big(R \cdot R^t - \frac{2}{n+1}(Re_1)(Re_1)^t\Big) \\
&= \frac{n^2}{n^2-1}\Big(I - \frac{2}{n+1}\frac{L^t a a^t L}{\|L^t a\|^2}\Big) \\
&= \frac{n^2}{n^2-1}\Big(I - \frac{2}{n+1}\frac{L^t a a^t L}{a^t Q a}\Big)
\end{aligned}
$$



## 6 The Ellipsoid Algorithm

$$
\begin{aligned}
E' &= L(\bar{E}') \\
&= \{L(x) \mid x^t \bar{Q}'^{-1} x \le 1\} \\
&= \{y \mid (L^{-1}y)^t \bar{Q}'^{-1} L^{-1} y \le 1\} \\
&= \{y \mid (y^t (L^t)^{-1} \bar{Q}'^{-1} L^{-1} y \le 1\} \\
&= \{y \mid (y^t (\underbrace{L\bar{Q}'L^t}_{Q'})^{-1} y \le 1\}
\end{aligned}
$$

## 6 The Ellipsoid Algorithm

Hence,

$$\begin{aligned} Q' &= L\bar{Q}'L^t \\ &= L \cdot \frac{n^2}{n^2-1}\Big(I - \frac{2}{n+1}\frac{L^t a a^t L}{a^t Q a}\Big) \cdot L^t \\ &= \frac{n^2}{n^2-1}\Big(Q - \frac{2}{n+1}\frac{Q a a^t Q}{a^t Q a}\Big) \end{aligned}$$



## Repeat: Size of basic solutions

**Lemma 24**

*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 $A$. Then every entry $x_i$ in a basic solution fulfills $|x_i| = \frac{D_i}{D}$ with $D_j, D \le 2^{2n\langle a_{\max}\rangle + n\log_2 n}$.*

In the following we use $\delta := 2^{2n\langle a_{\max}\rangle + n\log_2 n}$.



## Repeat: Size of basic solutions

**Proof:** Let $\bar{A} = (A| - A|I_m)$ Then the determinant of the matrices $\bar{A}_B$ and $\bar{B}_j$ can become at most

$$\det(\bar{A}_B) \le \|\vec{\ell}_{\max}\|^{2n} \le 2^{2n\langle a_{\max}\rangle + n\log_2 n} ,$$

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 $\bar{A}_B$ will not increase the determinant. Only the at most $2n$ columns from the matrices $A$ and $-A$ that $\bar{A}$ consists of will contribute.



## How do we find the first ellipsoid?

For feasibility checking we can assume that the polytop $P$ is bounded.

In this case every entry $x_i$ in a basic solution fulfills $|x_i| \le \delta$.

Hence, $P$ is contained in the cube $-\delta \le x_i \le \delta$.

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 $(n\delta)^n B(0,1)$.

## 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 $\lambda = \delta + 1$.



**Lemma 25**
*$P_\lambda$ is feasible if and only if $P$ is feasible.*

$\Leftarrow$: obvious!



$\Rightarrow$:

Consider the polytops

$$\bar{P} = \left\{x \mid (A| - A|I_m)x = b; x \ge 0\right\}$$

and

$$\bar{P}_\lambda = \left\{x \mid (A| - A|I_m)x = b + \frac{1}{\lambda}\begin{pmatrix}1\\ \vdots \\ 1\end{pmatrix}; x \ge 0\right\} .$$

$P$ is feasible if and only if $\bar{P}$ is feasible, and $P_\lambda$ feasible if and only if $\bar{P}_\lambda$ feasible.

$\bar{P}_\lambda$ is bounded since $P_\lambda$ and $P$ are bounded. Use $\bar{A} = (A| - A|I_m)$.



$\bar{P}_\lambda$ feasible implies that there is a basic feasible solution represented by

$$x_B = \bar{A}_B^{-1}b + \frac{1}{\lambda}\bar{A}_B^{-1}\begin{pmatrix}1\\ \vdots \\ 1\end{pmatrix}$$

where $\bar{A} = (A| - A|I_m)$. (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

$$(\bar{A}_B^{-1}b)_i < 0 \le (\bar{A}_B^{-1}b)_i + \frac{1}{\lambda}(\bar{A}_B^{-1}\vec{1})_i$$

But Cramers rule gives that $(\bar{A}_B^{-1}b)_i < 0$ implies that $(\bar{A}_B^{-1}b)_i \le -\frac{1}{\det(\bar{A}_B)}$ and $(\bar{A}_B^{-1}\vec{1})_i \le \det(\bar{B}_j)$, where $B_j$ is obtained by replacing the $j$-th column of $\bar{A}_B$ by $b$.

Then the determinant of the matrices $\bar{A}_B$ and $\bar{B}_j$ can become at most $\delta$.

Since, we chose $\lambda = \delta + 1$ this gives a contradiction.



**Lemma 26**

*If $P_\lambda$ is feasible then it contains a ball of radius $r := 1/\lambda \ge 1/(2\delta)$. This has a volume of at least $\frac{1}{(2\delta)^n} \cdot \mathrm{vol}(B(0,1))$.*



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,

$$\begin{aligned} i &> 2(n+1)\ln\left(\frac{\mathrm{vol}(B(0,R))}{\mathrm{vol}(B(0,r))}\right) \\ &= 2(n+1)\ln\left(n^n\delta^n \cdot 2^n\delta^n\right) \\ &\le 2n\ln(\delta) + n\ln(n) \\ &\le 4n^2\langle a_{\max}\rangle + 3n^2\log_2(n) \end{aligned}$$



# Ellipsoid Algorithm

**Input:** point $c \in \mathbb{R}^n$, radii $R$ and $r$, convex set $K \subseteq \mathbb{R}^n$
**Output:** point $x \in K$

- check whether $c \in K$; if yes **output** $\boldsymbol{c}$
- otherwise choose a violated hyperplane $a$;

$$c' = c - \frac{1}{n+1}\frac{Qa}{\sqrt{a^tQa}}$$

$$Q' = \frac{n^2}{n^2-1}\left(Q - \frac{2}{n+1}\frac{Qaa^tQ}{a^tQa}\right)$$

- if $\det(Q') \le \sqrt{r^n}$ **output fail**
- **repeat**

# 7 Karmarkar's 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 $0$.



# 7 Karmarkar's Algorithm

Suppose you start with $\max\{c^t x \mid Ax \leq 0; x \geq 0\}$.

- Multiply $c$ by $-1$ and do a minimization. $\Rightarrow$ minimization problem
- We can check for feasibility by using the two phase algorithm (first optimizing a different feasible LP; if the solution is non-zero the original LP is infeasible). Therefore, we can assume that the 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 $x_\ell$, $x'_\ell$ (both restricted to be positive) and the constraint $\sum_i x_i = 1$. $\Rightarrow$ solution lies in simplex
- Add $-(\sum_i x_i)b_i = -b_i$ to every constraint. $\Rightarrow$ vector $b$ becomes $0$
- If $A$ does not have full column rank we can delete constraints (or conclude that the LP is infeasible). $A$ has full row rank



# 7 Karmarkar's Algorithm

The algorithm computes (strictly) feasible interior points $x^0 = \frac{e}{n}, x^1, x^2, \ldots$ with

$$c^t x^k \leq e^{-\frac{k}{5n}} c^t x^0$$

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.



# 7 Karmarkar's 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). $\hat{x}$ is the point you reached.
3. Do a backtransformation to transform $\hat{x}$ into your new point $x'$.

## The Transformation

Let $\bar{Y} = \mathrm{diag}(\bar{x})$ the diagonal matrix with entries $\bar{x}$ on the diagonal.

Define

$$F_{\bar{x}} : x \mapsto \frac{\bar{Y}^{-1}x}{e^t\bar{Y}^{-1}x} .$$

The inverse function is

$$F_{\bar{x}}^{-1} : \hat{x} \mapsto \frac{\bar{Y}\hat{x}}{e^t\bar{Y}\hat{x}} .$$



## The Transformation

- $F_{\bar{x}}^{-1}$ 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.



## 7 Karmarkar's Algorithm

After the transformation we have the problem

$$\min\{c^t F_{\bar{x}}(x) \mid AF_{\bar{x}}(x); x \in \Delta\} = \min\left\{\frac{c^t\bar{Y}x}{e^t\bar{Y}x} \mid \frac{A\bar{Y}x}{e^t\bar{Y}x}; x \in \Delta\right\}$$

Since the optimum solution is $0$ this is the same as

$$\min\{\hat{c}^t x \mid \hat{A}x = 0, x \in \Delta\}$$

with $\hat{c} = \bar{Y}^t c = \bar{Y}c$ and $\hat{A} = A\bar{Y}$.



## 7 Karmarkar's Algorithm

When computing $\hat{x}$ we do not want to leave the simplex or touch its boundary.

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

$$B\left(\frac{e}{n}, r\right) \cap \left\{x \mid e^t x = 1\right\} \subseteq \Delta.$$

# 7 Karmarkar's Algorithm

This holds for $r = \|\frac{e}{n} - (e - e_1)\frac{1}{n-1}\|$.

This gives $r = \frac{1}{\sqrt{n(n-1)}}$.

Now we consider the problem

$$\min\{\hat{c}^t x \mid \hat{A}x = 0, x \in B(e/n, r) \cap \Delta\}$$



# 7 Karmarkar's Algorithm

Ideally we would like to go in direction of $-\hat{c}$ (starting from the center of the ellipsoid).

However, doing this may violate constraints $\hat{A}x = 0$ or the constraint $x \in \Delta$.

Therefore we first project $\hat{c}$ on the nullspace of

$$B = \begin{pmatrix} \hat{A} \\ e \end{pmatrix}$$

We use

$$P = I - B^t(BB^t)^{-1}B$$

Then

$$\hat{d} = P\hat{c}$$

is the required projection.



# 7 Karmarkar's Algorithm

We get the new point

$$\hat{x}(\rho) = \frac{e}{n} + \rho\frac{\hat{d}}{\|d\|}$$

for $\rho < r$.

Choose $\rho = \frac{\alpha}{n} < \alpha r$ with $\alpha = 1/3$.



# 8 Karmarkar's 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 $0$.

## 8 Karmarkar's Algorithm

Suppose you start with $\max\{c^t x \mid Ax \le 0; x \ge 0\}$.

- Multiply $c$ by $-1$ and do a minimization. $\Rightarrow$ minimization problem
- We can check for feasibility by using the two phase algorithm (first optimizing a different feasible LP; if the solution is non-zero the original LP is infeasible). Therefore, we can assume that the 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 $x_\ell$, $x'_\ell$ (both restricted to be positive) and the constraint $\sum_i x_i = 1$. $\Rightarrow$ solution lies in simplex
- Add $-(\sum_i x_i)b_i = -b_i$ to every constraint. $\Rightarrow$ vector $b$ becomes $0$
- If $A$ does not have full column rank we can delete constraints (or conclude that the LP is infeasible). $A$ has full row rank



## 8 Karmarkar's Algorithm

The algorithm computes (strictly) feasible interior points $\bar{x}^{(0)} = \frac{e}{n}, x^{(1)}, x^{(2)}, \dots$ with

$$c^t x^k \le 2^{-\Theta(L)} c^t x^0$$

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.



## 8 Karmarkar's 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). $\hat{x}$ is the point you reached.
3. Do a backtransformation to transform $\hat{x}$ into your new point $x'$.



## 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

$$F_{\bar{x}}^{-1} : \hat{x} \mapsto \frac{\bar{Y}\hat{x}}{e^t \bar{Y}\hat{x}} \enspace .$$

## 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.



## 8 Karmarkar's Algorithm

After the transformation we have the problem

$$\min\{c^t F_{\bar{x}}^{-1}(x) \mid AF_{\bar{x}}^{-1}(x); x \in \Delta\} = \min\left\{\frac{c^t \bar{Y} x}{e^t \bar{Y} x} \mid \frac{A\bar{Y}x}{e^t \bar{Y} x}; x \in \Delta\right\}$$

This holds since the back-transformation "reaches" every point in $\Delta$ (i.e. $F^{-1}(\Delta) = \Delta$).

Since the optimum solution is $0$ this problem is the same as

$$\min\{\hat{c}^t x \mid \hat{A}x = 0, x \in \Delta\}$$

with $\hat{c} = \bar{Y}^t c = \bar{Y} c$ and $\hat{A} = A\bar{Y}$.



## 8 Karmarkar's Algorithm

When computing $\hat{x}$ we do not want to leave the simplex or touch its boundary.

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

$$B\left(\frac{e}{n}, r\right) \cap \left\{x \mid e^t x = 1\right\} \subseteq \Delta.$$



## 8 Karmarkar's 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 $\Delta$.)

This gives $r = \frac{1}{\sqrt{n(n-1)}}$.

Now we consider the problem

$$\min\{\hat{c}^t x \mid \hat{A}x = 0, x \in B(e/n, r) \cap \Delta\}$$

# 8 Karmarkar's 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 $\hat{A}x = 0$ or the constraint $x \in \Delta$.

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.



# 8 Karmarkar's Algorithm

We get the new point

$$\hat{x}(\rho) = \frac{e}{n} - \rho\frac{\hat{d}}{\|d\|}$$

for $\rho < r$.

Choose $\rho = \alpha r$ with $\alpha = 1/4$.



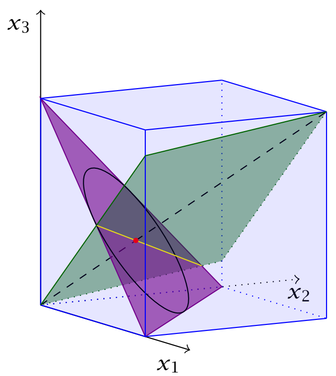
# 8 Karmarkar's Algorithm

**Iteration of Karmarkar's algorithm:**

- Current solution $\bar{x}$. $\bar{Y} := \mathrm{diag}(\bar{x}_1, \dots, \bar{x}_n)$.
- Transform the 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
  $$d = (I - B^t(BB^t)^{-1}B)\hat{c} \ ,$$
  where $B = \begin{pmatrix} \hat{A} \\ e^t \end{pmatrix}$.
- Set
  $$\hat{x} = \frac{e}{n} - \rho\frac{d}{\|d\|} \ ,$$
  with $\rho = \alpha r$ with $\alpha = 1/4$ and $r = 1/\sqrt{n(n-1)}$.
- Compute $\bar{x}_{\text{new}} = F_{\bar{x}}^{-1}(\hat{x})$.



# The Simplex



**Lemma 27**
*The new point $\hat{x}$ in the transformed space is the point that minimizes the cost $\hat{c}^t x$ among all feasible points in $B(\frac{e}{n}, \rho)$.*



**Proof:** Let $z$ be another feasible point in $B(\frac{e}{n}, \rho)$.

As $\hat{A}z = 0$, $\hat{A}\hat{x} = 0$, $e^t z = 1$, $e^t \hat{x} = 1$ we have

$$B(\hat{x} - z) = 0 \; .$$

Further,

$$\begin{aligned}(\hat{c} - d)^t &= (\hat{c} - P\hat{c})^t \\ &= (B^t(BB^t)^{-1}B\hat{c})^t \\ &= \hat{c}^t B^t (BB^t)^{-1} B\end{aligned}$$

Hence, we get

$$(\hat{c} - d)^t(\hat{x} - z) = 0 \text{ or } \hat{c}^t(\hat{x} - z) = d^t(\hat{x} - z)$$

which means that the cost-difference between $\hat{x}$ and $z$ is the same measured w.r.t. the cost-vector $\hat{c}$ or the projected cost-vector $d$.



But

$$\frac{d^t}{\|d\|}(\hat{x} - z) = \frac{d^t}{\|d\|}\left(\frac{e}{n} - \rho\frac{d}{\|d\|} - z\right) = \frac{d^t}{\|d\|}\left(\frac{e}{n} - z\right) - \rho < 0$$

as $\frac{e}{n} - z$ is a vector of length at most $\rho$.

This gives $d(\hat{x} - z) \leq 0$ and therefore $\hat{c}\hat{x} \leq \hat{c}z$.



In order to measure the progress of the algorithm we introduce a potential function $f$:

$$f(x) = \sum_j \ln(\frac{c^t x}{x_j}) = n\ln(c^t x) - \sum_j \ln(x_j) \; .$$

- 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 $z$ in the transformed space we use the potential function

$$\hat{f}(z) := f(F_{\bar{x}}^{-1}(z)) = f(\frac{\bar{Y}z}{e^t\bar{Y}z}) = f(\bar{Y}z)$$
$$= \sum_j \ln(\frac{c^t\bar{Y}z}{\bar{x}_j z_j}) = \sum_j \ln(\frac{\hat{c}^t z}{z_j}) - \sum_j \ln \bar{x}_j$$

**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

$$\hat{f}(\hat{x}) \leq \hat{f}(\frac{e}{n}) - \delta ,$$

where $\delta$ is a constant.

This gives

$$f(\bar{x}_{\text{new}}) \leq f(\bar{x}) - \delta .$$



Lemma 28
*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.



Let $z^*$ be the feasible point in the transformed space where $\hat{c}x$ is minimized. (Note that in contrast $\hat{x}$ is the point in the intersection of the feasible region and $B(\frac{e}{n},\rho)$ that minimizes this function; in general $z^* \neq \hat{x}$)

$z^*$ 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,

$$\hat{c}^t z = (1-\lambda)\hat{c}^t \frac{e}{n} + \lambda \hat{c}^t z^*$$

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

$$\begin{aligned}\hat{f}(\frac{e}{n}) - \hat{f}(z) &= \sum_j \ln(\frac{\hat{c}^t \frac{e}{n}}{\frac{1}{n}}) - \sum_j \ln(\frac{\hat{c}^t z}{z_j}) \\ &= \sum_j \ln(\frac{\hat{c}^t \frac{e}{n}}{\hat{c}^t z} \cdot \frac{z_j}{\frac{1}{n}}) \\ &= \sum_j \ln(\frac{n}{1-\lambda} z_j) \\ &= \sum_j \ln(\frac{n}{1-\lambda}((1-\lambda)\frac{1}{n} + \lambda z_j^*)) \\ &= \sum_j \ln(1 + \frac{n\lambda}{1-\lambda} z_j^*)\end{aligned}$$



We can use the fact that for non-negative $s_i$

$$\sum_i \ln(1+s_i) \geq \ln(1 + \sum_i s_i)$$

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 $\lambda$:

$$\alpha r = \rho = \|z - e/n\| = \|\lambda(z^* - e/n)\| \leq \lambda R$$

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

$$\lambda \geq \alpha/(n-1)$$

Then

$$1 + n\frac{\lambda}{1-\lambda} \geq 1 + \frac{n\alpha}{n-\alpha-1} \geq 1 + \alpha$$

This gives the lemma.

## Lemma 29

*If we choose* $\alpha = 1/4$ *and* $n \geq 4$ *in Karmarkar's algorithm the point* $\hat{x}$ *satisfies*

$$\hat{f}(\hat{x}) \leq \hat{f}(\frac{e}{n}) - \delta$$

*with* $\delta = 1/10$.



**Proof**:

Define

$$\begin{aligned} g(x) &= n \ln \frac{\hat{c}^t x}{\hat{c}^t \frac{e}{n}} \\ &= n(\ln \hat{c}^t x - \ln \hat{c}^t \frac{e}{n}) \ . \end{aligned}$$

This is the change in the cost part of the potential function when going from the center $\frac{e}{n}$ to the point $x$ in the transformed space.



We want to derive a lower bound on

$$\begin{aligned} \hat{f}(\frac{e}{n}) - \hat{f}(\hat{x}) = [\hat{f}(\frac{e}{n}) - \hat{f}(z)] & \\ + [\hat{f}(z) - (\hat{f}(\frac{e}{n}) + g(z))] & \\ - [\hat{f}(\hat{x}) - (\hat{f}(\frac{e}{n}) + g(\hat{x}))] & \\ + [g(z) - g(\hat{x})] & \end{aligned}$$

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

$$[g(z) - g(\hat{x})] \geq 0$$

since $\hat{x}$ is the point with minimum cost in the ball, and $g$ is monotonically increasing with cost.

For a point in the ball we have

$$\hat{f}(w) - (\hat{f}(\frac{e}{n}) + g(w)) = -\sum_j \ln \frac{w_j}{\frac{1}{n}}$$

(The increase in penalty when going from $\frac{e}{n}$ to $w$).

This is at most $\frac{\beta^2}{2(1-\beta)}$ with $\beta = n\alpha r$.
Hence,

$$\hat{f}(\frac{e}{n}) - \hat{f}(\hat{x}) \geq \ln(1+\alpha) - \frac{\beta^2}{(1-\beta)} \quad .$$



**Lemma 30**
*For* $|x| \leq \beta < 1$

$$|\ln(1+x) - x| \leq \frac{x^2}{2(1-\beta)} \quad .$$



This gives for $w \in B(\frac{e}{n}, \rho)$

$$\begin{aligned}
\left|\sum_j \ln \frac{w_j}{1/n}\right| &= \left|\sum_j \ln(\frac{1/n + (w_j - 1/n)}{1/n}) - \sum_j n(w_j - \frac{1}{n})\right| \\
&= \left|\sum_j \left[\ln(1 + \overbrace{n(w_j - 1/n)}^{\leq n\alpha r < 1}) - n(w_j - \frac{1}{n})\right]\right| \\
&\leq \sum_j \frac{n^2(w_j - 1/n)^2}{2(1 - \alpha n r)} \\
&\leq \frac{(\alpha n r)^2}{2(1 - \alpha n r)}
\end{aligned}$$



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 $\alpha = 1/4$.

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 $\mathcal{O}(n^3)$.



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 31

An $\alpha$-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 $\alpha$ of the value of an optimal solution.



**Minimization Problem:**

Let $\mathcal{I}$ denote the set of problem instances, and let for a given instance $I \in \mathcal{I}$, $\mathcal{F}(I)$ denote the set of feasible solutions. Further let $\mathrm{cost}(F)$ denote the cost of a feasible solution $F \in \mathcal{F}$.

Let for an algorithm $A$ and instance $I \in \mathcal{I}$, $A(I) \in \mathcal{F}(I)$ denote the feasible solution computed by $A$. Then $A$ is an approximation algorithm with approximation guarantee $\alpha \geq 1$ if

$$\forall I \in \mathcal{I} : \mathrm{cost}(A(I)) \leq \alpha \cdot \min_{F \in \mathcal{F}(I)} \{\mathrm{cost}(F)\} = \alpha \cdot \mathrm{OPT}(I)$$

**Maximization Problem:**

Let $\mathcal{I}$ denote the set of problem instances, and let for a given instance $I \in \mathcal{I}$, $\mathcal{F}(I)$ denote the set of feasible solutions. Further let $\mathrm{profit}(F)$ denote the profit of a feasible solution $F \in \mathcal{F}$.

Let for an algorithm $A$ and instance $I \in \mathcal{I}$, $A(I) \in \mathcal{F}(I)$ denote the feasible solution computed by $A$. Then $A$ is an approximation algorithm with approximation guarantee $\alpha \le 1$ if

$$\forall I \in \mathcal{I} : \mathrm{cost}(A(I)) \ge \alpha \cdot \max_{F \in \mathcal{F}(I)} \{\mathrm{profit}(F)\} = \alpha \cdot \mathrm{OPT}(I)$$



**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.



**What can we hope for?**

### Definition 32

A polynomial-time approximation scheme (PTAS) is a family of algorithms $\{A_\epsilon\}$, such that $A_\epsilon$ is a $(1+\epsilon)$-approximation algorithms (for minimization problems) or a $(1-\epsilon)$-approximation algorithms (for maximization problems).

Many NP-complete problems have polynomial time approximation schemes.



**There are difficult problems!**

The class $\mathrm{MAX\,SNP}$ (which we do not define) contains optimization problems like maximum cut or maximum satisfiability.

### Theorem 33

*For any $\mathrm{MAX\,SNP}$-hard problem, there does not exist a polynomial-time approximation scheme, unless $\mathrm{P} = \mathrm{NP}$.*

**There are really difficult problems!**

### Theorem 34

*For any constant $\epsilon > 0$ there does not exist an $\Omega(n^{\epsilon-1})$-approximation algorithm for the maximum clique problem on a given graph $G$ with $n$ nodes unless $\mathrm{P} = \mathrm{NP}$.*

Note that a $1/n$-approximation is trivial.



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 35

An Integer Linear Program or Integer Program is a Linear Program in which all variables are required to be integral.

### Definition 36

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!**

## 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

$$\sum_{i \in I} w_i \quad \text{is minimized.}$$



## IP-Formulation of Set Cover

$$\begin{array}{llrcl} \min & & \sum_i w_i x_i & & \\ \text{s.t.} & \forall u \in U & \sum_{i:u \in S_i} x_i & \geq & 1 \\ & \forall i \in \{1, \dots, k\} & x_i & \geq & 0 \\ & \forall i \in \{1, \dots, k\} & x_i & \text{integral} & \end{array}$$



## IP-Formulation of Set Cover

$$\begin{array}{llrcl} \min & & \sum_i w_i x_i & & \\ \text{s.t.} & \forall u \in U & \sum_{i:u \in S_i} x_i & \geq & 1 \\ & \forall i \in \{1, \dots, k\} & x_i & \in & \{0, 1\} \end{array}$$



## 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

$$\begin{array}{lllcl}
\min & & \sum_{v\in V} w_v x_v & & \\
\text{s.t.} & \forall e=(i,j)\in E & x_i + x_j & \geq & 1 \\
 & \forall v \in V & x_v & \in & \{0,1\}
\end{array}$$



## 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.

$$\begin{array}{lllcl}
\max & & \sum_{e\in E} x_e & & \\
\text{s.t.} & \forall v\in V & \sum_{e:v\in e} x_e & \leq & 1 \\
 & \forall e \in E & x_e & \in & \{0,1\}
\end{array}$$



## 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.

$$\begin{array}{lllcl}
\max & & \sum_{v\in V} w_v x_v & & \\
\text{s.t.} & \forall e=(i,j)\in E & x_i + x_j & \leq & 1 \\
 & \forall v \in V & x_v & \in & \{0,1\}
\end{array}$$



## 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.

$$\begin{array}{lllcl}
\max & & \sum_{i=1}^{n} p_i x_i & & \\
\text{s.t.} & & \sum_{i=1}^{n} w_i x_i & \leq & K \\
 & \forall i \in \{1,\dots,n\} & x_i & \in & \{0,1\}
\end{array}$$

## Facility Location

Given a set $L$ of (possible) locations for placing facilities and a set $C$ of customers together with cost functions $s : C \times L \to \mathbb{R}^+$ and $o : L \to \mathbb{R}^+$ find a set of facility locations $F$ together with an assignment $\phi : C \to F$ of customers to open facilities such that

$$\sum_{f \in F} o(f) + \sum_{c} s(c, \phi(c))$$

is minimized.

In the metric facility location problem we have

$$s(c,f) \le s(c,f') + s(c',f) + s(c',f') \ .$$



## Relaxations

**Definition 37**
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 $x_i\{0,1\}$.



By solving a relaxation we obtain an upper bound for a maximization problem and a lower bound for a minimization problem.



## 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:**

$$\begin{array}{lrlll} \min & & \sum_{i=1}^{k} w_i x_i & & \\ \text{s.t.} & \forall u \in U & \sum_{i:u \in S_i} x_i & \ge & 1 \\ & \forall i \in \{1,\dots,k\} & x_i & \in & [0,1] \end{array}$$

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.

**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 38
*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 $f \cdot \mathrm{OPT}$.

$$\begin{aligned}\sum_{i \in I} w_i &\leq \sum_{i=1}^{k} w_i (f \cdot x_i) \\ &= f \cdot \mathrm{cost}(x) \\ &\leq f \cdot \mathrm{OPT} \ .\end{aligned}$$



## Technique 2: Rounding the Dual Solution.

**The dual of the LP-relaxation:**

$$\begin{array}{llrcl} \max & & \sum_{u \in U} y_u & & \\ \text{s.t.} & \forall i \in \{1, \dots, k\} & \sum_{u:u \in S_i} y_u & \leq & w_i \\ & & y_u & \geq & 0 \end{array}$$

## 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 39**
*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:**

$$\begin{aligned}\sum_{i\in I} w_i &= \sum_{i\in I}\sum_{u:u\in S_i} y_u\\ &= \sum_u |\{i \in I : u \in S_i\}| \cdot y_u\\ &\le \sum_u f_u y_u\\ &\le f\sum_u y_u\\ &\le f \cdot \mathrm{OPT}\end{aligned}$$



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 $I$.

# 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 all 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 4** PrimalDual

$y \leftarrow 0$
$I \leftarrow \emptyset$
**while** exists $u \notin \bigcup_{i\in I} S_i$ **do**
  increase dual variable $y_i$ until constraint for some new set $S_\ell$ becomes tight
  $I \leftarrow I \cup \{\ell\}$



# Technique 4: The Greedy Algorithm

**Algorithm 5** Greedy

$I \leftarrow \emptyset$
$\hat{S}_j \leftarrow S_j$ for all $j$
**while** $I$ not a set cover **do**
  $\ell \leftarrow \arg\min_{j:\hat{S}_j \neq 0} \frac{w_j}{|\hat{S}_j|}$
  $I \leftarrow I \cup \{\ell\}$
  $\hat{S}_j \leftarrow \hat{S}_j - S_\ell$ for all $j$



# Technique 4: The Greedy Algorithm

**Lemma 40**

*Given positive numbers $a_1, \dots, a_k$ and $b_1, \dots, b_k$ then*

$$\min_i \frac{a_i}{b_i} \leq \frac{\sum_i a_i}{\sum_i b_i} \leq \max_i \frac{a_i}{b_i}$$

## Technique 4: The Greedy Algorithm

Let $n_\ell$ denote the number of elements that remain at the beginning of iteration $\ell$. $n_1 = n = |U|$ and $n_{s+1} = 0$ if we need $s$ iterations.

In the $\ell$-th iteration

$$\min_j \frac{w_j}{|\hat{S}_j|} \le \frac{\mathrm{OPT}}{n_\ell} \quad .$$

since an optimal algorithm can cover the remaining $n_\ell$ 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

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

$$\begin{aligned}
\sum_{j\in I} w_j &\le \sum_{\ell=1}^{s} \frac{n_\ell - n_{\ell+1}}{n_\ell} \cdot \mathrm{OPT} \\
&\le \mathrm{OPT} \sum_{\ell=1}^{s} \left( \frac{1}{n_\ell} + \frac{1}{n_\ell - 1} + \cdots + \frac{1}{n_{\ell+1}+1} \right) \\
&= \mathrm{OPT} \sum_{i=1}^{k} \frac{1}{i} \\
&= H_n \cdot \mathrm{OPT} \le \ln n + 1 \quad .
\end{aligned}$$



## 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):**

$$\begin{aligned}\Pr[u \text{ not covered in one round}] &= \prod_{j:u\in S_j}(1-x_j) \le \prod_{j:u\in S_j} e^{-x_j}\\ &= e^{-\sum_{j:u\in S_j} x_j} \le e^{-1} \enspace .\end{aligned}$$

**Probability that $u \in U$ is not covered (after $\ell$ rounds):**

$$\Pr[u \text{ not covered after } \ell \text{ round}] \le \frac{1}{e^{\ell}} \enspace .$$



$$\begin{aligned}&\Pr[\exists u \in U \text{ not covered after } \ell \text{ round}]\\ &\quad = \Pr[u_1 \text{ not covered} \lor u_2 \text{ not covered} \lor \ldots \lor u_n \text{ not covered}]\\ &\quad \le \sum_i \Pr[u_i \text{ not covered after } \ell \text{ rounds}] \le ne^{-\ell} \enspace .\end{aligned}$$

### Lemma 41

*With high probability $\mathcal{O}(\log n)$ rounds suffice.*

**With high probability:**
For any constant $\alpha$ the number of rounds is at most $\mathcal{O}(\log n)$ with probability at least $1 - n^{-\alpha}$.



**Proof:** We have

$$\Pr[\#\text{rounds} \ge (\alpha+1)\ln n] \le ne^{-(\alpha+1)\ln n} = n^{-\alpha} \enspace .$$



## Expected Cost

- Version A.
  Repeat for $s = (\alpha+1)\ln n$ rounds. If you don't have a cover simply take all sets.

  $$E[\text{cost}] \le (\alpha+1)\ln n \cdot \text{cost}(LP) + (\sum_j w_j)n^{-\alpha} = \mathcal{O}(\ln n)\cdot \text{OPT}$$

  If the weights are polynomially bounded (smallest weight is 1), sufficiently large $\alpha$ and OPT at least 1.

## 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

$E[\text{cost} \mid \text{success}]$

$$= \frac{1}{\Pr[\text{sucess}]}\Big(E[\text{cost}] - \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}]\Big)$$

$$\leq \frac{1}{\Pr[\text{sucess}]} E[\text{cost}] \leq \frac{1}{1 - n^{-\alpha}} (\alpha + 1)\ln n \cdot \text{cost}(LP)$$

$$\leq 2(\alpha + 1)\ln n \cdot \text{OPT}$$

for $n \geq 2$ and $\alpha \geq 1$.



Randomized rounding gives an $\mathcal{O}(\log n)$ approximation. The running time is polynomial with high probability.

### Theorem 42

*There is no approximation algorithm for set cover with approximation guarantee better than $\frac{1}{2}\log n$ unless $\mathrm{NP}$ has quasi-polynomial time algorithms (algorithms with running time $2^{\mathrm{poly}(\log n)}$).*



**Techniques:**

- Deterministic Rounding
- Rounding of the Dual
- Primal Dual
- Greedy
- Randomized Rounding
- Local Search
- Rounding the Data + Dynamic Programming



## Scheduling Jobs on Identical Parallel Machines

Given $n$ jobs, where job $j \in \{1, \dots, n\}$ 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

The average work performed by a machine is $\frac{1}{m}\sum_j p_j$. Therefore,

$$C^*_{\max} \geq \frac{1}{m}\max_j p_j$$



## 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 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 perform that reduces the makespan perform the switch.

REPEAT

## Local Search Analysis

Let $\ell$ be the job that finishes last in the produces schedule.

Let $S_\ell$ its start time, and let $C_\ell$ its completion time.

Note that every machine is busy before time $S_\ell$, because otherwise we could move the job $\ell$ and hence our schedule would not be locally optimal.



We can split the total processing time into two intervals one from $0$ to $S_\ell$ the other from $S_\ell$ to $C_\ell$.

The interval $[S_\ell, C_\ell]$ is of length $p_\ell \le C^*_{\max}$.

During the first interval $[0, S_\ell]$ all jobs 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

$$p_\ell + \frac{1}{m} \sum_{j \ne \ell} p_j = (1 - \frac{1}{m}) p_\ell + \frac{1}{m} \sum_j p_j \le (2 - \frac{1}{m}) C^*_{\max}$$



## 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 43
*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

$$C^*_{\max} + p_\ell \leq \frac{4}{3} C^*_{\max} .$$

  Hence, $p_n > C^*_{\max}/3$.
- This means that all jobs must have a processing time $> C^*_{\max}$.
- But then any machine in the optimum schedule can handle at most two jobs.
- For such instances Longest-Processing-Time-First is optimal.



# Traveling Salesman

Given a set of cities ($\{1, \ldots, n\}$) and a symmetric matrix $C = (c_{ij})$, $c_{ij} \geq 0$ that specifies for every pair $(i,j) \in [n] \times [n]$ the cost for travelling from city $i$ to city $j$. Find a permutation $\pi$ of the cities such that the round-trip cost

$$c_{\pi(1)\pi(n)} + \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)}$$

is minimized.



# Traveling Salesman

Theorem 44

*There does not exist an $O(2^n)$-approximation algorithm for TSP.*

**Hamiltonian Cycle**:
For a given undirected graph $G = (V, E)$ decide whether there exists a simple cycle that contains all nodes in $G$.

- Given an instance to HAMPATH we create an instance for TSP.
- If $(i,j) \notin E$ then set $c_{ij}$ to $n2^n$ otw. set $c_{ij}$ to 1. This instance has polynomial size.
- There exists a Hamiltonian Path iff there exists a tour with cost $n$. Otw. any tour has cost strictly larger than $2^n$.
- An $\mathcal{O}(2^n)$-approximation algorithm could decide btw. these cases. Hence, cannot exist unless $P = NP$.



# Metric Traveling Salesman

In the metric version we assume for every triple $i, j, k \in \{1, \ldots, n\}$

$$c_{ij} \leq c_{ij} + c_{jk} .$$

It is convenient to view the input as a complete undirected graph $G = (V, E)$, where $c_{ij}$ for an edge $(i,j)$ defines the distance between nodes $i$ and $j$.

## TSP: Lower Bound I

**Lemma 45**

*The cost $\mathrm{OPT}_{TSP}(G)$ of an optimum traveling salesman tour is at least as large as the weight $\mathrm{OPT}_{MST}(G)$ of a minimum spanning tree in $G$.*

**Proof:**

- Take the optimum TSP-tour.
- Delete one edge.
- This gives a spanning tree of cost at most $\mathrm{OPT}_{\mathrm{TSP}}(G)$.



## TSP: Greedy Algorithm

- Start with a tour on a subset $S$ containing a single node.
- Take the node $v$ closest to $S$. Add it $S$ and expand the existing tour on $S$ to include $v$.
- Repeat until all nodes have been processed.



## TSP: Greedy Algorithm

**Lemma 46**

*The Greedy algorithm is a $2$-approximation algorithm.*

Let $S_i$ be the set at the start of the $i$-th iteration, and let $v_i$ denote the node added during the iteration.

Further let $s_i \in S_i$ be the node closest to $v_i \in S_i$.

Let $r_i$ denote the successor of $s_i$ in the tour before inserting $v_i$.

We replace the edge $(s_i, r_i)$ in the tour by the two edges $(s_i, v_i)$ and $(v_i, r_i)$.

This increases the cost by

$$c_{s_i,v_i} + c_{v_i,r_i} - c_{s_i,r_i} \le 2c_{s_i,v_i}$$



## TSP: Greedy Algorithm

The edges $(s_i, v_i)$ considered during the Greedy algorithm are exactly the edges considered during PRIMs MST algorithm.

Hence,

$$\sum_i c_{s_i,v_i} = \mathrm{OPT}_{\mathrm{MST}}(G)$$

which with the previous lower bound gives a $2$-approximation.

## TSP: A different approach

Suppose that we are given an Eulerian graph $G' = (V, E', c')$ of $G = (V, E, c)$ such that for any edge $(i,j) \in E'$ $c'(i,j) \geq c(i,j)$.

Then we can find a TSP-tour of cost at most

$$\sum_{e \in E'} c'(e)$$

- Find an Euler tour of $G'$.
- Fix a permutation of the cities (i.e., a TSP-tour) by traversing the Euler tour and only note the first occurrence of a city.
- The cost of this TSP tour is at most the cost of the Euler tour because of triangle inequality.

This technique is known as short cutting the Euler tour.



## TSP: A different approach

Consider the following graph:

- Compute an MST of $G$.
- Duplicate all edges.

This graph is Eulerian, and the total cost of all edges is at most $2 \cdot \mathrm{OPT}_{\mathrm{MST}}(G)$.

Hence, short-cutting gives a tour of cost no more than $2 \cdot \mathrm{OPT}_{\mathrm{MST}}(G)$ which means we have a $2$-approximation.



## TSP: Can we do better?

Duplicating all edges in the MST seems to be rather wasteful.

We only need to make the graph Eulerian.

For this we compute a Minimum Matching between odd degree vertices in the MST (note that there are an even number of them).



## TSP: Can we do better?

An optimal tour on the odd-degree vertices has cost at most $\mathrm{OPT}_{\mathrm{TSP}}(G)$.

However, the edges of this tour give rise to two disjoint matchings. One of these matchings must have weight less than $\mathrm{OPT}_{\mathrm{TSP}}(G)/2$.

Adding this matching to the MST gives an Eulerian graph with edge weight at most

$$\mathrm{OPT}_{\mathrm{MST}}(G) + \mathrm{OPT}_{\mathrm{TSP}}(G)/2 \leq \frac{3}{2}\mathrm{OPT}_{\mathrm{TSP}}(G) \enspace ,$$

Short cutting gives a $\frac{3}{2}$-approximation for metric TSP.

This is the best that is known.

# 15 Rounding Data + Dynamic Programming

**Knapsack:**
Given a set of items $\{1,\dots,n\}$, 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 $I \subseteq \{1,\dots,n\}$ of items of total weight at most $W$ such that the profit is maximized (we can assume each $w_i \le W$).

$$\begin{array}{llrcl} \max & & \sum_{i=1}^{n} p_i x_i & & \\ \text{s.t.} & & \sum_{i=1}^{n} w_i x_i & \le & W \\ & \forall i \in \{1,\dots,n\} & x_i & \in & \{0,1\} \end{array}$$



# 15 Rounding Data + Dynamic Programming

**Algorithm 6** 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.



# 15 Rounding Data + Dynamic Programming

**Definition 47**
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.



# 15 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

$$\mathcal{O}(nP') = \mathcal{O}\Big(n\sum_i p_i'\Big) = \mathcal{O}\Big(n\sum_i \Big\lfloor \frac{p_i}{\epsilon M/n} \Big\rfloor\Big) \le \mathcal{O}\Big(\frac{n^3}{\epsilon}\Big) \ .$$

# 15 Rounding Data + Dynamic Programming

Let $S$ be the set of items returned by the algorithm, and let $O$ be an optimum set of items.

$$\begin{aligned}\sum_{i\in S} &\geq \mu \sum_{i\in S} p_i' \\ &\geq \mu \sum_{i\in O} p_i' \\ &\geq \sum_{i\in O} p_i - |O|\mu \\ &\geq \sum_{i\in O} p_i - n\mu \\ &= \sum_{i\in O} p_i - \epsilon M \\ &\geq (1-\epsilon)\mathrm{OPT} \ .\end{aligned}$$



# 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 $\ell$ 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.



# 15.2 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 $\ell$ is the last job (this only requires that all machines are busy before time $S_\ell$).

If $\ell$ 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 $\ell$ 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

$$(1 + \frac{1}{k})C^*_{\max}$$

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 48**
*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.



- 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

$$(1 + \frac{1}{k})T \ .$$

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

$$T + \frac{T}{k} \leq (1 + \frac{1}{k})T \ .$$



**Running Time:** There should not be a job with rounded size more than $T$ as otw. the problem becomes trivial.

Hence, any 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 $x$. There are only $(k+1)^{k^2}$ different vectors.

This means there are a constant number of different configurations.



Let $\mathrm{OPT}(n_1, \dots, n_{k^2})$ be the number of machines that are required to schedule input vector $(n_1, \dots, n_{k^2})$ with Makespan at most $T$.

If $\mathrm{OPT}(n_1, \dots, n_{k^2}) \leq m$ we can schedule the input.

We have

$$\mathrm{OPT}(n_1, \dots, n_{k^2}) = \begin{cases} 1 + \min\limits_{(s_1, \dots, s_{k^2}) \in C} \mathrm{OPT}(n_1 - s_1, \dots, n_{k^2} - s_{k^2}) & (n_1, \dots, n_{k^2}) \nleq 0 \\ 0 & \text{otw.} \end{cases}$$

where $C$ is the set of all configurations.

Hence, the running time is roughly $(k+1)^{k^2} n^{k^2} = (nk)^{k^2}$.



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 $1/\epsilon$.

**Theorem 49**
*There is no FPTAS for problems that are strongly NP-hard.*

## Last Time

Let $\mathrm{OPT}(n_1, \ldots, n_A)$ be the number of machines that are required to schedule input vector $(n_1, \ldots, n_A)$ with Makespan at most $T$ ($A$: number of different sizes).

If $\mathrm{OPT}(n_1, \ldots, n_A) \leq m$ we can schedule the input.

$$\mathrm{OPT}(n_1, \ldots, n_A) = \begin{cases} 0 & (n_1, \ldots, n_A) = 0 \\ 1 + \min\limits_{(s_1, \ldots, s_A) \in C} \mathrm{OPT}(n_1 - s_1, \ldots, n_A - s_A) & (n_1, \ldots, n_A) \geq 0 \\ \infty & \text{otw.} \end{cases}$$

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.



## 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 50**
*There is no $\rho$-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 $\rho$-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 51**
An asymptotic polynomial-time approximation scheme (APTAS) is a family of algorithms $\{A_\epsilon\}$ along with a constant $c$ such that $A_\epsilon$ 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 52**

*Any packing of items of size at most $\gamma$ into $\ell$ bins can be extended to a packing of all items into $\max\{\ell, \frac{1}{1-\gamma}\text{SIZE}(I)+1\}$ bins, where $\text{SIZE}(I)=\sum_i s_i$ is the sum of all item sizes.*

- If after Greedy we use more than $\ell$ bins all bins (apart from the last) must be full to at least $1-\gamma$.
- Hence, $r(1-\gamma) \le \text{SIZE}(I)$ where $r$ is the number of nearly-full bins.
- This gives the lemma.



Choose $\gamma = \epsilon/2$. Then we either use $\ell$ bins or at most

$$1/(1-\epsilon/2)\text{OPT}+1 \le (1+\epsilon)\text{OPT}+1$$

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.



**Lemma 53**

$\text{OPT}(I') \le \text{OPT}(I) \le \text{OPT}(I')+k$

**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 54**
$\mathrm{OPT}(I') \le \mathrm{OPT}(I) \le \mathrm{OPT}(I') + k$

**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 itemsfor group $2$ have been packed;
- ...



Assume that our instance does not contain pieces smaller than $\epsilon/2$. Then $\mathrm{SIZE}(I) \ge \epsilon n/2$.

We set $k = \lfloor \epsilon \mathrm{SIZE}(I) \rfloor$.

Then $n/k \le 2n/(\epsilon \mathrm{SIZE}(I)) \le 4/\epsilon^2$ (here we used $\lfloor \alpha \rfloor \ge \alpha/2$ for $\alpha \ge 1$).

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. This gives a $(1 + \epsilon)$-approximation because of the choice of $k$.



**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 $s_2$;
- ...
- $s_m$ 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,

$$\sum_i t_i \cdot s_i \leq 1 \ .$$

We call a vector that fulfills the above constraint a configuration.



## Configuration LP

Let $N$ be the number of configurations (exponential).

Let $T_1, \ldots, T_N$ be the sequence of all possible configurations (a configuration $T_j$ has $T_{ji}$ pieces of size $s_i$).

$$\begin{array}{llrcl}
\min & & \sum_{j=1}^{N} x_j & & \\
\text{s.t.} & \forall i \in \{1 \ldots m\} & \sum_{j=1}^{N} T_{ji} x_j & \geq & b_i \\
 & \forall j \in \{1, \ldots, N\} & x_j & \geq & 0 \\
 & \forall j \in \{1, \ldots, N\} & x_j & \text{integral} &
\end{array}$$



**How to solve this LP?**

later...



We can assume that each item has size at least $1/\text{SIZE}(I)$.

## 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

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 $G_2, \dots, G_{r-1}$ delete $n_i - n_{i-1}$ items.
- Observe that $n_i \geq n_{i-1}$.



**Lemma 55**
*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 56**
*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 7** 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

$$\text{OPT}_{\text{LP}}(I_1) + \text{OPT}_{\text{LP}}(I_2) \le \text{OPT}_{\text{LP}}(I') \le \text{OPT}_{\text{LP}}(I)$$

**Proof:**

- Each piece surviving in $I'$ can be mapped to a piece in $I$ of no lesser size. Hence, $\text{OPT}_{\text{LP}}(I') \le \text{OPT}_{\text{LP}}(I)$
- $\lfloor x_j \rfloor$ 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 $\text{OPT}_{\text{LP}}$ 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

We can show that $\text{size}(I_2) \le \text{SIZE}(I)/2$. 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 ($\le \text{SIZE}(I)/2$).
- 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.

# 17 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.



# 17 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 $\ell_j$.
- 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

$$W = \sum_j w_j X_j$$

$$
\begin{aligned}
E[W] &= \sum_j w_j E[X_j] \\
&= \sum_j w_j \Pr[C_j \text{ is satisified}] \\
&= \sum_j w_j (1 - (\tfrac{1}{2})^{\ell_j}) \\
&\geq \frac{1}{2} \sum_j w_j \\
&\geq \frac{1}{2} \mathrm{OPT}
\end{aligned}
$$



# 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: Randomized Rounding

Set each $x_i$ independently to true with probability $y_i$ (and, hence, to false with probability $(1 - y_i)$).



**Lemma 57 (Geometric Mean ≤ Arithmetic Mean)**

*For any nonnegative $a_1, \ldots, a_k$*

$$\left( \prod_{i=1}^{k} a_i \right)^{1/k} \leq \frac{1}{k} \sum_{i=1}^{k} a_i$$

## Lemma 58

*Let $f$ be a concave function on the interval $[0,1]$, with $f(0)=a$ and $f(1)=a+b$. Then $f(x)\ge bx+a$ for $x\in[0,1]$.*



$$\begin{aligned}\Pr[C_j \text{ not satisfied}] &= \prod_{i\in P_j}(1-y_i)\prod_{i\in N_j}y_i\\ &\le \left[\frac{1}{\ell_j}\left(\sum_{i\in P_j}(1-y_i)+\sum_{i\in N_j}y_i\right)\right]^{\ell_j}\\ &= \left[1-\frac{1}{\ell_j}\left(\sum_{i\in P_j}y_i+\sum_{i\in N_j}(1-y_i)\right)\right]^{\ell_j}\\ &\le \left(1-\frac{z_j}{\ell_j}\right)^{\ell_j} .\end{aligned}$$



The function $f(z)=1-(1-\frac{z}{\ell})^{\ell}$ is concave. Hence,

$$\begin{aligned}\Pr[C_j \text{ satisfied}] &\ge 1-\left(1-\frac{z_j}{\ell_j}\right)^{\ell_j}\\ &\ge \left[1-\left(1-\frac{1}{\ell_j}\right)^{\ell_j}\right]\cdot z_j .\end{aligned}$$



$$\begin{aligned}E[W] &= \sum_j w_j \Pr[C_j \text{ is satisfied}]\\ &\ge \sum_j w_j z_j\left[1-\left(1-\frac{1}{\ell_j}\right)^{\ell_j}\right]\\ &\ge \left(1-\frac{1}{e}\right)\mathrm{OPT} .\end{aligned}$$

## MAXSAT: The better of two

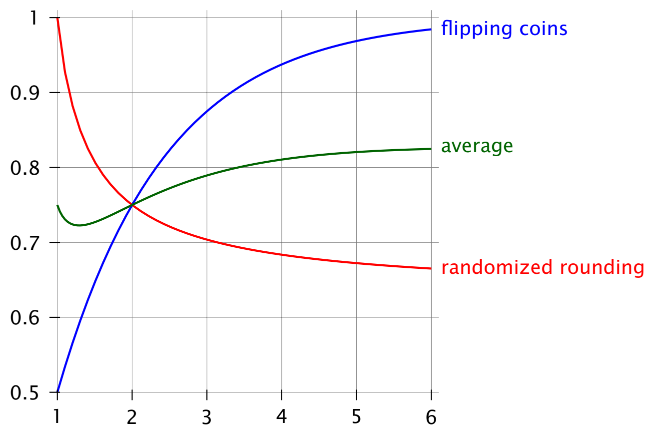
**Theorem 59**
*Choosing the better of the two solutions given by randomized rounding and coin flipping yields a $\frac{3}{4}$-approximation.*



Let $W_1$ be the value of randomized rounding and $W_2$ the value obtained by coin flipping.

$$
\begin{aligned}
E[\max\{W_1, W_2\}] &\geq E[\tfrac{1}{2}W_1 + \tfrac{1}{2}W_2] \\
&\geq \frac{1}{2}\sum_j w_j z_j \left[1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right] + \frac{1}{2}\sum_j w_j (1 - 2^{-\ell_j}) \\
&\geq \sum_j w_j z_j \Bigg[\underbrace{\frac{1}{2}\left(1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right) + \frac{1}{2}\left(1 - 2^{-\ell_j}\right)}_{\geq \frac{3}{4}}\Bigg] \\
&\geq \frac{3}{4}\mathrm{OPT}
\end{aligned}
$$








## 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 $f : [0,1] \to [0,1]$ and set $x_i$ to true with probability $f(y_i)$.

# MAXSAT: Nonlinear Randomized Rounding

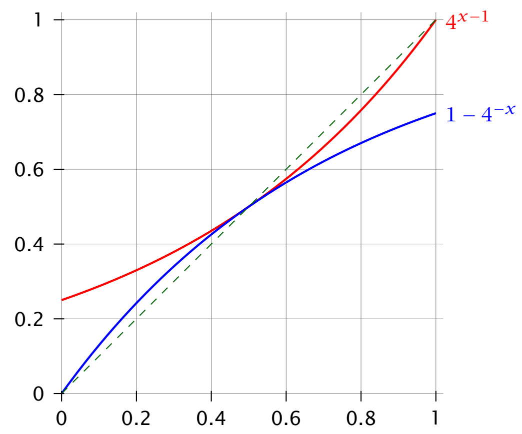
Let $f : [0,1] \to [0,1]$ be a function with

$$1 - 4^{-x} \le f(x) \le 4^{x-1}$$

**Theorem 60**

*Rounding the LP-solution with a function $f$ of the above form gives a $\frac{3}{4}$-approximation.*








$$\begin{aligned}
\Pr[C_j \text{ not satisfied}] &= \prod_{i \in P_j} (1 - f(y_i)) \prod_{i \in N_j} y_i \\
&\le \prod_{i \in P_j} 4^{-y_i} \prod_{i \in N_j} 4^{y_i - 1} \\
&= 4^{-(\sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i))} \\
&\le 4^{-z_j}
\end{aligned}$$



The function $g(z) = 1 - 4^{-z}$ is concave on $[0,1]$. Hence,

$$\Pr[C_j \text{ satisfied}] \ge 1 - 4^{-z_j} \ge \frac{3}{4} z_j \ .$$

Therefore,

$$E[W] = \sum_j w_j \Pr[C_j \text{ satisfied}] \ge \frac{3}{4} \sum_j w_j z_j \ge \frac{3}{4} \mathrm{OPT}$$

**Can we do better?**

Not if we compare ourselves to the value of an optimum LP-solution.

**Definition 61 (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 62**

*Our ILP-formulation for the MAXSAT problem has integrality gap at most $\frac{3}{4}$.*



# Facility Location

**Integer Program**

$$
\begin{array}{llrcl}
\min & & \multicolumn{3}{l}{\sum_{i\in F} f_i y_i + \sum_{i\in F}\sum_{j\in D} c_{ij}x_{ij}} \\
\text{s.t.} & \forall j\in D & \sum_{i\in F} x_{ij} & = & 1 \\
 & \forall i\in F, j\in D & x_{ij} & \le & y_i \\
 & \forall i\in F, j\in D & x_{ij} & \in & \{0,1\} \\
 & \forall i\in F & y_i & \in & \{0,1\}
\end{array}
$$

As usual we get an LP by relaxing the integrality constraints.



# Facility Location

**Dual Linear Program**

$$
\begin{array}{llrcl}
\max & & \multicolumn{3}{l}{\sum_{j\in D} v_j} \\
\text{s.t.} & \forall i\in F & \sum_{j\in D} w_{ij} & \le & f_i \\
 & \forall i\in F, j\in D & v_j - w_{ij} & \le & c_{ij} \\
 & \forall i\in F, j\in D & w_{ij} & \ge & 0
\end{array}
$$

# Facility Location

**Definition 63**
Given an LP solution $(x^*, y^*)$ we say that facility $i$ neighbours client $j$ if $x_{ij} > 0$. Let $N(j) = \{i \in F : x^*_{ij} > 0\}$.



**Lemma 64**
*If $(x^*, y^*)$ is an optimal solution to the facility location LP and $(v^*, w^*)$ is an optimal dual solution, then $x^*_{ij} > 0$ implies $c_{ij} < v^*_j$.*

Follows from slackness conditions.



Suppose we open set $S \subseteq F$ of facilities s.t. for all clients we have $S \cap N(j) \neq \emptyset$.

Then every client $j$ has a facility $i$ s.t. assignment cost for this client is at most $c_{ij} \leq v^*_j$.

Hence, the total assignment cost is

$$\sum_j c_{i_j j} \leq \sum_j v^*_j \leq \mathrm{OPT} \ ,$$

where $i_j$ is the facility that client $j$ is assigned to.



**Problem: Facility cost may be huge!**

Suppose we can partition a subset $F' \subseteq F$ of facilities into neighbour sets of some clients. I.e.

$$F' = \biguplus_k N(j_k)$$

where $j_1, j_2, \ldots$ form a subset of the clients.

Now in each set $N(j_k)$ we open the cheapest facility. Call it $f_{i_k}$.

We have

$$f_{i_k} = f_{ik} \sum_{i \in N(j_k)} x^*_{ij_k} \le \sum_{i \in N(j_k)} f_i x^*_{ij_k} \le \sum_{i \in N(j_k)} f_i y^*_i \ .$$

Summing over all $k$ gives

$$\sum_k f_{i_k} \le \sum_k \sum_{i \in N(j_k)} f_i y^*_i = \sum_{i \in F'} f_i y^*_i \le \sum_{i \in F} f_i y^*_i$$

Facility cost is at most the facility cost in an optimum solution.



**Problem: so far clients $j_1, j_2, \ldots$ have a neighboring facility. What about the others?**

### Definition 65

Let $N^2(j)$ denote all neighboring clients of the neighboring facilities of client $j$.

Note that $N(j)$ is a set of facilities while $N^2(j)$ is a set of clients.



**Algorithm 8** FacilityLocation

```
C ← D// unassigned clients
k ← 0
while C ≠ 0 do
    k ← k + 1
    choose j_k ∈ C that minimizes v*_j
    choose i_k ∈ N(j_k) as cheapest facility
    assign j_k and all unassigned clients in N²(j_k) to i_k
    C ← C − {j_k} − N²(j_k)
```



Facility cost of this algorithm is at most OPT because the sets $N(j_k)$ are disjoint.

**Total assignment cost:**

- Fix $k$; set $j = j_k$ and $i = i_k$. We know that $c_{ij} \le v^*_j$.
- Let $\ell \in N^2(j)$ and $h$ (one of) its neighbour(s) in $N(j)$.

$$c_{i\ell} \le c_{ij} + c_{hj} + c_{h\ell} \le v^*_j + v^*_j + v^*_\ell \le 3v^*_\ell$$

Summing this over all facilities gives that the total assignment cost is at most $3 \cdot \mathrm{OPT}$. Hence, we get a 4-approximation.

In the above analysis we use the inequality

$$\sum_{i\in F} f_i y_i^* \leq \mathrm{OPT}\ .$$

We know something stronger namely

$$\sum_{i\in F} f_i y_i^* + \sum_{i\in F}\sum_{j\in D} c_{ij}x_{ij}^* \leq \mathrm{OPT}\ .$$



**Observation:**

- Suppose when choosing a client $j_k$, instead of opening the cheapest facility in its neighborhood we choose a random facility according to $x_{ij_k}^*$.
- Then we incur connection cost

  $$\sum_i c_{ij_k}x_{ij_k}^*$$

  for client $j_k$. (In the previous algorithm we estimated this by $v_{j_k}^*$).
- Define

  $$C_j^* = \sum_i c_{ij}x_{ij}^*$$

  to be the connection cost for client $j$.



**What will our facility cost be?**

We only try to open a facility once (when it is in neighborhood of some $j_k$). (recall that neighborhoods of different $j_k's$ are disjoint).

We open facility $i$ with probability $x_{ij_k} \leq y_i$ (in case it is in some neighborhood; otw. we open it with probability zero).

Hence, the expected facility cost is at most

$$\sum_{i\in F} f_i y_i\ .$$



**Algorithm 9** FacilityLocation

$C \leftarrow D$// unassigned clients
$k \leftarrow 0$
**while** $C \neq 0$ **do**
    $k \leftarrow k+1$
    choose $j_k \in C$ that minimizes $v_j^* + C_j^*$
    choose $i_k \in N(j_k)$ according to probability $x_{ij_k}$.
    assign $j_k$ and all unassigned clients in $N^2(j_k)$ to $i_k$
    $C \leftarrow C - \{j_k\} - N^2(j_k)$

**Total assignment cost:**

- Fix $k$; set $j = j_k$.
- Let $\ell \in N^2(j)$ and $h$ (one of) its neighbour(s) in $N(j)$.
- If we assign a client $\ell$ to the same facility as $i$ we pay at most

$$\sum_i c_{ij} x^*_{ij_k} + c_{hj} + c_{h\ell} \leq C^*_j + v^*_j + v^*_\ell \leq C^*_\ell + 2v^*_\ell$$

Summing this over all clients gives that the total assignment cost is at most

$$\sum_j C^*_j + \sum_j 2v^*_j \leq \sum_j C^*_j + 2\mathrm{OPT}$$

Hence, it is at most $2\mathrm{OPT}$ plus the total assignment cost in an optimum solution.

Adding the facility cost gives a $3$-approximation.



Lemma 66 (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 67

*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*

$$\left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^L \leq e^{-L\delta^2/2}$$



## Integer Multicommodity Flows

- Given $s_i$-$t_i$ pairs in a graph.
- Connect each pair by a paths such that not too many path use any given edge.

$$\begin{array}{llrcl} \min & & & W & \\ \text{s.t.} & \forall i & \sum_{p \in \mathcal{P}_i} x_p & = & 1 \\ & & \sum_{p: e \in p} x_p & \leq & W \\ & & x_p & \in & \{0,1\} \end{array}$$

## 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 68**
*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 $W^* + \sqrt{cW^* \ln n}$.*



## 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

Choose $\delta = \sqrt{(c \ln n)/W^*}$.

Then

$$\Pr[Y_e \geq (1+\delta)W^*] < e^{-W^*\delta^2/3} = \frac{1}{n^{c/3}}$$

## 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

**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

$$\sum_j w_j = \sum_j \sum_{e\in S_j} y_e = \sum_e |\{j : e \in S_j\}| \cdot y_e \leq f \cdot \sum_e y_e \leq f \cdot \mathrm{OPT}$$



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:

$$y_e > 0 \Rightarrow 1 \le \sum_{j:e\in S_j} x_j \le f$$

This is sufficient to show that the solution is an $f$-approximation.



Suppose we have a primal/dual pair

| | | | | | |
|---|---|---|---|---|---|
| min | | $\sum_j c_j x_j$ | | | |
| s.t. | $\forall i$ | $\sum_{j:} a_{ij} x_j$ | $\ge$ | $b_i$ |
| | $\forall j$ | $x_j$ | $\ge$ | $0$ |

| | | | | |
|---|---|---|---|---|
| max | | $\sum_i b_i y_i$ | | |
| s.t. | $\forall j$ | $\sum_i a_{ij} y_i$ | $\le$ | $c_j$ |
| | $\forall i$ | $y_i$ | $\ge$ | $0$ |

and solutions that fulfill approximate slackness conditions:

$$x_j > 0 \Rightarrow \sum_i a_{ij} y_i \ge \frac{1}{\alpha} c_j$$

$$y_i > 0 \Rightarrow \sum_j a_{ij} x_j \le \beta b_i$$



Then

$$\underbrace{\boxed{\sum_j \overbrace{\boxed{c_j}}^{\text{right hand side of } j\text{-th dual constraint}} x_j}}_{\text{primal cost}} \le \alpha \sum_j \left(\sum_i a_{ij} y_i\right) x_j$$

$$= \alpha \sum_i \left(\sum_j a_{ij} x_j\right) y_i$$

$$\le \alpha\beta \cdot \underbrace{\boxed{\sum_i b_i y_i}}_{\text{dual objective}}$$



# Feedback Vertex Set for Undirected Graphs

- Given a graph $G = (V, E)$ and non-negative weights $w_v \ge 0$ 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.
- 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)

**Primal Relaxation:**

$$\begin{array}{llrcl} \min & & \sum_v w_v x_v & & \\ \text{s.t.} & \forall C \in C & \sum_{v \in C} x_v & \geq & 1 \\ & \forall v & x_v & \geq & 0 \end{array}$$

**Dual Formulation:**

$$\begin{array}{llrcl} \max & & \sum_{C \in C} y_C & & \\ \text{s.t.} & \forall v \in V & \sum_{C: v \in C} y_C & \leq & w_v \\ & \forall C & y_C & \geq & 0 \end{array}$$



If we perform the previous dual technique for Set Cover we get the following:

- Start with $x = 0$ and $y = 0$
- While there is a cycle $C$ that is not covered (does not contain a chosen vertex).
  - Increase $y_e$ until dual constraint for some vertex $v$ becomes tight.
  - set $x_v = 1$.



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 10** FeedbackVertexSet

$y \leftarrow 0$
$x \leftarrow 0$
**while** exists cycle $C$ in $G$ **do**
  increase $y_C$ until there is $v \in C$ s.t. $\sum_{C:v\in C} y_C = w_v$
  $x_v = 1$
  remove $v$ from $G$
  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 $\alpha$ we get an $\alpha$-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 $\alpha$ we get an $\alpha$-approximation.

**Theorem 69**
*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 $c$.

$$\begin{array}{llrcl} \min & & \sum_e c(e)x_e & & \\ \text{s.t.} & \forall S \in \mathcal{S} & \sum_{e:\delta(S)} x_e & \ge & 1 \\ & \forall e \in E & x_e & \in & \{0,1\} \end{array}$$

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

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 11** PrimalDualShortestPath

```
y ← 0
F ← ∅
while there is no s-t path in (V,F) do
      Let C be the connected component of (V,F) containing s
      Increase y_C until there is an edge e' ∈ δ(C) such that ∑_{S:e'∈δ(S)} y_S = c(e').
      F ← F ∪ {e'}
Let P be an s-t path in (V,F)
return P
```



**Lemma 70**

*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.

$$\sum_{e\in P} c_(e) = \sum_{e\in P} \sum_{S:e\in\delta(S)} y_S$$
$$= \sum_{S:s\in S, t\notin S} |P\cap\delta(S)|\cdot y_S\ .$$

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.



**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 $F$.

$$\begin{array}{lrccc} \min & & \sum_e c(e)x_e & & \\ \text{s.t.} & \forall S\subseteq V: S\in S_i \text{ for some } i & \sum_{e\in\delta(S)} x_e & \ge & 1 \\ & \forall e\in E & x_e & \in & \{0,1\} \end{array}$$

Here $S_i$ contains all sets $S$ such that $s_i \in S$ and $t_i \notin S$.



$$\begin{array}{lrccc} \max & & \sum_{S:\,\exists i \text{ s.t. } S\in S_i} y_S & & \\ \text{s.t.} & \forall e\in E & \sum_{S:e\in\delta(S)} y_S & \le & c(e) \\ & & y_S & \ge & 0 \end{array}$$

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 12** FirstTry

```
y ← 0
F ← ∅
while not all s_i-t_i pairs connected in F do
    Let C be some connected component of (V,F) such
    that |C ∩ {s_i, t_i}| = 1 for some i.
    Increase y_C until there is an edge e' ∈ δ(C) s.t.
    ∑_{S∈S_i : e'∈δ(S)} y_S = c_{e'}
    F ← F ∪ {e'}
Let P_i be an s_i-t_i path in (V,F)
return ⋃_i P_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 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 $|\delta(\{v_0\})\cap F| = k$.



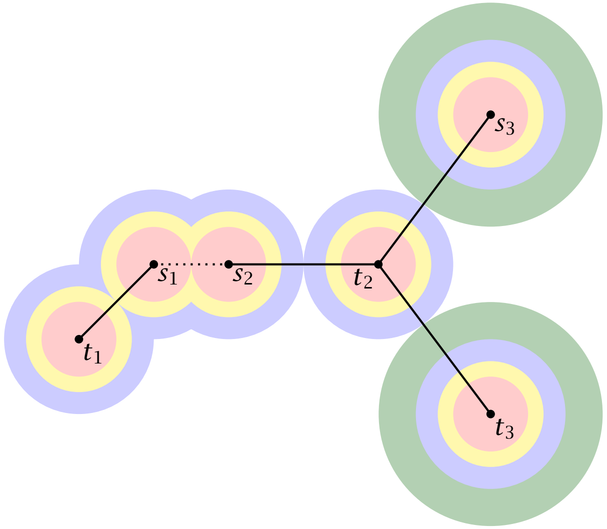
**Algorithm 13** SecondTry

```
y ← 0; F ← ∅; ℓ ← 0
while not all s_i-t_i pairs connected in F do
    ℓ ← ℓ + 1
    Let C be set of all connected components C of (V,F)
    such that |C ∩ {s_i, t_i}| = 1 for some i.
    Increase y_C for all C ∈ C uniformly until for some edge
    e_ℓ ∈ δ(C'), C' ∈ C s.t. ∑_{S:e_ℓ∈δ(S)} y_S = c_{e_ℓ}
    F ← F ∪ {e_ℓ}
F' ← F
for k ← ℓ 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






### Lemma 71

*For any $C$ in any iteration of the algorithm*

$$\sum_{C\in C} |\delta(C)\cap F'| \le 2|C|$$

This means that the number of times a moat from $C$ 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

  $$\epsilon \sum_{C\in C} |F'\cap\delta(C)|$$

  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 72

*For any set of connected components $C$ in any iteration of the algorithm*

$$\sum_{C\in C} |\delta(C)\cap F'| \le 2|C|$$

**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

$$\sum_{v\in R} \deg(v) \geq \sum_{C\in C} |\delta(C) \cap F'| \overset{?}{\leq} 2|C| = 2|R|$$



- Suppose that no node in $B$ has degree one.
- Then

$$\begin{aligned}\sum_{v\in R} \deg(v) &= \sum_{v\in R\cup B} \deg(v) - \sum_{v\in B} \deg(v) \\ &\leq 2(|R| + |B|) - 2|B| = 2|R|\end{aligned}$$

- 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.
  - But this means that the cluster corresponding to $b$ must separate a source-target pair.
  - But then it must be a red node.