# Part III

# Approximation Algorithms

There are many practically important optimization problems that are NP-hard.

**What can we do?**

- Heuristics.
- Exploit special structure of instances occurring in practice.
- Consider algorithms that do not compute the optimal solution but provide solutions that are close to optimum.

There are many practically important optimization problems that are NP-hard.

**What can we do?**

- Heuristics.
- Exploit special structure of instances occurring in practise.
- Consider algorithms that do not compute the optimal solution but provide solutions that are close to optimum.

There are many practically important optimization problems that are NP-hard.

**What can we do?**

- Heuristics.
- Exploit special structure of instances occurring in practise.
- Consider algorithms that do not compute the optimal solution but provide solutions that are close to optimum.

There are many practically important optimization problems that are NP-hard.

**What can we do?**

- Heuristics.
- Exploit special structure of instances occurring in practise.
- Consider algorithms that do not compute the optimal solution but provide solutions that are close to optimum.

There are many practically important optimization problems that are NP-hard.

**What can we do?**

- Heuristics.
- Exploit special structure of instances occurring in practise.
- Consider algorithms that do not compute the optimal solution but provide solutions that are close to optimum.

**Definition 2**

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.

## Why approximation algorithms?

- We need algorithms for hard problems.
- It gives a rigorous mathematical base for studying heuristics.
- It provides a metric to compare the difficulty of various optimization problems.
- Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

## Why not?

- Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

## Why approximation algorithms?

- We need algorithms for hard problems.
- It gives a rigorous mathematical base for studying heuristics.
- It provides a metric to compare the difficulty of various optimization problems.
- Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

## Why not?

- Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

## Why approximation algorithms?

- We need algorithms for hard problems.
- It gives a rigorous mathematical base for studying heuristics.
- It provides a metric to compare the difficulty of various optimization problems.
- Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

## Why not?

- Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

## Why approximation algorithms?

- We need algorithms for hard problems.
- It gives a rigorous mathematical base for studying heuristics.
- It provides a metric to compare the difficulty of various optimization problems.
- Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

## Why not?

- Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

## Why approximation algorithms?

- We need algorithms for hard problems.
- It gives a rigorous mathematical base for studying heuristics.
- It provides a metric to compare the difficulty of various optimization problems.
- Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

## Why not?

- Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

## Why approximation algorithms?

- We need algorithms for hard problems.
- It gives a rigorous mathematical base for studying heuristics.
- It provides a metric to compare the difficulty of various optimization problems.
- Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

## Why not?

- Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

## Why approximation algorithms?

- We need algorithms for hard problems.
- It gives a rigorous mathematical base for studying heuristics.
- It provides a metric to compare the difficulty of various optimization problems.
- Proving theorems may give a deeper theoretical understanding which in turn leads to new algorithmic approaches.

## Why not?

- Sometimes the results are very pessimistic due to the fact that an algorithm has to provide a close-to-optimum solution on every instance.

## Definition 3

An optimization problem $P = (\mathcal{I}, \mathrm{sol}, m, \mathrm{goal})$ is in **NPO** if

- $x \in \mathcal{I}$ can be decided in polynomial time
- $y \in \mathrm{sol}(\mathcal{I})$ can be verified in polynomial time
- $m$ can be computed in polynomial time
- $\mathrm{goal} \in \{\min, \max\}$

In other words: the decision problem is there a solution $y$ with $m(x, y)$ at most/at least $z$ is in NP.

- $x$ is problem instance
- $y$ is candidate solution
- $m^*(x)$ cost/profit of an optimal solution

**Definition 4 (Performance Ratio)**

$$R(x,y) := \max\left\{\frac{m(x,y)}{m^*(x)}, \frac{m^*(x)}{m(x,y)}\right\}$$

**Definition 5 ($r$-approximation)**

An algorithm $A$ is an $r$-approximation algorithm iff

$$\forall x \in \mathcal{I} : R(x, A(x)) \leq r \;,$$

and $A$ runs in polynomial time.

## Definition 6 (PTAS)

A PTAS for a problem $P$ from NPO is an algorithm that takes as input $x \in \mathcal{I}$ and $\epsilon > 0$ and produces a solution $y$ for $x$ with

$$R(x, y) \le 1 + \epsilon \ .$$

The running time is polynomial in $|x|$.

approximation with arbitrary good factor... fast?

## Problems that have a PTAS

**Scheduling.** Given $m$ jobs with known processing times; schedule the jobs on $n$ machines such that the MAKESPAN is minimized.

## Definition 7 (FPTAS)

An FPTAS for a problem $P$ from NPO is an algorithm that takes as input $x \in \mathcal{I}$ and $\epsilon > 0$ and produces a solution $y$ for $x$ with

$$R(x,y) \le 1 + \epsilon \ .$$

The running time is polynomial in $|x|$ and $1/\epsilon$.

approximation with arbitrary good factor... fast!

## Problems that have an FPTAS

**KNAPSACK.** Given a set of items with profits and weights choose a subset of total weight at most $W$ s.t. the profit is maximized.

### Definition 8 (APX – approximable)

A problem $P$ from NPO is in APX if there exist a constant $r \geq 1$ and an $r$-approximation algorithm for $P$.

constant factor approximation...

## Problems that are in APX

**MAXCUT.** Given a graph $G = (V, E)$; partition $V$ into two disjoint pieces $A$ and $B$ s. t. the number of edges between both pieces is maximized.

**MAX-3SAT.** Given a 3CNF-formula. Find an assignment to the variables that satisfies the maximum number of clauses.

**Problems with polylogarithmic approximation guarantees**

- Set Cover
- Minimum Multicut
- Sparsest Cut
- Minimum Bisection

There is an $r$-approximation with $r \leq \mathcal{O}(\log^c(|x|))$ for some constant $c$.

Note that only for some of the above problem a matching lower bound is known.

**There are really difficult problems!**

**Theorem 9**

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

Note that an $n$-approximation is trivial.

**There are really difficult problems!**

**Theorem 9**
*For any constant $\epsilon > 0$ there does not exist an $\Omega(n^{1-\epsilon})$-approximation algorithm for the maximum clique problem on a given graph $G$ with $n$ nodes unless $\mathrm{P} = \mathrm{NP}$.*

Note that an $n$-approximation is trivial.

**There are really difficult problems!**

**Theorem 9**

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

Note that an $n$-approximation is trivial.

**There are weird problems!**

Asymmetric $k$-Center admits an $\mathcal{O}(\log^* n)$-approximation.

There is no $o(\log^* n)$-approximation to Asymmetric $k$-Center unless $NP \subseteq DTIME(n^{\log\log\log n})$.

Class APX not important in practise.

Instead of saying problem $P$ is in APX one says problem $P$ admits a 4-approximation.

One only says that a problem is APX-hard.

A crucial ingredient for the design and analysis of approximation algorithms is a technique to obtain an upper bound (for maximization problems) or a lower bound (for minimization problems).

Therefore Linear Programs or Integer Linear Programs play a vital role in the design of many approximation algorithms.

A crucial ingredient for the design and analysis of approximation algorithms is a technique to obtain an upper bound (for maximization problems) or a lower bound (for minimization problems).

Therefore Linear Programs or Integer Linear Programs play a vital role in the design of many approximation algorithms.

**Definition 10**

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

**Definition 11**

A Mixed Integer Program is a Linear Program in which a subset of the variables are required to be integral.

**Definition 10**

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

**Definition 11**

A Mixed Integer Program is a Linear Program in which a subset of the variables are required to be integral.

Many important combinatorial optimization problems can be formulated in the form of an Integer Program.

Note that solving Integer Programs in general is NP-complete!

Many important combinatorial optimization problems can be formulated in the form of an Integer Program.

**Note that solving Integer Programs in general is NP-complete!**

# Set Cover

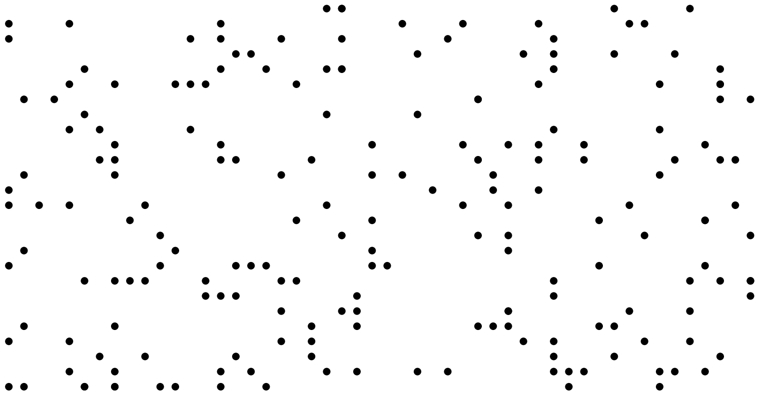
Given a ground set $U$, a collection of subsets $S_1, \dots, S_k \subseteq U$, where the $i$-th subset $S_i$ has weight/cost $w_i$. Find a collection $I \subseteq \{1, \dots, k\}$ such that

$$\forall u \in U \exists i \in I: \ u \in S_i \quad \text{(every element is covered)}$$

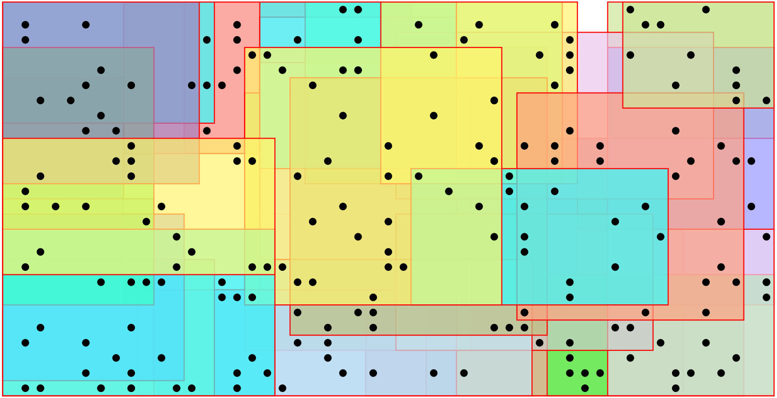
and

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

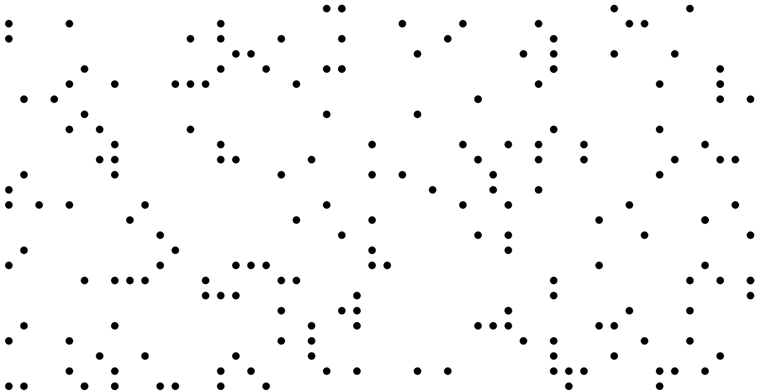
# Set Cover



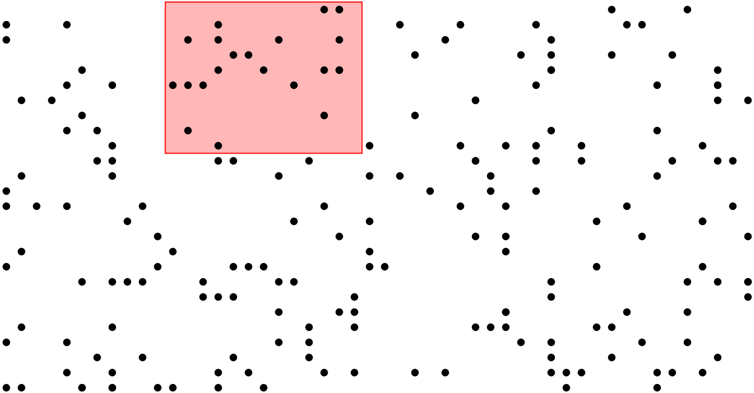
# Set Cover



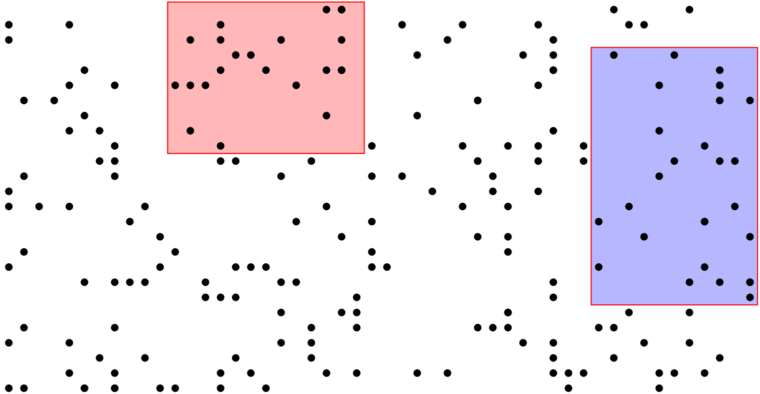
# Set Cover



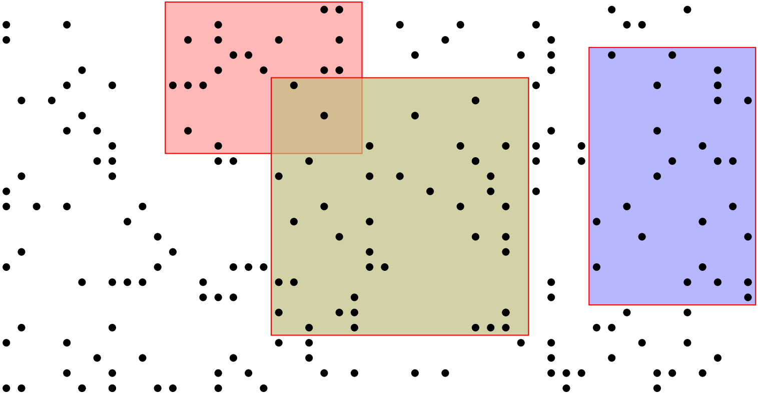
# Set Cover



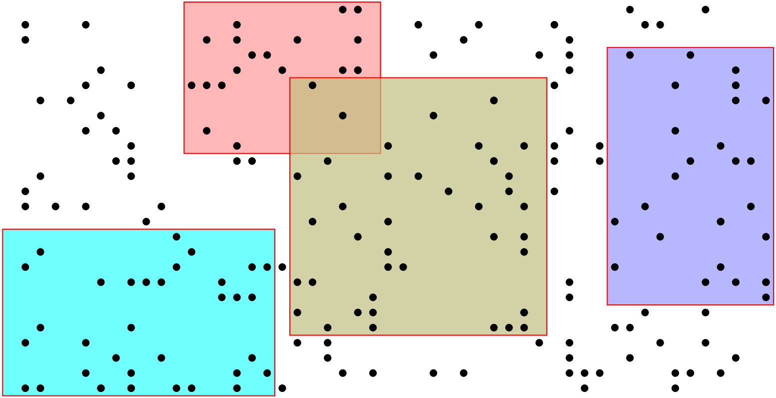
# Set Cover



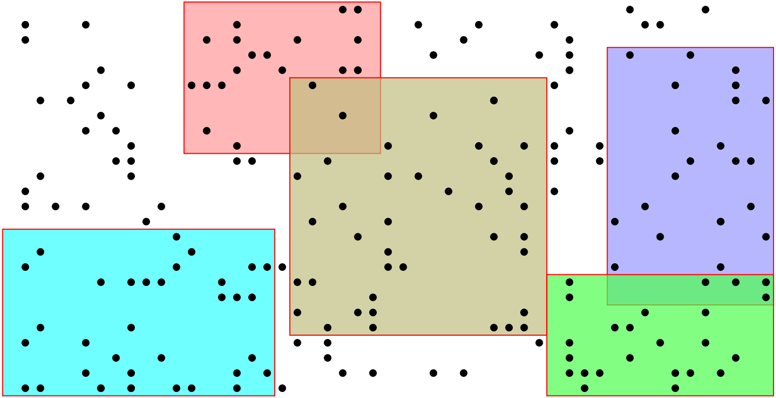
# Set Cover



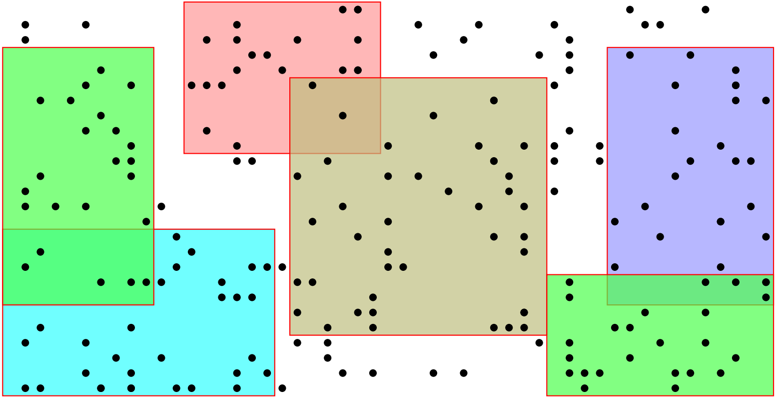
# Set Cover



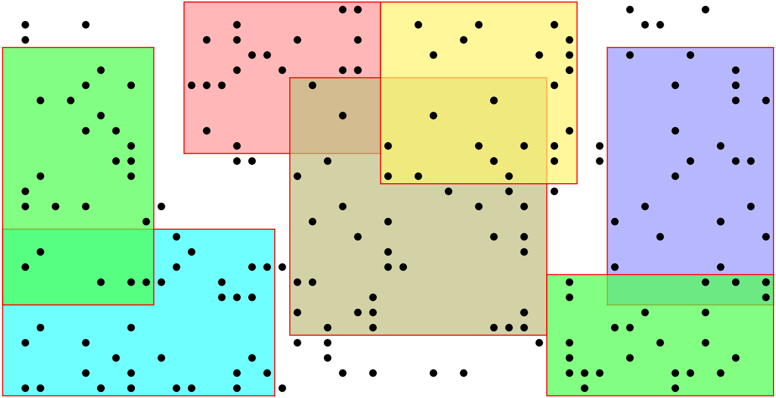
# Set Cover



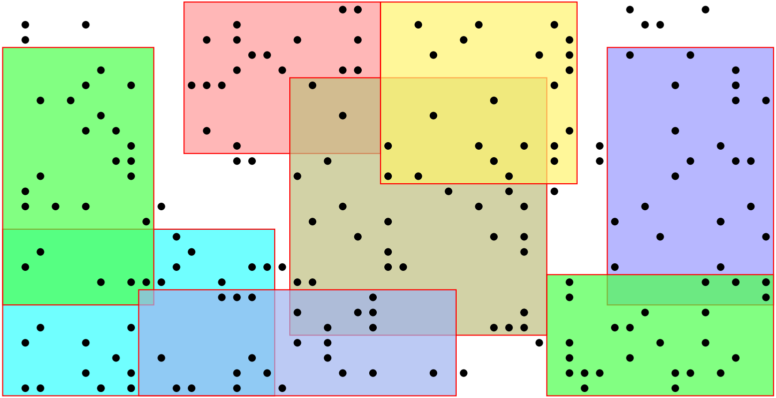
# Set Cover



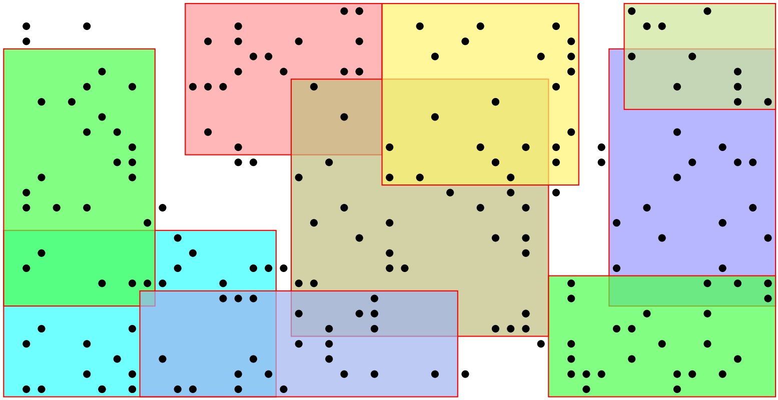
# Set Cover



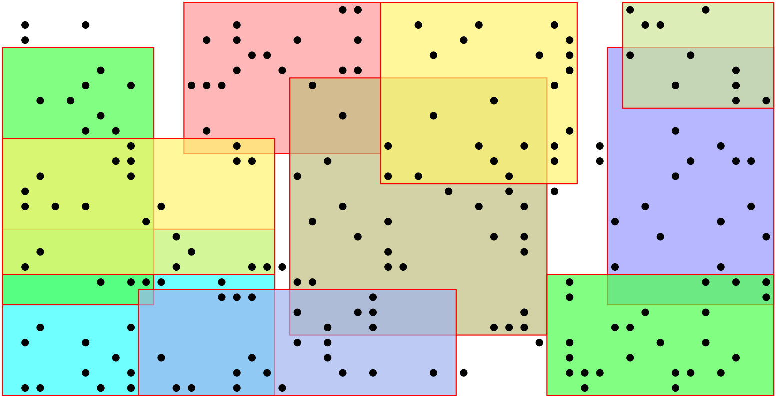
# Set Cover



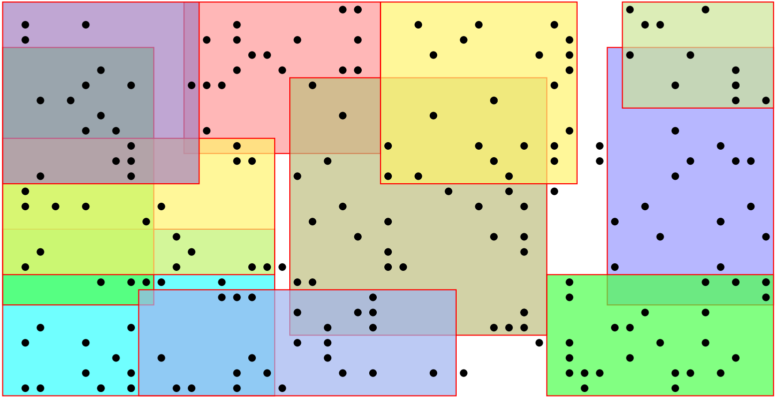
# Set Cover



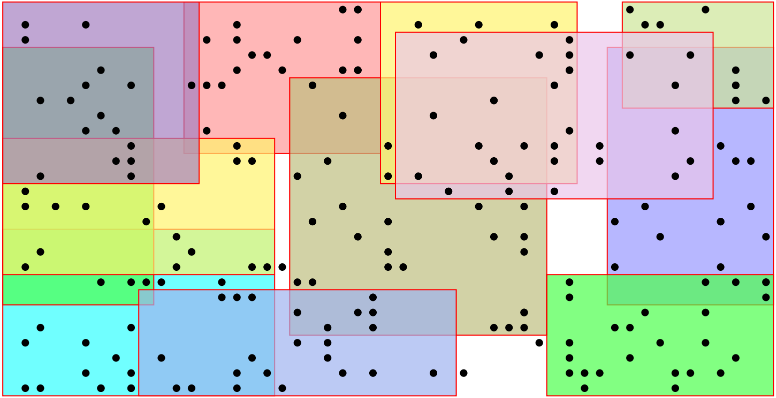
# Set Cover



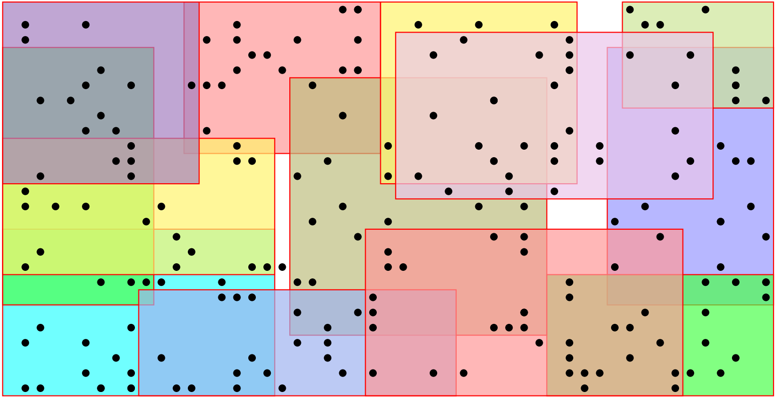
# Set Cover



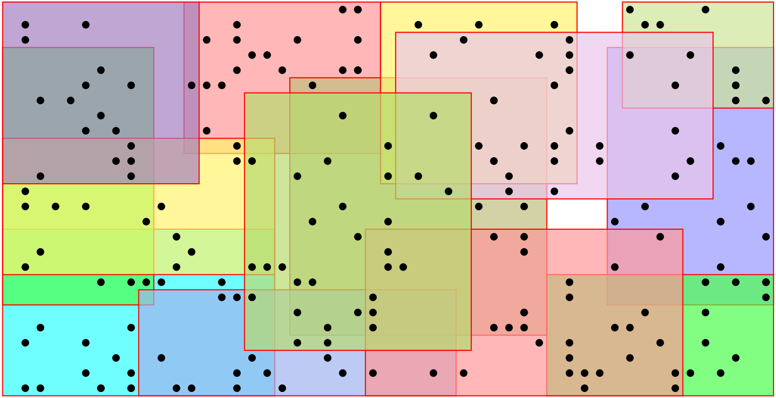
# Set Cover



# Set Cover



# Set Cover



# IP-Formulation of Set Cover

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

# Vertex Cover

Given a graph $G = (V, E)$ and a weight $w_v$ for every node. Find a vertex subset $S \subseteq V$ of minimum weight such that every edge is incident to at least one vertex in $S$.

# IP-Formulation of Vertex Cover

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

# Maximum Weighted Matching

Given a graph $G = (V, E)$, and a weight $w_e$ for every edge $e \in E$. Find a subset of edges of maximum weight such that no vertex is incident to more than one edge.

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

# Maximum Weighted Matching

Given a graph $G = (V, E)$, and a weight $w_e$ for every edge $e \in E$. Find a subset of edges of maximum weight such that no vertex is incident to more than one edge.

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

# Maximum Independent Set

Given a graph $G = (V, E)$, and a weight $w_v$ for every node $v \in V$. Find a subset $S \subseteq V$ of nodes of maximum weight such that no two vertices in $S$ are adjacent.

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

# Maximum Independent Set

Given a graph $G = (V, E)$, and a weight $w_v$ for every node $v \in V$. Find a subset $S \subseteq V$ of nodes of maximum weight such that no two vertices in $S$ are adjacent.

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

# Knapsack

Given a set of items $\{1,\dots,n\}$, where the $i$-th item has weight $w_i$ and profit $p_i$, and given a threshold $K$. Find a subset $I \subseteq \{1,\dots,n\}$ of items of total weight at most $K$ such that the profit is maximized.

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

# Knapsack

Given a set of items $\{1, \ldots, n\}$, where the $i$-th item has weight $w_i$ and profit $p_i$, and given a threshold $K$. Find a subset $I \subseteq \{1, \ldots, n\}$ of items of total weight at most $K$ such that the profit is maximized.

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

# Relaxations

**Definition 12**
A linear program LP is a relaxation of an integer program IP if any feasible solution for IP is also feasible for LP and if the objective values of these solutions are identical in both programs.

We obtain a relaxation for all examples by writing $x_i \in [0,1]$ instead of $x_i \in \{0,1\}$.

# Relaxations

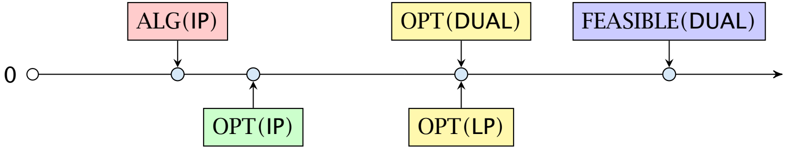
**Definition 12**
A linear program LP is a relaxation of an integer program IP if any feasible solution for IP is also feasible for LP and if the objective values of these solutions are identical in both programs.

We obtain a relaxation for all examples by writing $x_i \in [0,1]$ instead of $x_i \in \{0,1\}$.

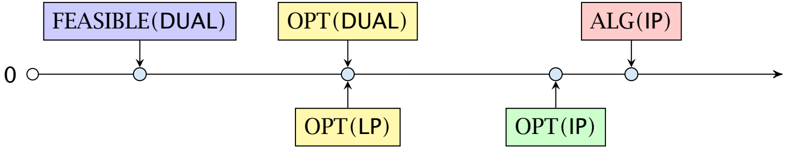
By solving a relaxation we obtain an upper bound for a maximization problem and a lower bound for a minimization problem.

# Relations

**Maximization Problems:**




**Minimization Problems:**



# Technique 1: Round the LP solution.

We first solve the LP-relaxation and then we round the fractional values so that we obtain an integral solution.

**Set Cover relaxation:**

| | | | | |
|---|---|---|---|---|
| min | | $\sum_{i=1}^{k} w_i x_i$ | | |
| s.t. | $\forall u \in U$ | $\sum_{i:u\in S_i} x_i$ | $\geq$ | $1$ |
| | $\forall i \in \{1,\dots,k\}$ | $x_i$ | $\in$ | $[0,1]$ |

Let $f_u$ be the number of sets that the element $u$ is contained in (the frequency of $u$). Let $f = \max_u\{f_u\}$ be the maximum frequency.

# Technique 1: Round the LP solution.

We first solve the LP-relaxation and then we round the fractional values so that we obtain an integral solution.

**Set Cover relaxation:**

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

Let $f_u$ be the number of sets that the element $u$ is contained in (the frequency of $u$). Let $f = \max_u \{f_u\}$ be the maximum frequency.

# Technique 1: Round the LP solution.

We first solve the LP-relaxation and then we round the fractional values so that we obtain an integral solution.

**Set Cover relaxation:**

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

Let $f_u$ be the number of sets that the element $u$ is contained in (the frequency of $u$). Let $f = \max_u\{f_u\}$ be the maximum frequency.

# Technique 1: Round the LP solution.

**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 13**

*The rounding algorithm gives an $f$-approximation.*

**Proof:** Every $u \in U$ is covered.

- We know that $\sum_{j: u \in S_j} x_j \geq 1$.
- The sum contains at most $f_u \leq f$ elements.
- Therefore one of the sets that contain $u$ must have $x_j \geq 1/f$.
- This set will be selected. Hence, $u$ is covered.

# Technique 1: Round the LP solution.

**Lemma 13**

*The rounding algorithm gives an $f$-approximation.*

**Proof:** Every $u \in U$ is covered.

- We know that $\sum_{i: u \in S_i} x_i \geq 1$.
- The sum contains at most $f_u \leq f$ elements.
- Therefore one of the sets that contain $u$ must have $x_i \geq 1/f$.
- This set will be selected. Hence, $u$ is covered.

# Technique 1: Round the LP solution.

**Lemma 13**

*The rounding algorithm gives an $f$-approximation.*

**Proof:** Every $u \in U$ is covered.

- We know that $\sum_{i:u \in S_i} x_i \geq 1$.
- The sum contains at most $f_u \leq f$ elements.
- Therefore one of the sets that contain $u$ must have $x_i \geq 1/f$.
- This set will be selected. Hence, $u$ is covered.

# Technique 1: Round the LP solution.

**Lemma 13**

*The rounding algorithm gives an $f$-approximation.*

**Proof:** Every $u \in U$ is covered.

- We know that $\sum_{i:u\in S_i} x_i \geq 1$.
- The sum contains at most $f_u \leq f$ elements.
- Therefore one of the sets that contain $u$ must have $x_i \geq 1/f$.
- This set will be selected. Hence, $u$ is covered.

# Technique 1: Round the LP solution.

**Lemma 13**

*The rounding algorithm gives an $f$-approximation.*

**Proof:** Every $u \in U$ is covered.

- We know that $\sum_{i:u\in S_i} x_i \geq 1$.
- The sum contains at most $f_u \leq f$ elements.
- Therefore one of the sets that contain $u$ must have $x_i \geq 1/f$.
- This set will be selected. Hence, $u$ is covered.

# Technique 1: Round the LP solution.

## Lemma 13

*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 $f \cdot \text{OPT}$.

# Technique 1: Round the LP solution.

The cost of the rounded solution is at most $f \cdot \mathrm{OPT}$.

$$\sum_{i \in I} w_i$$

# Technique 1: Round the LP solution.

The cost of the rounded solution is at most $f \cdot \mathrm{OPT}$.

$$\sum_{i \in I} w_i \leq \sum_{i=1}^{k} w_i (f \cdot x_i)$$

# Technique 1: Round the LP solution.

The cost of the rounded solution is at most $f \cdot \mathrm{OPT}$.

$$\sum_{i \in I} w_i \le \sum_{i=1}^{k} w_i (f \cdot x_i) = f \cdot \mathrm{cost}(x)$$

# Technique 1: Round the LP solution.

The cost of the rounded solution is at most $f \cdot \text{OPT}$.

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

# Technique 2: Rounding the Dual Solution.

**Relaxation for Set Cover**

**Primal:**

$$\begin{array}{lll} \min & \sum_{i\in I} w_i x_i & \\ \text{s.t. } \forall u & \sum_{i:u\in S_i} x_i & \geq 1 \\ & x_i & \geq 0 \end{array}$$

**Dual:**

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

# Technique 2: Rounding the Dual Solution.

**Relaxation for Set Cover**

**Primal:**

$$\begin{array}{lrl} \min & \sum_{i\in I} w_i x_i & \\ \text{s.t. } \forall u & \sum_{i:u\in S_i} x_i & \geq 1 \\ & x_i & \geq 0 \end{array}$$

**Dual:**

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

# Technique 2: Rounding the Dual Solution.

**Relaxation for Set Cover**

**Primal:**

$$\begin{array}{lrl} \min & \sum_{i \in I} w_i x_i & \\ \text{s.t. } \forall u & \sum_{i:u \in S_i} x_i & \geq 1 \\ & x_i & \geq 0 \end{array}$$

**Dual:**

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

# Technique 2: Rounding the Dual Solution.

**Rounding Algorithm:**

Let $I$ denote the index set of sets for which the dual constraint is tight. This means for all $i \in I$

$$\sum_{u: u \in S_i} y_u = w_i$$

# Technique 2: Rounding the Dual Solution.

**Lemma 14**

*The resulting index set is an $f$-approximation.*

**Proof:**

Every $u \in U$ is covered.

- Suppose there is a $u$ that is not covered.
- This means $\sum_{u : u \in S_j} y_u < w_j$ for all sets $S_j$ that contain $u$.
- But then $y_u$ could be increased in the dual solution without violating any constraint. This is a contradiction to the fact that the dual solution is optimal.

# Technique 2: Rounding the Dual Solution.

**Lemma 14**

*The resulting index set is an $f$-approximation.*

**Proof:**

Every $u \in U$ is covered.

- Suppose there is a $u$ that is not covered.
- This means $\sum_{u: u \in S_i} y_u < w_i$ for all sets $S_i$ that contain $u$.
- But then $y_u$ could be increased in the dual solution without violating any constraint. This is a contradiction to the fact that the dual solution is optimal.

# Technique 2: Rounding the Dual Solution.

**Lemma 14**

*The resulting index set is an $f$-approximation.*

**Proof:**

Every $u \in U$ is covered.

- Suppose there is a $u$ that is not covered.
- This means $\sum_{u:u\in S_i} y_u < w_i$ for all sets $S_i$ that contain $u$.
- But then $y_u$ could be increased in the dual solution without violating any constraint. This is a contradiction to the fact that the dual solution is optimal.

# Technique 2: Rounding the Dual Solution.

**Lemma 14**

*The resulting index set is an $f$-approximation.*

**Proof:**

Every $u \in U$ is covered.

- Suppose there is a $u$ that is not covered.
- This means $\sum_{u:u\in S_i} y_u < w_i$ for all sets $S_i$ that contain $u$.
- But then $y_u$ could be increased in the dual solution without violating any constraint. This is a contradiction to the fact that the dual solution is optimal.

# Technique 2: Rounding the Dual Solution.

**Lemma 14**

*The resulting index set is an $f$-approximation.*

**Proof:**

Every $u \in U$ is covered.

- Suppose there is a $u$ that is not covered.
- This means $\sum_{u:u\in S_i} y_u < w_i$ for all sets $S_i$ that contain $u$.
- But then $y_u$ could be increased in the dual solution without violating any constraint. This is a contradiction to the fact that the dual solution is optimal.

# Technique 2: Rounding the Dual Solution.

**Proof:**

$$\sum_{i \in I} w_i$$

# Technique 2: Rounding the Dual Solution.

**Proof:**

$$\sum_{i \in I} w_i = \sum_{i \in I} \sum_{u : u \in S_i} y_u$$

# Technique 2: Rounding the Dual Solution.

**Proof:**

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

# Technique 2: Rounding the Dual Solution.

**Proof:**

$$\sum_{i\in I} w_i = \sum_{i\in I}\sum_{u:u\in S_i} y_u$$

$$= \sum_u |\{i \in I : u \in S_i\}| \cdot y_u$$

$$\le \sum_u f_u y_u$$

# Technique 2: Rounding the Dual Solution.

**Proof:**

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

# Technique 2: Rounding the Dual Solution.

**Proof:**

$$\begin{aligned}\sum_{i\in I} w_i &= \sum_{i\in I}\sum_{u:u\in S_i} y_u \\ &= \sum_u |\{i\in I : u\in S_i\}|\cdot y_u \\ &\le \sum_u f_u y_u \\ &\le f\sum_u y_u \\ &\le f\,\mathrm{cost}(x^*)\end{aligned}$$

# Technique 2: Rounding the Dual Solution.

**Proof:**

$$\begin{aligned}\sum_{i\in I} w_i &= \sum_{i\in I}\sum_{u:u\in S_i} y_u\\ &= \sum_u |\{i\in I : u\in S_i\}|\cdot y_u\\ &\le \sum_u f_u y_u\\ &\le f\sum_u y_u\\ &\le f\,\mathrm{cost}(x^*)\\ &\le f\cdot \mathrm{OPT}\end{aligned}$$

Let $I$ denote the solution obtained by the first rounding algorithm and $I'$ be the solution returned by the second algorithm. Then

$$I \subseteq I' \ .$$

This means $I'$ is never better than $I$.

Suppose that we take $S_i$ in the first algorithm. I.e., $i \in I$.

This means $x_i \geq \frac{1}{f}$.

Because of Complementary Slackness Conditions the corresponding constraint in the dual must be tight.

Hence, the second algorithm will also choose $S_i$.

Let $I$ denote the solution obtained by the first rounding algorithm and $I'$ be the solution returned by the second algorithm. Then

$$I \subseteq I' \ .$$

This means $I'$ is never better than $I$.

- Suppose that we take $S_i$ in the first algorithm. I.e., $i \in I$.
- This means $x_i \geq \frac{1}{f}$.
- Because of Complementary Slackness Conditions the corresponding constraint in the dual must be tight.
- Hence, the second algorithm will also choose $S_i$.

Let $I$ denote the solution obtained by the first rounding algorithm and $I'$ be the solution returned by the second algorithm. Then

$$I \subseteq I' \ .$$

This means $I'$ is never better than $I$.

- Suppose that we take $S_i$ in the first algorithm. I.e., $i \in I$.
- This means $x_i \geq \frac{1}{f}$.
- Because of Complementary Slackness Conditions the corresponding constraint in the dual must be tight.
- Hence, the second algorithm will also choose $S_i$.

Let $I$ denote the solution obtained by the first rounding algorithm and $I'$ be the solution returned by the second algorithm. Then

$$I \subseteq I' \ .$$

This means $I'$ is never better than $I$.

- Suppose that we take $S_i$ in the first algorithm. I.e., $i \in I$.
- This means $x_i \geq \frac{1}{f}$.
- Because of Complementary Slackness Conditions the corresponding constraint in the dual must be tight.
- Hence, the second algorithm will also choose $S_i$.

Let $I$ denote the solution obtained by the first rounding algorithm and $I'$ be the solution returned by the second algorithm. Then

$$I \subseteq I' \ .$$

This means $I'$ is never better than $I$.

- Suppose that we take $S_i$ in the first algorithm. I.e., $i \in I$.
- This means $x_i \geq \frac{1}{f}$.
- Because of Complementary Slackness Conditions the corresponding constraint in the dual must be tight.
- Hence, the second algorithm will also choose $S_i$.

# Technique 3: The Primal Dual Method

The previous two rounding algorithms have the disadvantage that it is necessary to solve the LP. The following method also gives an $f$-approximation without solving the LP.

For estimating the cost of the solution we only required two properties.

- The solution is dual feasible and, hence,

$$\sum_u y_u \le \text{cost}(x^*) \le \text{OPT}$$

  where $x^*$ is an optimum solution to the primal LP.

- The set $I$ contains only sets for which the dual inequality is tight.

Of course, we also need that $I$ is a cover.

# Technique 3: The Primal Dual Method

The previous two rounding algorithms have the disadvantage that it is necessary to solve the LP. The following method also gives an $f$-approximation without solving the LP.

For estimating the cost of the solution we only required two properties.

- The solution is dual feasible and, hence,

$$\sum_u y_u \le \mathrm{cost}(x^*) \le \mathrm{OPT}$$

  where $x^*$ is an optimum solution to the primal LP.

- The set $I$ contains only sets for which the dual inequality is tight.

Of course, we also need that $I$ is a cover.

# Technique 3: The Primal Dual Method

The previous two rounding algorithms have the disadvantage that it is necessary to solve the LP. The following method also gives an $f$-approximation without solving the LP.

For estimating the cost of the solution we only required two properties.

1. The solution is dual feasible and, hence,

$$\sum_u y_u \leq \text{cost}(x^*) \leq \text{OPT}$$

   where $x^*$ is an optimum solution to the primal LP.

2. The set $I$ contains only sets for which the dual inequality is tight.

Of course, we also need that $I$ is a cover.

# Technique 3: The Primal Dual Method

The previous two rounding algorithms have the disadvantage that it is necessary to solve the LP. The following method also gives an $f$-approximation without solving the LP.

For estimating the cost of the solution we only required two properties.

1. The solution is dual feasible and, hence,

$$\sum_u y_u \leq \text{cost}(x^*) \leq \text{OPT}$$

   where $x^*$ is an optimum solution to the primal LP.

2. The set $I$ contains only sets for which the dual inequality is tight.

Of course, we also need that $I$ is a cover.

# Technique 3: The Primal Dual Method

The previous two rounding algorithms have the disadvantage that it is necessary to solve the LP. The following method also gives an $f$-approximation without solving the LP.

For estimating the cost of the solution we only required two properties.

1. The solution is dual feasible and, hence,

$$\sum_u y_u \leq \text{cost}(x^*) \leq \text{OPT}$$

   where $x^*$ is an optimum solution to the primal LP.

2. The set $I$ contains only sets for which the dual inequality is tight.

Of course, we also need that $I$ is a cover.

# Technique 3: The Primal Dual Method

**Algorithm 1** PrimalDual

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

# Technique 4: The Greedy Algorithm

**Algorithm 1** Greedy

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

In every round the Greedy algorithm takes the set that covers remaining elements in the most cost-effective way.

We choose a set such that the ratio between cost and still uncovered elements in the set is minimized.

# Technique 4: The Greedy Algorithm

**Lemma 15**

*Given positive numbers* $a_1, \ldots, a_k$ *and* $b_1, \ldots, b_k$*, and* $S \subseteq \{1, \ldots, k\}$ *then*

$$\min_i \frac{a_i}{b_i} \le \frac{\sum_{i \in S} a_i}{\sum_{i \in S} b_i} \le \max_i \frac{a_i}{b_i}$$

# Technique 4: The Greedy Algorithm

Let $n_\ell$ 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

$$\min_j \frac{w_j}{|\hat{S}_j|} \le \frac{\sum_{j\in\mathrm{OPT}} w_j}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} = \frac{\mathrm{OPT}}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} \le \frac{\mathrm{OPT}}{n_\ell}$$

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

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

$$\min_j \frac{w_j}{|\hat{S}_j|} \leq \frac{\sum_{j\in\mathrm{OPT}} w_j}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} = \frac{\mathrm{OPT}}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} \leq \frac{\mathrm{OPT}}{n_\ell}$$

since an optimal algorithm can cover the remaining $n_\ell$ elements with cost OPT.

Let $\hat{S}_j$ be a subset that minimizes this ratio. Hence, $w_j/|\hat{S}_j| \leq \frac{\mathrm{OPT}}{n_\ell}$.

# Technique 4: The Greedy Algorithm

Let $n_\ell$ 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

$$\min_j \frac{w_j}{|\hat{S}_j|} \leq \frac{\sum_{j\in\mathrm{OPT}} w_j}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} = \frac{\mathrm{OPT}}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} \leq \frac{\mathrm{OPT}}{n_\ell}$$

since an optimal algorithm can cover the remaining $n_\ell$ elements with cost OPT.

Let $\hat{S}_j$ be a subset that minimizes this ratio. Hence, $w_j/|\hat{S}_j| \leq \frac{\mathrm{OPT}}{n_\ell}$.

# Technique 4: The Greedy Algorithm

Let $n_\ell$ 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

$$\min_j \frac{w_j}{|\hat{S}_j|} \le \frac{\sum_{j\in\mathrm{OPT}} w_j}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} = \frac{\mathrm{OPT}}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} \le \frac{\mathrm{OPT}}{n_\ell}$$

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

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

$$\min_j \frac{w_j}{|\hat{S}_j|} \le \frac{\sum_{j\in\mathrm{OPT}} w_j}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} = \frac{\mathrm{OPT}}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} \le \frac{\mathrm{OPT}}{n_\ell}$$

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

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

$$\min_j \frac{w_j}{|\hat{S}_j|} \leq \frac{\sum_{j\in\mathrm{OPT}} w_j}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} = \frac{\mathrm{OPT}}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} \leq \frac{\mathrm{OPT}}{n_\ell}$$

since an optimal algorithm can cover the remaining $n_\ell$ elements with cost OPT.

Let $\hat{S}_j$ be a subset that minimizes this ratio. Hence, $w_j/|\hat{S}_j| \leq \frac{\mathrm{OPT}}{n_\ell}$.

# Technique 4: The Greedy Algorithm

Let $n_\ell$ 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

$$\min_j \frac{w_j}{|\hat{S}_j|} \leq \frac{\sum_{j\in\mathrm{OPT}} w_j}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} = \frac{\mathrm{OPT}}{\sum_{j\in\mathrm{OPT}} |\hat{S}_j|} \leq \frac{\mathrm{OPT}}{n_\ell}$$

since an optimal algorithm can cover the remaining $n_\ell$ elements with cost OPT.

Let $\hat{S}_j$ be a subset that minimizes this ratio. Hence, $w_j/|\hat{S}_j| \leq \frac{\mathrm{OPT}}{n_\ell}$.

# Technique 4: The Greedy Algorithm

Adding this set to our solution means $n_{\ell+1} = n_\ell - |\hat{S}_j|$.

$$w_j \le \frac{|\hat{S}_j|\mathrm{OPT}}{n_\ell} = \frac{n_\ell - n_{\ell+1}}{n_\ell} \cdot \mathrm{OPT}$$

# Technique 4: The Greedy Algorithm

Adding this set to our solution means $n_{\ell+1} = n_\ell - |\hat{S}_j|$.

$$w_j \le \frac{|\hat{S}_j|\mathrm{OPT}}{n_\ell} = \frac{n_\ell - n_{\ell+1}}{n_\ell} \cdot \mathrm{OPT}$$

# Technique 4: The Greedy Algorithm

$$\sum_{j \in I} w_j$$

# Technique 4: The Greedy Algorithm

$$\sum_{j \in I} w_j \leq \sum_{\ell=1}^{s} \frac{n_\ell - n_{\ell+1}}{n_\ell} \cdot \mathrm{OPT}$$

# Technique 4: The Greedy Algorithm

$$\sum_{j\in I} w_j \le \sum_{\ell=1}^{s} \frac{n_\ell - n_{\ell+1}}{n_\ell} \cdot \mathrm{OPT}$$

$$\le \mathrm{OPT} \sum_{\ell=1}^{s} \left( \frac{1}{n_\ell} + \frac{1}{n_\ell - 1} + \dots + \frac{1}{n_{\ell+1} + 1} \right)$$

# Technique 4: The Greedy Algorithm

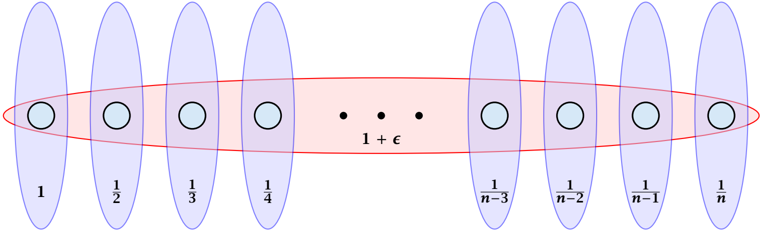
$$
\begin{aligned}
\sum_{j\in I} w_j &\le \sum_{\ell=1}^{s} \frac{n_\ell - n_{\ell+1}}{n_\ell} \cdot \mathrm{OPT} \\
&\le \mathrm{OPT} \sum_{\ell=1}^{s} \left( \frac{1}{n_\ell} + \frac{1}{n_\ell - 1} + \cdots + \frac{1}{n_{\ell+1} + 1} \right) \\
&= \mathrm{OPT} \sum_{i=1}^{k} \frac{1}{i}
\end{aligned}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

# Technique 4: The Greedy Algorithm

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

# Technique 4: The Greedy Algorithm

**A tight example:**



# Technique 5: Randomized Rounding

One round of randomized rounding:
Pick set $S_j$ uniformly at random with probability $1 - x_j$ (for all $j$).

**Version A:** Repeat rounds until you have a cover.

**Version B:** Repeat for $s$ rounds. If you have a cover STOP. Otherwise, repeat the whole algorithm.

# Technique 5: Randomized Rounding

One round of randomized rounding:
Pick set $S_j$ uniformly at random with probability $1 - x_j$ (for all $j$).

**Version A:** Repeat rounds until you have a cover.

**Version B:** Repeat for $s$ rounds. If you have a cover STOP. Otherwise, repeat the whole algorithm.

# Technique 5: Randomized Rounding

One round of randomized rounding:
Pick set $S_j$ uniformly at random with probability $1 - x_j$ (for all $j$).

**Version A:** Repeat rounds until you have a cover.

**Version B:** Repeat for $s$ rounds. If you have a cover STOP.
Otherwise, repeat the whole algorithm.

**Probability that $u \in U$ is not covered (in one round):**

$$\Pr[u \text{ not covered in one round}]$$

**Probability that $u \in U$ is not covered (in one round):**

$$\Pr[u \text{ not covered in one round}] = \prod_{j:u\in S_j} (1 - x_j)$$

**Probability that $u \in U$ is not covered (in one round):**

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

**Probability that $u \in U$ is not covered (in one round):**

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

**Probability that $u \in U$ is not covered (in one round):**

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

**Probability that $u \in U$ is not covered (in one round):**

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

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

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

$\Pr[\exists u \in U \text{ not covered after } \ell \text{ round}]$

$$
\begin{aligned}
&\Pr[\exists u \in U \text{ not covered after } \ell \text{ round}] \\
&\qquad = \Pr[u_1 \text{ not covered} \vee u_2 \text{ not covered} \vee \ldots \vee u_n \text{ not covered}]
\end{aligned}
$$

$$\Pr[\exists u \in U \text{ not covered after } \ell \text{ round}]$$
$$= \Pr[u_1 \text{ not covered} \vee u_2 \text{ not covered} \vee \ldots \vee u_n \text{ not covered}]$$
$$\leq \sum_i \Pr[u_i \text{ not covered after } \ell \text{ rounds}]$$

$$\Pr[\exists u \in U \text{ not covered after } \ell \text{ round}]$$
$$= \Pr[u_1 \text{ not covered} \vee u_2 \text{ not covered} \vee \ldots \vee u_n \text{ not covered}]$$
$$\leq \sum_i \Pr[u_i \text{ not covered after } \ell \text{ rounds}] \leq ne^{-\ell} \ .$$

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

**Lemma 16**

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

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

**Lemma 16**

*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

$$\Pr[\#\text{rounds} \geq (\alpha+1)\ln n] \leq ne^{-(\alpha+1)\ln n} = n^{-\alpha} \;.$$

# Expected Cost

- Version A.
  Repeat for $s = (\alpha + 1)\ln n$ rounds. If you don't have a cover simply take for each element $u$ the cheapest set that contains $u$.

# Expected Cost

- Version A.
  Repeat for $s = (\alpha + 1)\ln n$ rounds. If you don't have a cover simply take for each element $u$ the cheapest set that contains $u$.

  $E[\text{cost}]$

# Expected Cost

- Version A.
  Repeat for $s = (\alpha + 1)\ln n$ rounds. If you don't have a cover simply take for each element $u$ the cheapest set that contains $u$.

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

# Expected Cost

- Version A.
  Repeat for $s = (\alpha + 1)\ln n$ rounds. If you don't have a cover simply take for each element $u$ the cheapest set that contains $u$.

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

# Expected Cost

- Version B.
  Repeat for $s = (\alpha + 1)\ln n$ rounds. If you don't have a cover simply repeat the whole process.

$$E[\text{cost}] =$$

# Expected Cost

- Version B.
  Repeat for $s = (\alpha + 1)\ln n$ rounds. If you don't have a cover simply repeat the whole process.

$$
\begin{aligned}
E[\text{cost}] = \Pr[\text{success}] \cdot E[\text{cost} \mid \text{success}] \\
+ \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}]
\end{aligned}
$$

# Expected Cost

- Version B.
  Repeat for $s = (\alpha + 1)\ln n$ rounds. If you don't have a cover simply repeat the whole process.

$$
\begin{aligned}
E[\text{cost}] = \Pr[\text{success}] \cdot E[\text{cost} \mid \text{success}] \\
+ \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}]
\end{aligned}
$$

This means

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

# Expected Cost

- Version B.
  Repeat for $s = (\alpha + 1)\ln n$ rounds. If you don't have a cover simply repeat the whole process.

$$E[\text{cost}] = \Pr[\text{success}] \cdot E[\text{cost} \mid \text{success}] + \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}]$$

This means

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

# Expected Cost

- Version B.
  Repeat for $s = (\alpha + 1)\ln n$ rounds. If you don't have a cover simply repeat the whole process.

$$E[\text{cost}] = \Pr[\text{success}] \cdot E[\text{cost} \mid \text{success}] + \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}]$$

This means

$$\begin{aligned} & E[\text{cost} \mid \text{success}] \\ & \quad = \frac{1}{\Pr[\text{succ.}]}\Big(E[\text{cost}] - \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}]\Big) \\ & \quad \le \frac{1}{\Pr[\text{succ.}]} E[\text{cost}] \le \frac{1}{1 - n^{-\alpha}} (\alpha + 1) \ln n \cdot \text{cost}(LP) \end{aligned}$$

# Expected Cost

- Version B.
  Repeat for $s = (\alpha + 1)\ln n$ rounds. If you don't have a cover simply repeat the whole process.

$$\begin{aligned} E[\text{cost}] = \Pr[\text{success}] \cdot E[\text{cost} \mid \text{success}] \\ + \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}] \end{aligned}$$

This means

$$\begin{aligned} &E[\text{cost} \mid \text{success}] \\ &= \frac{1}{\Pr[\text{succ.}]}\Big(E[\text{cost}] - \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}]\Big) \\ &\le \frac{1}{\Pr[\text{succ.}]} E[\text{cost}] \le \frac{1}{1 - n^{-\alpha}} (\alpha + 1)\ln n \cdot \text{cost}(\text{LP}) \\ &\le 2(\alpha + 1)\ln n \cdot \text{OPT} \end{aligned}$$

# Expected Cost

- Version B.
  Repeat for $s = (\alpha + 1)\ln n$ rounds. If you don't have a cover simply repeat the whole process.

$$E[\text{cost}] = \Pr[\text{success}] \cdot E[\text{cost} \mid \text{success}] + \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}]$$

This means

$$\begin{aligned} E[\text{cost} \mid \text{success}] &= \frac{1}{\Pr[\text{succ.}]}\Big(E[\text{cost}] - \Pr[\text{no success}] \cdot E[\text{cost} \mid \text{no success}]\Big) \\ &\le \frac{1}{\Pr[\text{succ.}]} E[\text{cost}] \le \frac{1}{1 - n^{-\alpha}} (\alpha + 1)\ln n \cdot \text{cost(LP)} \\ &\le 2(\alpha + 1)\ln n \cdot \text{OPT} \end{aligned}$$

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

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

**Theorem 17 (without proof)**

*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 $2^{\mathrm{poly}(\log n)}$).*

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

**Theorem 17 (without proof)**

*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 $2^{\mathrm{poly}(\log n)}$).*

# Integrality Gap

The integrality gap of the SetCover LP is $\Omega(\log n)$.

- $n = 2^k - 1$
- Elements are all vectors $\boldsymbol{i}$ over $GF[2]$ of length $k$ (excluding zero vector).
- Every vector $\boldsymbol{j}$ defines a set as follows

$$S_{\boldsymbol{j}} := \{\boldsymbol{i} \mid \boldsymbol{i} \cdot \boldsymbol{j} = 1\}$$

- each set contains $2^{k-1}$ vectors; each vector is contained in $2^{k-1}$ sets
- $x_i = \frac{1}{2^{k-1}} = \frac{2}{n+1}$ is fractional solution.

# Integrality Gap

Every collection of $p < k$ sets does not cover all elements.

Hence, we get a gap of $\Omega(\log n)$.

**Techniques:**

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

# Scheduling Jobs on Identical Parallel Machines

Given $n$ jobs, where job $j \in \{1, \ldots, n\}$ has processing time $p_j$. Schedule the jobs on $m$ identical parallel machines such that the Makespan (finishing time of the last job) is minimized.

$$
\begin{array}{llrcl}
\min & & & & L \\
\text{s.t.} & \forall \text{machines } i & \sum_j p_j \cdot x_{j,i} & \leq & L \\
& \forall \text{jobs } j & \sum_i x_{j,i} \geq 1 & & \\
& \forall i, j & x_{j,i} & \in & \{0, 1\}
\end{array}
$$

Here the variable $x_{j,i}$ is the decision variable that describes whether job $j$ is assigned to machine $i$.

# 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

Let for a given schedule $C_j$ denote the finishing time of machine $j$, and let $C_{\max}$ be the makespan.

Let $C^*_{\max}$ denote the makespan of an optimal solution.

Clearly

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

as the longest job needs to be scheduled somewhere.

# Lower Bounds on the Solution

Let for a given schedule $C_j$ denote the finishing time of machine $j$, and let $C_{\max}$ be the makespan.

Let $C^*_{\max}$ denote the makespan of an optimal solution.

Clearly

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

as the longest job needs to be scheduled somewhere.

# Lower Bounds on the Solution

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

Therefore,

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

# Lower Bounds on the Solution

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

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

# Local Search

A local search algorithm successivley makes certain small (cost/profit improving) changes to a solution until it does not find such changes anymore.

It is conceptionally very different from a Greedy algorithm as a feasible solution is always maintained.

Sometimes the running time is difficult to prove.

# Local Search

A local search algorithm successivley makes certain small (cost/profit improving) changes to a solution until it does not find such changes anymore.

It is conceptionally very different from a Greedy algorithm as a feasible solution is always maintained.

Sometimes the running time is difficult to prove.

# Local Search

A local search algorithm successivley makes certain small (cost/profit improving) changes to a solution until it does not find such changes anymore.

It is conceptionally very different from a Greedy algorithm as a feasible solution is always maintained.

Sometimes the running time is difficult to prove.

# Local Search

A local search algorithm successivley makes certain small (cost/profit improving) changes to a solution until it does not find such changes anymore.

It is conceptionally very different from a Greedy algorithm as a feasible solution is always maintained.

Sometimes the running time is difficult to prove.

# Local Search for Scheduling

**Local Search Strategy:** Take the job that finishes last and try to move it to another machine. If there is such a move that reduces the makespan, perform the switch.

REPEAT

# Local Search for Scheduling

**Local Search Strategy:** Take the job that finishes last and try to move it to another machine. If there is such a move that reduces the makespan, perform the switch.

REPEAT

# Local Search for Scheduling

**Local Search Strategy:** Take the job that finishes last and try to move it to another machine. If there is such a move that reduces the makespan, perform the switch.

REPEAT

# Local Search Analysis

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

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

# Local Search Analysis

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

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

# Local Search Analysis

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

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

# Local Search Analysis

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

Let $S_\ell$ be its start time, and let $C_\ell$ be 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 processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \le \sum_{j \ne \ell} p_j \ .$$

Hence, the length of the schedule is at most

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

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 processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \le \sum_{j \ne \ell} p_j \enspace .$$

Hence, the length of the schedule is at most

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

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 $p_\ell \leq C^*_{\max}$.

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \leq \sum_{j \neq \ell} p_j \ .$$

Hence, the length of the schedule is at most

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

We can split the total processing time into two intervals one from $0$ to $S_\ell$ 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 processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \le \sum_{j \ne \ell} p_j \ .$$

Hence, the length of the schedule is at most

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

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 processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \le \sum_{j \neq \ell} p_j \enspace .$$

Hence, the length of the schedule is at most

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

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

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

During the first interval $[0, S_\ell]$ all processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \leq \sum_{j \neq \ell} p_j \enspace .$$

Hence, the length of the schedule is at most

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

We can split the total processing time into two intervals one from $0$ to $S_\ell$ 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 processors are busy, and, hence, the total work performed in this interval is

$$m \cdot S_\ell \le \sum_{j \ne \ell} p_j \ .$$

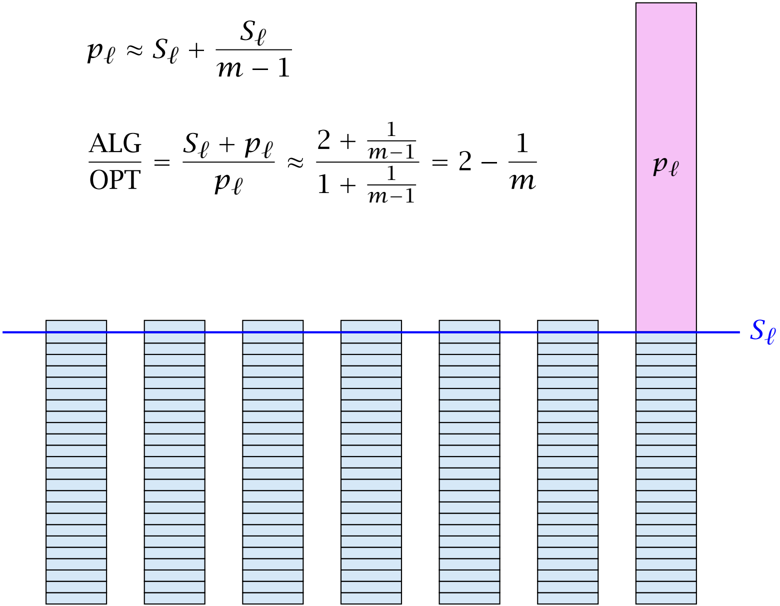
Hence, the length of the schedule is at most

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

# A Tight Example

$$p_\ell \approx S_\ell + \frac{S_\ell}{m-1}$$

$$\frac{\text{ALG}}{\text{OPT}} = \frac{S_\ell + p_\ell}{p_\ell} \approx \frac{2 + \frac{1}{m-1}}{1 + \frac{1}{m-1}} = 2 - \frac{1}{m}$$



# A Greedy Strategy

**List Scheduling:**
Order all processes in a list. When a machine runs empty assign the next yet unprocessed job to it.

Alternatively:
Consider processes in some order. Assign the $i$-th process to the least loaded machine.

It is easy to see that the result of these greedy strategies fulfill the local optimally condition of our local search algorithm. Hence, these also give 2-approximations.

# A Greedy Strategy

**List Scheduling:**
Order all processes in a list. When a machine runs empty assign the next yet unprocessed job to it.

Alternatively:
Consider processes in some order. Assign the $i$-th process to the least loaded machine.

It is easy to see that the result of these greedy strategies fulfill the local optimally condition of our local search algorithm. Hence, these also give 2-approximations.

# A Greedy Strategy

**List Scheduling:**
Order all processes in a list. When a machine runs empty assign the next yet unprocessed job to it.

Alternatively:
Consider processes in some order. Assign the $i$-th process to the least loaded machine.

It is easy to see that the result of these greedy strategies fulfill the local optimally condition of our local search algorithm. Hence, these also give 2-approximations.

# A Greedy Strategy

**List Scheduling:**
Order all processes in a list. When a machine runs empty assign the next yet unprocessed job to it.

Alternatively:
Consider processes in some order. Assign the $i$-th process to the least loaded machine.

It is easy to see that the result of these greedy strategies fulfill the local optimally condition of our local search algorithm. Hence, these also give 2-approximations.

# A Greedy Strategy

**Lemma 18**
*If we order the list according to non-increasing processing times the approximation guarantee of the list scheduling strategy improves to* $4/3$*.*

**Proof:**

- Let $p_1 \geq \cdots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- Wlog. the last job to finish is $n$ (otw. deleting this job gives another counter-example with fewer jobs).
- If $p_n \leq C^*_{\max}/3$ the previous analysis gives us a schedule length of at most

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

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

**Proof:**

- Let $p_1 \geq \cdots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- Wlog. the last job to finish is $n$ (otw. deleting this job gives another counter-example with fewer jobs).
- If $p_n \leq C^*_{\max}/3$ the previous analysis gives us a schedule length of at most

$$C^*_{\max} + p_n \leq \frac{4}{3} C^*_{\max} \enspace .$$

**Proof:**

- Let $p_1 \geq \cdots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- Wlog. the last job to finish is $n$ (otw. deleting this job gives another counter-example with fewer jobs).
- If $p_n \leq C^*_{\max}/3$ the previous analysis gives us a schedule length of at most

$$C^*_{\max} + p_n \leq \frac{4}{3} C^*_{\max} \enspace .$$

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

**Proof:**

- Let $p_1 \geq \cdots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- Wlog. the last job to finish is $n$ (otw. deleting this job gives another counter-example with fewer jobs).
- If $p_n \leq C^*_{\max}/3$ the previous analysis gives us a schedule length of at most

$$C^*_{\max} + p_n \leq \frac{4}{3}C^*_{\max} \enspace .$$

 Hence, $p_n > C^*_{\max}/3$.

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

**Proof:**

- Let $p_1 \geq \cdots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- Wlog. the last job to finish is $n$ (otw. deleting this job gives another counter-example with fewer jobs).
- If $p_n \leq C^*_{\max}/3$ the previous analysis gives us a schedule length of at most

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

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

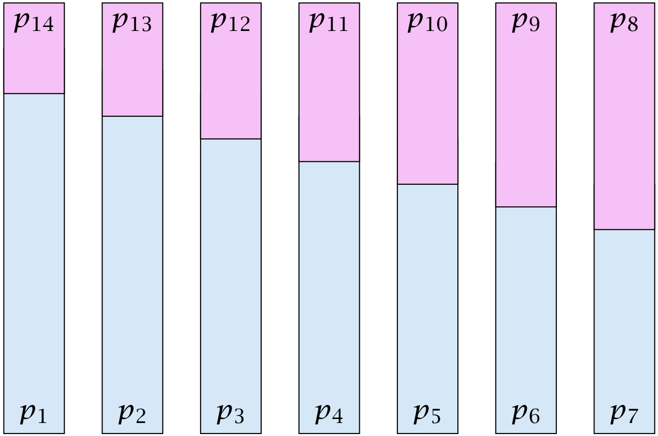
**Proof:**

- Let $p_1 \geq \cdots \geq p_n$ denote the processing times of a set of jobs that form a counter-example.
- Wlog. the last job to finish is $n$ (otw. deleting this job gives another counter-example with fewer jobs).
- If $p_n \leq C^*_{\max}/3$ the previous analysis gives us a schedule length of at most

$$C^*_{\max} + p_n \leq \frac{4}{3}C^*_{\max} \enspace .$$

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

When in an optimal solution a machine can have at most 2 jobs the optimal solution looks as follows.



- We can assume that one machine schedules $p_1$ and $p_n$ (the largest and smallest job).
- If not assume wlog. that $p_1$ is scheduled on machine $A$ and $p_n$ on machine $B$.
- Let $p_A$ and $p_B$ be the other job scheduled on $A$ and $B$, respectively.
- $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling $p_1$ and $p_n$ on one machine and $p_A$ and $p_B$ on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

- We can assume that one machine schedules $p_1$ and $p_n$ (the largest and smallest job).
- If not assume wlog. that $p_1$ is scheduled on machine $A$ and $p_n$ on machine $B$.
- Let $p_A$ and $p_B$ be the other job scheduled on $A$ and $B$, respectively.
- $p_1 + p_n \le p_1 + p_A$ and $p_A + p_B \le p_1 + p_A$, hence scheduling $p_1$ and $p_n$ on one machine and $p_A$ and $p_B$ on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

- We can assume that one machine schedules $p_1$ and $p_n$ (the largest and smallest job).
- If not assume wlog. that $p_1$ is scheduled on machine $A$ and $p_n$ on machine $B$.
- Let $p_A$ and $p_B$ be the other job scheduled on $A$ and $B$, respectively.
- $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling $p_1$ and $p_n$ on one machine and $p_A$ and $p_B$ on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

- We can assume that one machine schedules $p_1$ and $p_n$ (the largest and smallest job).
- If not assume wlog. that $p_1$ is scheduled on machine $A$ and $p_n$ on machine $B$.
- Let $p_A$ and $p_B$ be the other job scheduled on $A$ and $B$, respectively.
- $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling $p_1$ and $p_n$ on one machine and $p_A$ and $p_B$ on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

- We can assume that one machine schedules $p_1$ and $p_n$ (the largest and smallest job).
- If not assume wlog. that $p_1$ is scheduled on machine $A$ and $p_n$ on machine $B$.
- Let $p_A$ and $p_B$ be the other job scheduled on $A$ and $B$, respectively.
- $p_1 + p_n \leq p_1 + p_A$ and $p_A + p_B \leq p_1 + p_A$, hence scheduling $p_1$ and $p_n$ on one machine and $p_A$ and $p_B$ on the other, cannot increase the Makespan.
- Repeat the above argument for the remaining machines.

# 16 Rounding Data + Dynamic Programming

**Knapsack:**
Given a set of items $\{1,\dots,n\}$, 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 $w_i \leq W$).

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

# 16 Rounding Data + Dynamic Programming

**Knapsack:**
Given a set of items $\{1, \ldots, n\}$, where the $i$-th item has weight $w_i \in \mathbb{N}$ and profit $p_i \in \mathbb{N}$, and given a threshold $W$. Find a subset $I \subseteq \{1, \ldots, n\}$ of items of total weight at most $W$ such that the profit is maximized (we can assume each $w_i \leq W$).

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

# 16 Rounding Data + Dynamic Programming

**Algorithm 1** Knapsack

$A(1) \leftarrow [(0,0),(p_1,w_1)]$
**for** $j \leftarrow 2$ to $n$ **do**
    $A(j) \leftarrow A(j-1)$
    **for** each $(p,w) \in A(j-1)$ **do**
        **if** $w + w_j \le W$ **then**
            add $(p+p_j, w+w_j)$ to $A(j)$
        remove dominated pairs from $A(j)$
**return** $\max_{(p,w)\in A(n)} p$

The running time is $\mathcal{O}(n \cdot \min\{W,P\})$, where $P = \sum_i p_i$ is the total profit of all items. This is only pseudo-polynomial.

# 16 Rounding Data + Dynamic Programming

**Definition 19**
An algorithm is said to have pseudo-polynomial running time if the running time is polynomial when the numerical part of the input is encoded in unary.

# 16 Rounding Data + Dynamic Programming

- Let $M$ be the maximum profit of an element.

# 16 Rounding Data + Dynamic Programming

- Let $M$ be the maximum profit of an element.
- Set $\mu := \epsilon M/n$.

# 16 Rounding Data + Dynamic Programming

- Let $M$ be the maximum profit of an element.
- Set $\mu := \epsilon M/n$.
- Set $p'_i := \lfloor p_i/\mu \rfloor$ for all $i$.

# 16 Rounding Data + Dynamic Programming

- Let $M$ be the maximum profit of an element.
- Set $\mu := \epsilon M/n$.
- Set $p'_i := \lfloor p_i/\mu \rfloor$ for all $i$.
- Run the dynamic programming algorithm on this revised instance.

# 16 Rounding Data + Dynamic Programming

- Let $M$ be the maximum profit of an element.
- Set $\mu := \epsilon M/n$.
- Set $p_i' := \lfloor p_i/\mu \rfloor$ for all $i$.
- Run the dynamic programming algorithm on this revised instance.

Running time is at most

$$\mathcal{O}(nP')$$

# 16 Rounding Data + Dynamic Programming

- Let $M$ be the maximum profit of an element.
- Set $\mu := \epsilon M/n$.
- Set $p_i' := \lfloor p_i/\mu \rfloor$ for all $i$.
- Run the dynamic programming algorithm on this revised instance.

Running time is at most

$$\mathcal{O}(nP') = \mathcal{O}\Big(n \sum_i p_i'\Big)$$

# 16 Rounding Data + Dynamic Programming

- Let $M$ be the maximum profit of an element.
- Set $\mu := \epsilon M/n$.
- Set $p'_i := \lfloor p_i/\mu \rfloor$ for all $i$.
- Run the dynamic programming algorithm on this revised instance.

Running time is at most

$$\mathcal{O}(nP') = \mathcal{O}\Big(n\sum_i p'_i\Big) = \mathcal{O}\Big(n\sum_i \Big\lfloor \frac{p_i}{\epsilon M/n} \Big\rfloor\Big)$$

# 16 Rounding Data + Dynamic Programming

- Let $M$ be the maximum profit of an element.
- Set $\mu := \epsilon M/n$.
- Set $p_i' := \lfloor p_i/\mu \rfloor$ for all $i$.
- Run the dynamic programming algorithm on this revised instance.

Running time is at most

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

# 16 Rounding Data + Dynamic Programming

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

$$\sum_{i\in S} p_i$$

# 16 Rounding Data + Dynamic Programming

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

$$\sum_{i\in S} p_i \geq \mu \sum_{i\in S} p'_i$$

# 16 Rounding Data + Dynamic Programming

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

$$\begin{aligned}\sum_{i\in S} p_i &\geq \mu \sum_{i\in S} p'_i \\ &\geq \mu \sum_{i\in O} p'_i\end{aligned}$$

# 16 Rounding Data + Dynamic Programming

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

$$\begin{aligned}\sum_{i\in S} p_i &\geq \mu \sum_{i\in S} p_i' \\ &\geq \mu \sum_{i\in O} p_i' \\ &\geq \sum_{i\in O} p_i - |O|\mu\end{aligned}$$

# 16 Rounding Data + Dynamic Programming

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

$$\begin{aligned}\sum_{i\in S} p_i &\geq \mu \sum_{i\in S} p_i' \\ &\geq \mu \sum_{i\in O} p_i' \\ &\geq \sum_{i\in O} p_i - |O|\mu \\ &\geq \sum_{i\in O} p_i - n\mu\end{aligned}$$

# 16 Rounding Data + Dynamic Programming

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

$$
\begin{aligned}
\sum_{i\in S} p_i &\geq \mu \sum_{i\in S} p_i' \\
&\geq \mu \sum_{i\in O} p_i' \\
&\geq \sum_{i\in O} p_i - |O|\mu \\
&\geq \sum_{i\in O} p_i - n\mu \\
&= \sum_{i\in O} p_i - \epsilon M
\end{aligned}
$$

# 16 Rounding Data + Dynamic Programming

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

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

# 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.

# 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.

# 16.2 Scheduling Revisited

Partition the input into long jobs and short jobs.

# 16.2 Scheduling Revisited

Partition the input into long jobs and short jobs.

A job $j$ is called short if

$$p_j \leq \frac{1}{km} \sum_i p_i$$

# 16.2 Scheduling Revisited

Partition the input into long jobs and short jobs.

A job $j$ is called short if

$$p_j \leq \frac{1}{km} \sum_i p_i$$

**Idea:**

1. Find the optimum Makespan for the long jobs by brute force.

# 16.2 Scheduling Revisited

Partition the input into long jobs and short jobs.

A job $j$ is called short if

$$p_j \leq \frac{1}{km} \sum_i p_i$$

**Idea:**

1. Find the optimum Makespan for the long jobs by brute force.
2. Then use the list scheduling algorithm for the short jobs, always assigning the next job to the least loaded machine.

We still have the inequality

$$\frac{1}{m}\sum_{j\neq\ell} p_j + p_\ell$$

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

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.

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

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

There are at most $km$ long jobs. Hence, the number of possibilities of scheduling these jobs on $m$ machines is at most $m^{km}$, which is constant if $m$ is constant. Hence, it is easy to implement the algorithm in polynomial time.

**Theorem 20**

*The above algorithm gives a polynomial time approximation scheme (PTAS) for the problem of scheduling $n$ jobs on $m$ identical machines if $m$ is constant.*

We choose $k = \lceil \frac{1}{\epsilon} \rceil$.

Hence we get a schedule of length at most

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

There are at most $km$ long jobs. Hence, the number of possibilities of scheduling these jobs on $m$ machines is at most $m^{km}$, which is constant if $m$ is constant. Hence, it is easy to implement the algorithm in polynomial time.

**Theorem 20**

*The above algorithm gives a polynomial time approximation scheme (PTAS) for the problem of scheduling $n$ jobs on $m$ identical machines if $m$ is constant.*

We choose $k = \lceil \frac{1}{\epsilon} \rceil$.

Hence we get a schedule of length at most

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

There are at most $km$ long jobs. Hence, the number of possibilities of scheduling these jobs on $m$ machines is at most $m^{km}$, which is constant if $m$ is constant. Hence, it is easy to implement the algorithm in polynomial time.

**Theorem 20**

*The above algorithm gives a polynomial time approximation scheme (PTAS) for the problem of scheduling $n$ jobs on $m$ identical machines if $m$ is constant.*

We choose $k = \lceil \frac{1}{\epsilon} \rceil$.

How to get rid of the requirement that $m$ is constant?

We first design an algorithm that works as follows:
On input of $T$ it either finds a schedule of length $(1+\frac{1}{k})T$ or certifies that no schedule of length at most $T$ exists (assume $T \geq \frac{1}{m}\sum_j p_j$).

We partition the jobs into long jobs and short jobs:

- A job is long if its size is larger than $T/k$.
- Otw. it is a short job.

How to get rid of the requirement that $m$ is constant?

We first design an algorithm that works as follows:
On input of $T$ it either finds a schedule of length $(1 + \frac{1}{k})T$ or certifies that no schedule of length at most $T$ exists (assume $T \geq \frac{1}{m}\sum_j p_j$).

We partition the jobs into long jobs and short jobs:

- A job is long if its size is larger than $T/k$.
- Otw. it is a short job.

How to get rid of the requirement that $m$ is constant?

We first design an algorithm that works as follows:
On input of $T$ it either finds a schedule of length $(1+\frac{1}{k})T$ or certifies that no schedule of length at most $T$ exists (assume $T \geq \frac{1}{m}\sum_j p_j$).

We partition the jobs into long jobs and short jobs:

- A job is long if its size is larger than $T/k$.
- Otw. it is a short job.

How to get rid of the requirement that $m$ is constant?

We first design an algorithm that works as follows:
On input of $T$ it either finds a schedule of length $(1 + \frac{1}{k})T$ or certifies that no schedule of length at most $T$ exists (assume $T \geq \frac{1}{m}\sum_j p_j$).

We partition the jobs into long jobs and short jobs:

- A job is long if its size is larger than $T/k$.
- Otw. it is a short job.

- We round all long jobs down to multiples of $T/k^2$.
- For these rounded sizes we first find an optimal schedule.
- If this schedule does not have length at most $T$ we conclude that also the original sizes don't allow such a schedule.
- If we have a good schedule we extend it by adding the short jobs according to the LPT rule.

- We round all long jobs down to multiples of $T/k^2$.
- For these rounded sizes we first find an optimal schedule.
- If this schedule does not have length at most $T$ we conclude that also the original sizes don't allow such a schedule.
- If we have a good schedule we extend it by adding the short jobs according to the LPT rule.

- We round all long jobs down to multiples of $T/k^2$.
- For these rounded sizes we first find an optimal schedule.
- If this schedule does not have length at most $T$ we conclude that also the original sizes don't allow such a schedule.
- If we have a good schedule we extend it by adding the short jobs according to the LPT rule.

- We round all long jobs down to multiples of $T/k^2$.
- For these rounded sizes we first find an optimal schedule.
- If this schedule does not have length at most $T$ we conclude that also the original sizes don't allow such a schedule.
- If we have a good schedule we extend it by adding the short jobs according to the LPT rule.

After the first phase the rounded sizes of the long jobs assigned to a machine add up to at most $T$.

There can be at most $k$ (long) jobs assigned to a machine as otw. their rounded sizes would add up to more than $T$ (note that the rounded size of a long job is at least $T/k$).

Since, jobs had been rounded to multiples of $T/k^2$ going from rounded sizes to original sizes gives that the Makespan is at most

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

After the first phase the rounded sizes of the long jobs assigned to a machine add up to at most $T$.

There can be at most $k$ (long) jobs assigned to a machine as otw. their rounded sizes would add up to more than $T$ (note that the rounded size of a long job is at least $T/k$).

Since, jobs had been rounded to multiples of $T/k^2$ going from rounded sizes to original sizes gives that the Makespan is at most

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

After the first phase the rounded sizes of the long jobs assigned to a machine add up to at most $T$.

There can be at most $k$ (long) jobs assigned to a machine as otw. their rounded sizes would add up to more than $T$ (note that the rounded size of a long job is at least $T/k$).

Since, jobs had been rounded to multiples of $T/k^2$ going from rounded sizes to original sizes gives that the Makespan is at most

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

During the second phase there always must exist a machine with load at most $T$, since $T$ is larger than the average load.

Assigning the current (short) job to such a machine gives that the new load is at most

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

During the second phase there always must exist a machine with load at most $T$, since $T$ is larger than the average load. Assigning the current (short) job to such a machine gives that the new load is at most

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

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

Hence, any large job has rounded size of $\frac{i}{k^2}T$ for $i \in \{k, \dots, k^2\}$. Therefore the number of different inputs is at most $n^{k^2}$ (described by a vector of length $k^2$ where, the $i$-th entry describes the number of jobs of size $\frac{i}{k^2}T$). This is polynomial.

The schedule/configuration of a particular machine $x$ can be described by a vector of length $k^2$ where the $i$-th entry describes the number of jobs of rounded size $\frac{i}{k^2}T$ assigned to $x$. There are only $(k+1)^{k^2}$ different vectors.

This means there are a constant number of different machine configurations.

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

Hence, any large job has rounded size of $\frac{i}{k^2}T$ for $i \in \{k, \dots, k^2\}$. Therefore the number of different inputs is at most $n^{k^2}$ (described by a vector of length $k^2$ where, the $i$-th entry describes the number of jobs of size $\frac{i}{k^2}T$). This is polynomial.

The schedule/configuration of a particular machine $x$ can be described by a vector of length $k^2$ where the $i$-th entry describes the number of jobs of rounded size $\frac{i}{k^2}T$ assigned to $x$. There are only $(k+1)^{k^2}$ different vectors.

This means there are a constant number of different machine configurations.

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

Hence, any large job has rounded size of $\frac{i}{k^2}T$ for $i \in \{k, \ldots, k^2\}$. Therefore the number of different inputs is at most $n^{k^2}$ (described by a vector of length $k^2$ where, the $i$-th entry describes the number of jobs of size $\frac{i}{k^2}T$). This is polynomial.

The schedule/configuration of a particular machine $x$ can be described by a vector of length $k^2$ where the $i$-th entry describes the number of jobs of rounded size $\frac{i}{k^2}T$ assigned to $x$. There are only $(k+1)^{k^2}$ different vectors.

This means there are a constant number of different machine configurations.

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

Hence, any large job has rounded size of $\frac{i}{k^2}T$ for $i \in \{k, \dots, k^2\}$. Therefore the number of different inputs is at most $n^{k^2}$ (described by a vector of length $k^2$ where, the $i$-th entry describes the number of jobs of size $\frac{i}{k^2}T$). This is polynomial.

The schedule/configuration of a particular machine $x$ can be described by a vector of length $k^2$ where the $i$-th entry describes the number of jobs of rounded size $\frac{i}{k^2}T$ assigned to $x$. There are only $(k+1)^{k^2}$ different vectors.

This means there are a constant number of different machine configurations.

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

We have

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

where $C$ is the set of all configurations.

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

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

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

We have

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

where $C$ is the set of all configurations.

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

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

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

We have

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

where $C$ is the set of all configurations.

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

Let $\mathrm{OPT}(n_1, \dots, n_{k^2})$ 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

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

where $C$ is the set of all configurations.

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

We can turn this into a PTAS by choosing $k = \lceil 1/\epsilon \rceil$ and using binary search. This gives a running time that is exponential in $1/\epsilon$.

Can we do better?

Scheduling on identical machines with the goal of minimizing Makespan is a strongly NP-complete problem.

**Theorem 21**

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

We can turn this into a PTAS by choosing $k = \lceil 1/\epsilon \rceil$ and using binary search. This gives a running time that is exponential in $1/\epsilon$.

Can we do better?

Scheduling on identical machines with the goal of minimizing Makespan is a strongly NP-complete problem.

**Theorem 21**

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

We can turn this into a PTAS by choosing $k = \lceil 1/\epsilon \rceil$ and using binary search. This gives a running time that is exponential in $1/\epsilon$.

**Can we do better?**

Scheduling on identical machines with the goal of minimizing Makespan is a strongly NP-complete problem.

**Theorem 21**

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

We can turn this into a PTAS by choosing $k = \lceil 1/\epsilon \rceil$ and using binary search. This gives a running time that is exponential in $1/\epsilon$.

Can we do better?

Scheduling on identical machines with the goal of minimizing Makespan is a strongly NP-complete problem.

**Theorem 21**

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

- Suppose we have an instance with polynomially bounded processing times $p_i \leq q(n)$
- We set $k := \lceil 2nq(n) \rceil \geq 2\,\mathrm{OPT}$
- Then

$$\mathrm{ALG} \leq \left(1 + \frac{1}{k}\right)\mathrm{OPT} \leq \mathrm{OPT} + \frac{1}{2}$$

- But this means that the algorithm computes the optimal solution as the optimum is integral.
- This means we can solve problem instances if processing times are polynomially bounded
- Running time is $\mathcal{O}(\mathrm{poly}(n,k)) = \mathcal{O}(\mathrm{poly}(n))$
- For strongly NP-complete problems this is not possible unless P=NP

- Suppose we have an instance with polynomially bounded processing times $p_i \leq q(n)$
- We set $k := \lceil 2nq(n) \rceil \geq 2\,\mathrm{OPT}$
- Then

$$\mathrm{ALG} \leq \left(1 + \frac{1}{k}\right)\mathrm{OPT} \leq \mathrm{OPT} + \frac{1}{2}$$

- But this means that the algorithm computes the optimal solution as the optimum is integral.
- This means we can solve problem instances if processing times are polynomially bounded
- Running time is $\mathcal{O}(\mathrm{poly}(n,k)) = \mathcal{O}(\mathrm{poly}(n))$
- For strongly NP-complete problems this is not possible unless P=NP

- Suppose we have an instance with polynomially bounded processing times $p_i \leq q(n)$
- We set $k := \lceil 2nq(n) \rceil \geq 2\,\mathrm{OPT}$
- Then

$$\mathrm{ALG} \leq \left(1 + \frac{1}{k}\right)\mathrm{OPT} \leq \mathrm{OPT} + \frac{1}{2}$$

- But this means that the algorithm computes the optimal solution as the optimum is integral.
- This means we can solve problem instances if processing times are polynomially bounded
- Running time is $\mathcal{O}(\mathrm{poly}(n,k)) = \mathcal{O}(\mathrm{poly}(n))$
- For strongly NP-complete problems this is not possible unless P=NP

- Suppose we have an instance with polynomially bounded processing times $p_i \leq q(n)$
- We set $k := \lceil 2nq(n) \rceil \geq 2\,\mathrm{OPT}$
- Then
$$\mathrm{ALG} \leq \left(1 + \frac{1}{k}\right)\mathrm{OPT} \leq \mathrm{OPT} + \frac{1}{2}$$
- But this means that the algorithm computes the optimal solution as the optimum is integral.
- This means we can solve problem instances if processing times are polynomially bounded
- Running time is $\mathcal{O}(\mathrm{poly}(n,k)) = \mathcal{O}(\mathrm{poly}(n))$
- For strongly NP-complete problems this is not possible unless P=NP

- Suppose we have an instance with polynomially bounded processing times $p_i \leq q(n)$
- We set $k := \lceil 2nq(n) \rceil \geq 2\,\mathrm{OPT}$
- Then
$$\mathrm{ALG} \leq \left(1 + \frac{1}{k}\right)\mathrm{OPT} \leq \mathrm{OPT} + \frac{1}{2}$$
- But this means that the algorithm computes the optimal solution as the optimum is integral.
- This means we can solve problem instances if processing times are polynomially bounded
- Running time is $\mathcal{O}(\mathrm{poly}(n,k)) = \mathcal{O}(\mathrm{poly}(n))$
- For strongly NP-complete problems this is not possible unless P=NP

- Suppose we have an instance with polynomially bounded processing times $p_i \leq q(n)$
- We set $k := \lceil 2nq(n) \rceil \geq 2\,\mathrm{OPT}$
- Then
$$\mathrm{ALG} \leq \left(1 + \frac{1}{k}\right)\mathrm{OPT} \leq \mathrm{OPT} + \frac{1}{2}$$
- But this means that the algorithm computes the optimal solution as the optimum is integral.
- This means we can solve problem instances if processing times are polynomially bounded
- Running time is $\mathcal{O}(\mathrm{poly}(n,k)) = \mathcal{O}(\mathrm{poly}(n))$
- For strongly NP-complete problems this is not possible unless P=NP

- Suppose we have an instance with polynomially bounded processing times $p_i \le q(n)$
- We set $k := \lceil 2nq(n) \rceil \ge 2\,\mathrm{OPT}$
- Then
$$\mathrm{ALG} \le \left(1 + \frac{1}{k}\right)\mathrm{OPT} \le \mathrm{OPT} + \frac{1}{2}$$
- But this means that the algorithm computes the optimal solution as the optimum is integral.
- This means we can solve problem instances if processing times are polynomially bounded
- Running time is $\mathcal{O}(\mathrm{poly}(n,k)) = \mathcal{O}(\mathrm{poly}(n))$
- For strongly NP-complete problems this is not possible unless P=NP

# More General

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

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

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

where $C$ is the set of all configurations.

$|C| \le (B+1)^A$, where $B$ is the number of jobs that possibly can fit on the same machine.

The running time is then $O((B+1)^A n^A)$ because the dynamic programming table has just $n^A$ entries.

# More General

Let $\mathrm{OPT}(n_1, \ldots, n_A)$ 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.

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

where $C$ is the set of all configurations.

$|C| \leq (B+1)^A$, where $B$ is the number of jobs that possibly can fit on the same machine.

The running time is then $O((B+1)^A n^A)$ because the dynamic programming table has just $n^A$ entries.

## More General

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

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

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

where $C$ is the set of all configurations.

$|C| \leq (B+1)^A$, where $B$ is the number of jobs that possibly can fit on the same machine.

The running time is then $O((B+1)^A n^A)$ because the dynamic programming table has just $n^A$ entries.

# Bin Packing

Given $n$ items with sizes $s_1, \ldots, s_n$ where

$$1 > s_1 \geq \cdots \geq s_n > 0 \ .$$

Pack items into a minimum number of bins where each bin can hold items of total size at most 1.

**Theorem 22**

*There is no $\rho$-approximation for Bin Packing with $\rho < 3/2$ unless $\mathrm{P} = \mathrm{NP}$.*

# Bin Packing

Given $n$ items with sizes $s_1, \ldots, s_n$ where

$$1 > s_1 \geq \cdots \geq s_n > 0 \ .$$

Pack items into a minimum number of bins where each bin can hold items of total size at most 1.

**Theorem 22**
*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 $b_1, \dots, b_n$ with $B = \sum_i b_i$ even. Can we partition the integers into two sets $S$ and $T$ s.t.

$$\sum_{i \in S} b_i = \sum_{i \in T} b_i \quad ?$$

- We can solve this problem by setting $s_i := 2b_i/B$ and asking whether we can pack the resulting items into 2 bins or not.
- A $\rho$-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

**Proof**

- In the partition problem we are given positive integers $b_1, \dots, b_n$ with $B = \sum_i b_i$ even. Can we partition the integers into two sets $S$ and $T$ s.t.

$$\sum_{i \in S} b_i = \sum_{i \in T} b_i \quad ?$$

- We can solve this problem by setting $s_i := 2b_i/B$ and asking whether we can pack the resulting items into 2 bins or not.
- A $\rho$-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

**Proof**

- In the partition problem we are given positive integers $b_1, \dots, b_n$ with $B = \sum_i b_i$ even. Can we partition the integers into two sets $S$ and $T$ s.t.

$$\sum_{i \in S} b_i = \sum_{i \in T} b_i \quad ?$$

- We can solve this problem by setting $s_i := 2b_i/B$ and asking whether we can pack the resulting items into $2$ bins or not.
- A $\rho$-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

**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 23

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

**Definition 23**

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

**Definition 23**
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 24**

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

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

# Bin Packing

Again we can differentiate between small and large items.

**Lemma 24**

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

- If after Greedy we use more than $\ell$ 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.

# Bin Packing

Again we can differentiate between small and large items.

### Lemma 24

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

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

# Bin Packing

Again we can differentiate between small and large items.

**Lemma 24**

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

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

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

$$\frac{1}{1-\epsilon/2} \cdot \mathrm{OPT} + 1 \leq (1+\epsilon) \cdot \mathrm{OPT} + 1$$

bins.

It remains to find an algorithm for the large items.

# Bin Packing

**Linear Grouping:**

Generate an instance $I'$ (for large items) as follows.

- Order large items according to size.
- Let the first $k$ items belong to group 1; the following $k$ items belong to group 2; etc.
- Delete items in the first group;
- Round items in the remaining groups to the size of the largest item in the group.

# Bin Packing

**Linear Grouping:**

Generate an instance $I'$ (for large items) as follows.

- Order large items according to size.
- Let the first $k$ items belong to group 1; the following $k$ items belong to group $2$; etc.
- Delete items in the first group;
- Round items in the remaining groups to the size of the largest item in the group.

# Bin Packing

**Linear Grouping:**

Generate an instance $I'$ (for large items) as follows.

- Order large items according to size.
- Let the first $k$ items belong to group 1; the following $k$ items belong to group $2$; etc.
- Delete items in the first group;
- Round items in the remaining groups to the size of the largest item in the group.

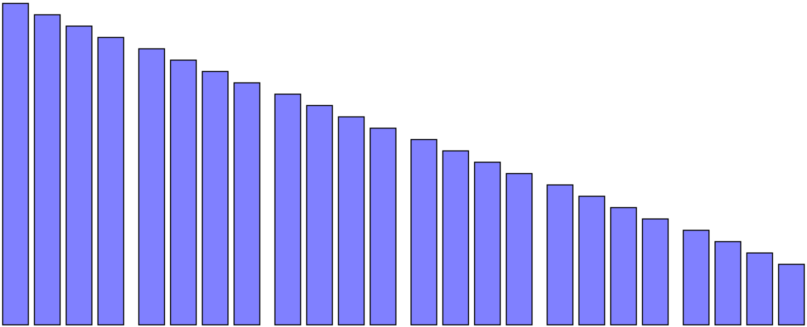
# Bin Packing

**Linear Grouping:**

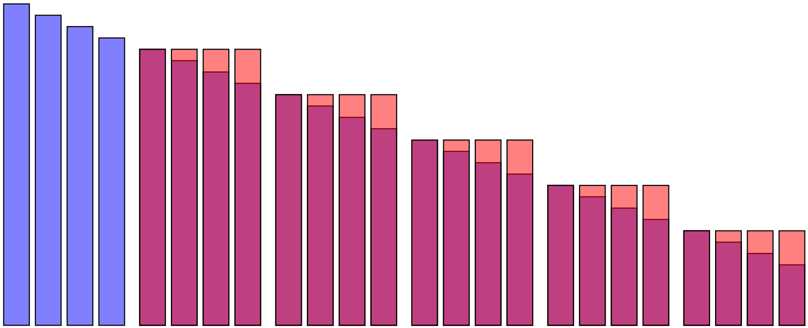
Generate an instance $I'$ (for large items) as follows.

- Order large items according to size.
- Let the first $k$ items belong to group 1; the following $k$ items belong to group $2$; etc.
- Delete items in the first group;
- Round items in the remaining groups to the size of the largest item in the group.

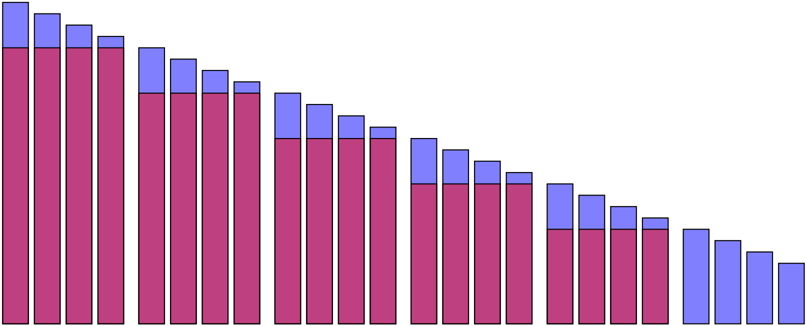
# Linear Grouping



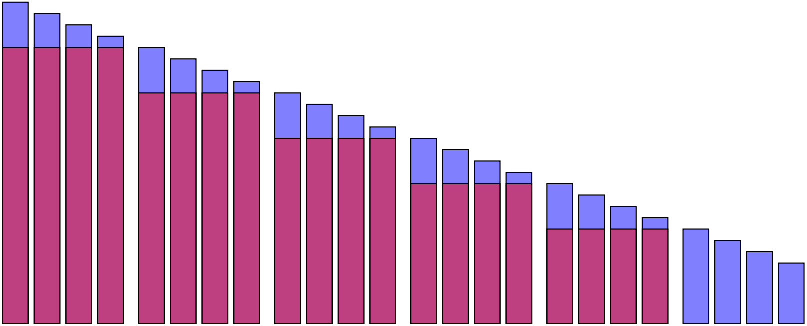
# Linear Grouping



# Linear Grouping



# Linear Grouping



**Lemma 25**

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

**Proof 1:**

- Any bin packing for $I$ gives a bin packing for $I'$ as follows.
- Pack the items of group 2, where in the packing for $I$ the items for group 1 have been packed;
- Pack the items of groups 3, where in the packing for $I$ the items for group 2 have been packed;
- ...

## Lemma 25

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

**Proof 1:**

- Any bin packing for $I$ gives a bin packing for $I'$ as follows.
- Pack the items of group 2, where in the packing for $I$ the items for group 1 have been packed;
- Pack the items of groups 3, where in the packing for $I$ the items for group 2 have been packed;
- ...

## Lemma 25

$\mathrm{OPT}(I') \leq \mathrm{OPT}(I) \leq \mathrm{OPT}(I') + k$

**Proof 1:**

- Any bin packing for $I$ gives a bin packing for $I'$ as follows.
- Pack the items of group $2$, where in the packing for $I$ the items for group $1$ have been packed;
- Pack the items of groups $3$, where in the packing for $I$ the items for group $2$ have been packed;
- ...

## Lemma 25

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

**Proof 1:**

- Any bin packing for $I$ gives a bin packing for $I'$ as follows.
- Pack the items of group $2$, where in the packing for $I$ the items for group $1$ have been packed;
- Pack the items of groups $3$, where in the packing for $I$ the items for group $2$ have been packed;
- ...

## Lemma 25

$\mathrm{OPT}(I') \leq \mathrm{OPT}(I) \leq \mathrm{OPT}(I') + k$

**Proof 1:**

- Any bin packing for $I$ gives a bin packing for $I'$ as follows.
- Pack the items of group $2$, where in the packing for $I$ the items for group $1$ have been packed;
- Pack the items of groups $3$, where in the packing for $I$ the items for group $2$ have been packed;
- ...

## Lemma 26

$\mathrm{OPT}(I') \leq \mathrm{OPT}(I) \leq \mathrm{OPT}(I') + k$

**Proof 2:**

- Any bin packing for $I'$ gives a bin packing for $I$ as follows.
- Pack the items of group 1 into $k$ new bins;
- Pack the items of groups 2, where in the packing for $I'$ the items for group 2 have been packed;
- ...

## Lemma 26

$\mathrm{OPT}(I') \leq \mathrm{OPT}(I) \leq \mathrm{OPT}(I') + k$

**Proof 2:**

- Any bin packing for $I'$ gives a bin packing for $I$ as follows.
- Pack the items of group $1$ into $k$ new bins;
- Pack the items of groups $2$, where in the packing for $I'$ the items for group $2$ have been packed;
- ...

## Lemma 26

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

**Proof 2:**

- Any bin packing for $I'$ gives a bin packing for $I$ as follows.
- Pack the items of group $1$ into $k$ new bins;
- Pack the items of groups $2$, where in the packing for $I'$ the items for group $2$ have been packed;
- ...

## Lemma 26

$\mathrm{OPT}(I') \leq \mathrm{OPT}(I) \leq \mathrm{OPT}(I') + k$

**Proof 2:**

- Any bin packing for $I'$ gives a bin packing for $I$ as follows.
- Pack the items of group $1$ into $k$ new bins;
- Pack the items of groups $2$, where in the packing for $I'$ the items for group $2$ have been packed;
- ...

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

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

Then $n/k \leq n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (here we used $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes ($4/\epsilon^2$) and at most a constant number ($2/\epsilon$) can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

- cost (for large items) at most

$$\text{OPT}(I') + k \leq \text{OPT}(I) + \epsilon \text{SIZE}(I) \leq (1+\epsilon)\text{OPT}(I)$$

- running time $\mathcal{O}((\frac{2}{\epsilon} n)^{4/\epsilon^2})$.

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

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

Then $n/k \leq n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (here we used $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes ($4/\epsilon^2$) and at most a constant number ($2/\epsilon$) can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

- cost (for large items) at most

$$\mathrm{OPT}(I') + k \leq \mathrm{OPT}(I) + \epsilon \mathrm{SIZE}(I) \leq (1+\epsilon)\mathrm{OPT}(I)$$

- running time $\mathcal{O}((\frac{2}{\epsilon}n)^{4/\epsilon^2})$.

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

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

Then $n/k \leq n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (here we used $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes ($4/\epsilon^2$) and at most a constant number ($2/\epsilon$) can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

- cost (for large items) at most

$$\text{OPT}(I') + k \leq \text{OPT}(I) + \epsilon\text{SIZE}(I) \leq (1+\epsilon)\text{OPT}(I)$$

- running time $\mathcal{O}((\frac{2}{\epsilon}n)^{4/\epsilon^2})$.

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

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

Then $n/k \leq n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (here we used $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes ($4/\epsilon^2$) and at most a constant number ($2/\epsilon$) can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

- cost (for large items) at most

$$\text{OPT}(I') + k \leq \text{OPT}(I) + \epsilon \text{SIZE}(I) \leq (1+\epsilon)\text{OPT}(I)$$

- running time $\mathcal{O}((\frac{2}{\epsilon} n)^{4/\epsilon^2})$.

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

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

Then $n/k \leq n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (here we used $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes ($4/\epsilon^2$) and at most a constant number ($2/\epsilon$) can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

- cost (for large items) at most

$$\mathrm{OPT}(I') + k \leq \mathrm{OPT}(I) + \epsilon \mathrm{SIZE}(I) \leq (1+\epsilon)\mathrm{OPT}(I)$$

- running time $\mathcal{O}((\frac{2}{\epsilon}n)^{4/\epsilon^2})$.

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

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

Then $n/k \leq n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (here we used $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes $(4/\epsilon^2)$ and at most a constant number $(2/\epsilon)$ can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

- cost (for large items) at most

$$\mathrm{OPT}(I') + k \leq \mathrm{OPT}(I) + \epsilon \mathrm{SIZE}(I) \leq (1+\epsilon)\mathrm{OPT}(I)$$

- running time $\mathcal{O}((\frac{2}{\epsilon} n)^{4/\epsilon^2})$.

## Can we do better?

In the following we show how to obtain a solution where the number of bins is only

$$\mathrm{OPT}(I) + \mathcal{O}(\log^2(\mathrm{SIZE}(I))) \ .$$

Note that this is usually better than a guarantee of

$$(1+\epsilon)\mathrm{OPT}(I) + 1 \ .$$

## Can we do better?

In the following we show how to obtain a solution where the number of bins is only

$$\mathrm{OPT}(I) + \mathcal{O}(\log^2(\mathrm{SIZE}(I))) \ .$$

Note that this is usually better than a guarantee of

$$(1+\epsilon)\mathrm{OPT}(I) + 1 \ .$$

## Can we do better?

In the following we show how to obtain a solution where the number of bins is only

$$\mathrm{OPT}(I) + \mathcal{O}(\log^2(\mathrm{SIZE}(I))) \ .$$

Note that this is usually better than a guarantee of

$$(1+\epsilon)\mathrm{OPT}(I) + 1 \ .$$

# Configuration LP

**Change of Notation:**

- Group pieces of identical size.
- Let $s_1$ denote the largest size, and let $b_1$ denote the number of pieces of size $s_1$.
- $s_2$ is second largest size and $b_2$ number of pieces of size $s_2$;
- ...
- $s_m$ smallest size and $b_m$ number of pieces of size $s_m$.

# Configuration LP

**Change of Notation:**

- Group pieces of identical size.
- Let $s_1$ denote the largest size, and let $b_1$ denote the number of pieces of size $s_1$.
- $s_2$ is second largest size and $b_2$ number of pieces of size $s_2$;
- ...
- $s_m$ smallest size and $b_m$ number of pieces of size $s_m$.

# Configuration LP

**Change of Notation:**

- Group pieces of identical size.
- Let $s_1$ denote the largest size, and let $b_1$ denote the number of pieces of size $s_1$.
- $s_2$ is second largest size and $b_2$ number of pieces of size $s_2$;
- ...
- $s_m$ smallest size and $b_m$ number of pieces of size $s_m$.

# Configuration LP

**Change of Notation:**

- Group pieces of identical size.
- Let $s_1$ denote the largest size, and let $b_1$ denote the number of pieces of size $s_1$.
- $s_2$ is second largest size and $b_2$ number of pieces of size $s_2$;
- ...
- $s_m$ smallest size and $b_m$ number of pieces of size $s_m$.

# Configuration LP

**Change of Notation:**

- Group pieces of identical size.
- Let $s_1$ denote the largest size, and let $b_1$ denote the number of pieces of size $s_1$.
- $s_2$ is second largest size and $b_2$ number of pieces of size $s_2$;
- ...
- $s_m$ smallest size and $b_m$ number of pieces of size $s_m$.

# Configuration LP

A possible packing of a bin can be described by an $m$-tuple $(t_1, \ldots, t_m)$, where $t_i$ describes the number of pieces of size $s_i$. Clearly,

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

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

# Configuration LP

A possible packing of a bin can be described by an $m$-tuple $(t_1, \ldots, t_m)$, where $t_i$ describes the number of pieces of size $s_i$. Clearly,

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

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

# Configuration LP

A possible packing of a bin can be described by an $m$-tuple $(t_1, \ldots, t_m)$, where $t_i$ describes the number of pieces of size $s_i$. Clearly,

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

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

# Configuration LP

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

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

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

# Configuration LP

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

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

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

# Configuration LP

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

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

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

# Configuration LP

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

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

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

**How to solve this LP?**

later...

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

# Harmonic Grouping

- Sort items according to size (monotonically decreasing).
- Process items in this order; close the current group if size of items in the group is at least 2 (or larger). Then open new group.
- I.e., $G_1$ is the smallest cardinality set of largest items s.t. total size sums up to at least 2. Similarly, for $G_2, \dots, G_{r-1}$.
- Only the size of items in the last group $G_r$ may sum up to less than 2.

# Harmonic Grouping

- Sort items according to size (monotonically decreasing).
- Process items in this order; close the current group if size of items in the group is at least $2$ (or larger). Then open new group.
- I.e., $G_1$ is the smallest cardinality set of largest items s.t. total size sums up to at least $2$. Similarly, for $G_2, \dots, G_{r-1}$.
- Only the size of items in the last group $G_r$ may sum up to less than $2$.

# Harmonic Grouping

- Sort items according to size (monotonically decreasing).
- Process items in this order; close the current group if size of items in the group is at least $2$ (or larger). Then open new group.
- I.e., $G_1$ is the smallest cardinality set of largest items s.t. total size sums up to at least $2$. Similarly, for $G_2, \dots, G_{r-1}$.
- Only the size of items in the last group $G_r$ may sum up to less than $2$.

# Harmonic Grouping

- Sort items according to size (monotonically decreasing).
- Process items in this order; close the current group if size of items in the group is at least $2$ (or larger). Then open new group.
- I.e., $G_1$ is the smallest cardinality set of largest items s.t. total size sums up to at least $2$. Similarly, for $G_2, \dots, G_{r-1}$.
- Only the size of items in the last group $G_r$ may sum up to less than $2$.

# Harmonic Grouping

From the grouping we obtain instance $I'$ as follows:

- Round all items in a group to the size of the largest group member.
- Delete all items from group $G_1$ and $G_r$.
- For groups $G_2, \ldots, G_{r-1}$ delete $n_i - n_{i-1}$ items.
- Observe that $n_i \geq n_{i-1}$.

# Harmonic Grouping

From the grouping we obtain instance $I'$ as follows:

- Round all items in a group to the size of the largest group member.
- Delete all items from group $G_1$ and $G_r$.
- For groups $G_2, \ldots, G_{r-1}$ delete $n_i - n_{i-1}$ items.
- Observe that $n_i \geq n_{i-1}$.

# Harmonic Grouping

From the grouping we obtain instance $I'$ as follows:

- Round all items in a group to the size of the largest group member.
- Delete all items from group $G_1$ and $G_r$.
- For groups $G_2, \ldots, G_{r-1}$ delete $n_i - n_{i-1}$ items.
- Observe that $n_i \geq n_{i-1}$.

# Harmonic Grouping

From the grouping we obtain instance $I'$ as follows:

- Round all items in a group to the size of the largest group member.
- Delete all items from group $G_1$ and $G_r$.
- For groups $G_2, \ldots, G_{r-1}$ delete $n_i - n_{i-1}$ items.
- Observe that $n_i \geq n_{i-1}$.

## Lemma 27

*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 27**

*The number of different sizes in $I'$ is at most $\text{SIZE}(I)/2$.*

- Each group that survives (recall that $G_1$ and $G_r$ are deleted) has total size at least $2$.
- Hence, the number of surviving groups is at most $\text{SIZE}(I)/2$.
- All items in a group have the same size in $I'$.

**Lemma 27**

*The number of different sizes in $I'$ is at most $\text{SIZE}(I)/2$.*

- Each group that survives (recall that $G_1$ and $G_r$ are deleted) has total size at least $2$.
- Hence, the number of surviving groups is at most $\text{SIZE}(I)/2$.
- All items in a group have the same size in $I'$.

## Lemma 27

*The number of different sizes in $I'$ is at most* $\text{SIZE}(I)/2$.

- Each group that survives (recall that $G_1$ and $G_r$ are deleted) has total size at least $2$.
- Hence, the number of surviving groups is at most $\text{SIZE}(I)/2$.
- All items in a group have the same size in $I'$.

## Lemma 28

*The total size of deleted items is at most $\mathcal{O}(\log(\mathrm{SIZE}(I)))$.*

- The total size of items in $G_1$ and $G_r$ is at most 6 as a group has total size at most 3.
- Consider a group $G_i$ that has strictly more items than $G_{i-1}$.
- It discards $n_i - n_{i-1}$ pieces of total size at most

$$3\frac{n_i - n_{i-1}}{n_i} \le \sum_{j=n_{i-1}+1}^{n_i} \frac{3}{j}$$

since the smallest piece has size at most $3/n_i$.
- Summing over all $i$ that have $n_i > n_{i-1}$ gives a bound of at most

$$\sum_{j=1}^{n_r} \frac{3}{j} \le \mathcal{O}(\log(\mathrm{SIZE}(I)))\ .$$

(note that $n_r \le \mathrm{SIZE}(I)$ since we assume that the size of each item is at least $1/\mathrm{SIZE}(I)$).

**Lemma 28**
*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 $1/\mathrm{SIZE}(I)$).

**Lemma 28**

*The total size of deleted items is at most $\mathcal{O}(\log(\text{SIZE}(I)))$.*

- The total size of items in $G_1$ and $G_r$ is at most $6$ as a group has total size at most $3$.
- Consider a group $G_i$ that has strictly more items than $G_{i-1}$.
- It discards $n_i - n_{i-1}$ pieces of total size at most

$$3\frac{n_i - n_{i-1}}{n_i} \le \sum_{j=n_{i-1}+1}^{n_i} \frac{3}{j}$$

  since the smallest piece has size at most $3/n_i$.
- Summing over all $i$ that have $n_i > n_{i-1}$ gives a bound of at most

$$\sum_{j=1}^{n_{r-1}} \frac{3}{j} \le \mathcal{O}(\log(\text{SIZE}(I))) \ .$$

  (note that $n_r \le \text{SIZE}(I)$ since we assume that the size of each item is at least $1/\text{SIZE}(I)$).

**Lemma 28**

*The total size of deleted items is at most $\mathcal{O}(\log(\text{SIZE}(I)))$.*

- The total size of items in $G_1$ and $G_r$ is at most $6$ as a group has total size at most $3$.
- Consider a group $G_i$ that has strictly more items than $G_{i-1}$.
- It discards $n_i - n_{i-1}$ pieces of total size at most

$$3\frac{n_i - n_{i-1}}{n_i} \leq \sum_{j=n_{i-1}+1}^{n_i} \frac{3}{j}$$

since the smallest piece has size at most $3/n_i$.

- Summing over all $i$ that have $n_i > n_{i-1}$ gives a bound of at most

$$\sum_{j=1}^{n_{r-1}} \frac{3}{j} \leq \mathcal{O}(\log(\text{SIZE}(I)))\ .$$

(note that $n_r \leq \text{SIZE}(I)$ since we assume that the size of each item is at least $1/\text{SIZE}(I)$).

**Lemma 28**

*The total size of deleted items is at most $\mathcal{O}(\log(\mathrm{SIZE}(I)))$.*

- The total size of items in $G_1$ and $G_r$ is at most $6$ as a group has total size at most $3$.
- Consider a group $G_i$ that has strictly more items than $G_{i-1}$.
- It discards $n_i - n_{i-1}$ pieces of total size at most

$$3\frac{n_i - n_{i-1}}{n_i} \leq \sum_{j=n_{i-1}+1}^{n_i} \frac{3}{j}$$

  since the smallest piece has size at most $3/n_i$.
- Summing over all $i$ that have $n_i > n_{i-1}$ gives a bound of at most

$$\sum_{j=1}^{n_{r-1}} \frac{3}{j} \leq \mathcal{O}(\log(\mathrm{SIZE}(I))) \ .$$

  (note that $n_r \leq \mathrm{SIZE}(I)$ since we assume that the size of each item is at least $1/\mathrm{SIZE}(I)$).

**Algorithm 1** BinPack

**if** $\text{SIZE}(I) < 10$ **then**
    pack remaining items greedily
Apply harmonic grouping to create instance $I'$; pack discarded items in at most $\mathcal{O}(\log(\text{SIZE}(I)))$ bins.
Let $x$ be optimal solution to configuration LP
Pack $\lfloor x_j \rfloor$ bins in configuration $T_j$ for all $j$; call the packed instance $I_1$.
Let $I_2$ be remaining pieces from $I'$
Pack $I_2$ via BinPack$(I_2)$

# Analysis

$$\mathrm{OPT_{LP}}(I_1) + \mathrm{OPT_{LP}}(I_2) \le \mathrm{OPT_{LP}}(I') \le \mathrm{OPT_{LP}}(I)$$

# Analysis

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

**Proof:**

- Each piece surviving in $I'$ can be mapped to a piece in $I$ of no lesser size. Hence, $\mathrm{OPT}_{\mathrm{LP}}(I') \le \mathrm{OPT}_{\mathrm{LP}}(I)$
- $\lfloor x_j \rfloor$ is feasible solution for $I_1$ (even integral).
- $x_j - \lfloor x_j \rfloor$ is feasible solution for $I_2$.

# Analysis

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

**Proof:**

- Each piece surviving in $I'$ can be mapped to a piece in $I$ of no lesser size. Hence, $\mathrm{OPT}_{\mathrm{LP}}(I') \le \mathrm{OPT}_{\mathrm{LP}}(I)$
- $\lfloor x_j \rfloor$ is feasible solution for $I_1$ (even integral).
- $x_j - \lfloor x_j \rfloor$ is feasible solution for $I_2$.

# Analysis

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

**Proof:**

- Each piece surviving in $I'$ can be mapped to a piece in $I$ of no lesser size. Hence, $\mathrm{OPT}_{\mathrm{LP}}(I') \le \mathrm{OPT}_{\mathrm{LP}}(I)$
- $\lfloor x_j \rfloor$ is feasible solution for $I_1$ (even integral).
- $x_j - \lfloor x_j \rfloor$ is feasible solution for $I_2$.

# Analysis

Each level of the recursion partitions pieces into three types

1. Pieces discarded at this level.
2. Pieces scheduled because they are in $I_1$.
3. Pieces in $I_2$ are handed down to the next level.

Pieces of type 2 summed over all recursion levels are packed into at most $\text{OPT}_{\text{LP}}$ 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

Each level of the recursion partitions pieces into three types

1. Pieces discarded at this level.
2. Pieces scheduled because they are in $I_1$.
3. Pieces in $I_2$ are handed down to the next level.

Pieces of type 2 summed over all recursion levels are packed into at most $\mathrm{OPT}_{\mathrm{LP}}$ many bins.

Pieces of type 1 are packed into at most

$$\mathcal{O}(\log(\mathrm{SIZE}(I))) \cdot L$$

many bins where $L$ is the number of recursion levels.

# Analysis

Each level of the recursion partitions pieces into three types

1. Pieces discarded at this level.
2. Pieces scheduled because they are in $I_1$.
3. Pieces in $I_2$ are handed down to the next level.

Pieces of type 2 summed over all recursion levels are packed into at most $\mathrm{OPT}_{\mathrm{LP}}$ many bins.

Pieces of type 1 are packed into at most

$$\mathcal{O}(\log(\mathrm{SIZE}(I))) \cdot L$$

many bins where $L$ is the number of recursion levels.

# Analysis

Each level of the recursion partitions pieces into three types

1. Pieces discarded at this level.
2. Pieces scheduled because they are in $I_1$.
3. Pieces in $I_2$ are handed down to the next level.

Pieces of type 2 summed over all recursion levels are packed into at most $\mathrm{OPT}_{\mathrm{LP}}$ many bins.

Pieces of type 1 are packed into at most

$$\mathcal{O}(\log(\mathrm{SIZE}(I))) \cdot L$$

many bins where $L$ is the number of recursion levels.

# Analysis

Each level of the recursion partitions pieces into three types

1. Pieces discarded at this level.
2. Pieces scheduled because they are in $I_1$.
3. Pieces in $I_2$ are handed down to the next level.

Pieces of type 2 summed over all recursion levels are packed into at most $\mathrm{OPT}_{\mathrm{LP}}$ many bins.

Pieces of type 1 are packed into at most

$$\mathcal{O}(\log(\mathrm{SIZE}(I))) \cdot L$$

many bins where $L$ is the number of recursion levels.

# Analysis

We can show that $\mathrm{SIZE}(I_2) \leq \mathrm{SIZE}(I)/2$. Hence, the number of recursion levels is only $\mathcal{O}(\log(\mathrm{SIZE}(I_{\text{original}})))$ in total.

- The number of non-zero entries in the solution to the configuration LP for $I'$ is at most the number of constraints, which is the number of different sizes ($\leq \mathrm{SIZE}(I)/2$).
- The total size of items in $I_2$ can be at most $\sum_{j=1}^{N} x_j - \lfloor x_j \rfloor$ which is at most the number of non-zero entries in the solution to the configuration LP.

# Analysis

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

# Analysis

We can show that $\text{SIZE}(I_2) \leq \text{SIZE}(I)/2$. Hence, the number of recursion levels is only $\mathcal{O}(\log(\text{SIZE}(I_{\text{original}})))$ in total.

- The number of non-zero entries in the solution to the configuration LP for $I'$ is at most the number of constraints, which is the number of different sizes ($\leq \text{SIZE}(I)/2$).
- The total size of items in $I_2$ can be at most $\sum_{j=1}^{N} x_j - \lfloor x_j \rfloor$ which is at most the number of non-zero entries in the solution to the configuration LP.

# How to solve the LP?

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

In total we have $b_i$ pieces of size $s_i$.

**Primal**

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

**Dual**

$$\begin{array}{llrcl} \max & & \sum_{i=1}^{m} y_i b_i & & \\ \text{s.t.} & \forall j \in \{1, \dots, N\} & \sum_{i=1}^{m} T_{ji} y_i & \leq & 1 \\ & \forall i \in \{1, \dots, m\} & y_i & \geq & 0 \end{array}$$

# How to solve the LP?

Let $T_1, \dots, T_N$ be the sequence of all possible configurations (a configuration $T_j$ has $T_{ji}$ pieces of size $s_i$).
In total we have $b_i$ pieces of size $s_i$.

**Primal**

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

**Dual**

$$\begin{array}{llrcl} \max & & \sum_{i=1}^{m} y_i b_i & & \\ \text{s.t.} & \forall j \in \{1, \dots, N\} & \sum_{i=1}^{m} T_{ji} y_i & \leq & 1 \\ & \forall i \in \{1, \dots, m\} & y_i & \geq & 0 \end{array}$$

# How to solve the LP?

Let $T_1, \dots, T_N$ be the sequence of all possible configurations (a configuration $T_j$ has $T_{ji}$ pieces of size $s_i$).
In total we have $b_i$ pieces of size $s_i$.

**Primal**

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

**Dual**

$$\begin{array}{llrcl} \max & & \sum_{i=1}^{m} y_i b_i & & \\ \text{s.t.} & \forall j \in \{1, \dots, N\} & \sum_{i=1}^{m} T_{ji} y_i & \leq & 1 \\ & \forall i \in \{1, \dots, m\} & y_i & \geq & 0 \end{array}$$

# Separation Oracle

Suppose that I am given variable assignment $y$ for the dual.

**How do I find a violated constraint?**

I have to find a configuration $T_j = (T_{j1}, \dots, T_{jm})$ that

is feasible, i.e.,

and has a large profit

But this is the Knapsack problem.

# Separation Oracle

Suppose that I am given variable assignment $y$ for the dual.

**How do I find a violated constraint?**

I have to find a configuration $T_j = (T_{j1}, \dots, T_{jm})$ that

- is feasible, i.e.,

$$\sum_{i=1}^{m} T_{ji} \cdot s_i \leq 1 \enspace ,$$

- and has a large profit

$$\sum_{i=1}^{m} T_{ji} y_i > 1$$

But this is the Knapsack problem.

# Separation Oracle

Suppose that I am given variable assignment $y$ for the dual.

**How do I find a violated constraint?**

I have to find a configuration $T_j = (T_{j1}, \ldots, T_{jm})$ that

- is feasible, i.e.,

$$\sum_{i=1}^{m} T_{ji} \cdot s_i \leq 1 \ ,$$

- and has a large profit

$$\sum_{i=1}^{m} T_{ji} y_i > 1$$

But this is the Knapsack problem.

# Separation Oracle

Suppose that I am given variable assignment $y$ for the dual.

**How do I find a violated constraint?**

I have to find a configuration $T_j = (T_{j1}, \ldots, T_{jm})$ that

- is feasible, i.e.,

$$\sum_{i=1}^{m} T_{ji} \cdot s_i \leq 1 \ ,$$

- and has a large profit

$$\sum_{i=1}^{m} T_{ji} y_i > 1$$

But this is the Knapsack problem.

# Separation Oracle

We have FPTAS for Knapsack. This means if a constraint is violated with $1 + \epsilon' = 1 + \frac{\epsilon}{1-\epsilon}$ we find it, since we can obtain at least $(1 - \epsilon)$ of the optimal profit.

The solution we get is feasible for:

**Dual′**

$$\begin{array}{llrcl} \max & & \sum_{i=1}^{m} y_i b_i & & \\ \text{s.t.} & \forall j \in \{1, \dots, N\} & \sum_{i=1}^{m} T_{ji} y_i & \le & 1 + \epsilon' \\ & \forall i \in \{1, \dots, m\} & y_i & \ge & 0 \end{array}$$

**Primal′**

$$\begin{array}{llrcl} \min & & (1 + \epsilon') \sum_{j=1}^{N} x_j & & \\ \text{s.t.} & \forall i \in \{1 \dots m\} & \sum_{j=1}^{N} T_{ji} x_j & \ge & b_i \\ & \forall j \in \{1, \dots, N\} & x_j & \ge & 0 \end{array}$$

# Separation Oracle

We have FPTAS for Knapsack. This means if a constraint is violated with $1 + \epsilon' = 1 + \frac{\epsilon}{1-\epsilon}$ we find it, since we can obtain at least $(1 - \epsilon)$ of the optimal profit.

The solution we get is feasible for:

**Dual′**

$$\begin{array}{llrcl} \max & & \sum_{i=1}^{m} y_i b_i & & \\ \text{s.t.} & \forall j \in \{1,\ldots,N\} & \sum_{i=1}^{m} T_{ji} y_i & \le & 1 + \epsilon' \\ & \forall i \in \{1,\ldots,m\} & y_i & \ge & 0 \end{array}$$

**Primal′**

$$\begin{array}{llrcl} \min & & (1+\epsilon') \sum_{j=1}^{N} x_j & & \\ \text{s.t.} & \forall i \in \{1\ldots m\} & \sum_{j=1}^{N} T_{ji} x_j & \ge & b_i \\ & \forall j \in \{1,\ldots,N\} & x_j & \ge & 0 \end{array}$$

# Separation Oracle

We have FPTAS for Knapsack. This means if a constraint is violated with $1 + \epsilon' = 1 + \frac{\epsilon}{1-\epsilon}$ we find it, since we can obtain at least $(1 - \epsilon)$ of the optimal profit.

The solution we get is feasible for:

**Dual′**

$$\begin{array}{llrcl} \max & & \sum_{i=1}^{m} y_i b_i & & \\ \text{s.t.} & \forall j \in \{1, \dots, N\} & \sum_{i=1}^{m} T_{ji} y_i & \leq & 1 + \epsilon' \\ & \forall i \in \{1, \dots, m\} & y_i & \geq & 0 \end{array}$$

**Primal′**

$$\begin{array}{llrcl} \min & & (1 + \epsilon') \sum_{j=1}^{N} x_j & & \\ \text{s.t.} & \forall i \in \{1 \dots m\} & \sum_{j=1}^{N} T_{ji} x_j & \geq & b_i \\ & \forall j \in \{1, \dots, N\} & x_j & \geq & 0 \end{array}$$

## Separation Oracle

We have FPTAS for Knapsack. This means if a constraint is violated with $1+\epsilon' = 1+\frac{\epsilon}{1-\epsilon}$ we find it, since we can obtain at least $(1-\epsilon)$ of the optimal profit.

The solution we get is feasible for:

**Dual′**

$$\begin{array}{llrcl} \max & & \sum_{i=1}^{m} y_i b_i & & \\ \text{s.t.} & \forall j \in \{1,\dots,N\} & \sum_{i=1}^{m} T_{ji} y_i & \leq & 1+\epsilon' \\ & \forall i \in \{1,\dots,m\} & y_i & \geq & 0 \end{array}$$

**Primal′**

$$\begin{array}{llrcl} \min & & (1+\epsilon') \sum_{j=1}^{N} x_j & & \\ \text{s.t.} & \forall i \in \{1\dots m\} & \sum_{j=1}^{N} T_{ji} x_j & \geq & b_i \\ & \forall j \in \{1,\dots,N\} & x_j & \geq & 0 \end{array}$$

# Separation Oracle

If the value of the computed dual solution (which may be infeasible) is $z$ then

$$\mathrm{OPT} \leq z \leq (1 + \epsilon')\mathrm{OPT}$$

**How do we get good primal solution (not just the value)?**

- The constraints used when computing $z$ certify that the solution is feasible for DUAL′.
- Suppose that we drop all unused constraints in DUAL. We will compute the same solution feasible for DUAL′.
- Let DUAL″ be DUAL without unused constraints.
- The dual to DUAL″ is PRIMAL where we ignore variables for which the corresponding dual constraint has not been used.
- The optimum value for PRIMAL″ is at most $(1 + \epsilon')\mathrm{OPT}$.
- We can compute the corresponding solution in polytime.

## Separation Oracle

If the value of the computed dual solution (which may be infeasible) is $z$ then

$$\mathrm{OPT} \leq z \leq (1+\epsilon')\mathrm{OPT}$$

**How do we get good primal solution (not just the value)?**

- The constraints used when computing $z$ certify that the solution is feasible for $\mathrm{DUAL}'$.
- Suppose that we drop all unused constraints in $\mathrm{DUAL}$. We will compute the same solution feasible for $\mathrm{DUAL}'$.
- Let $\mathrm{DUAL}''$ be $\mathrm{DUAL}$ without unused constraints.
- The dual to $\mathrm{DUAL}''$ is $\mathrm{PRIMAL}$ where we ignore variables for which the corresponding dual constraint has not been used.
- The optimum value for $\mathrm{PRIMAL}''$ is at most $(1+\epsilon')\mathrm{OPT}$.
- We can compute the corresponding solution in polytime.

# Separation Oracle

If the value of the computed dual solution (which may be infeasible) is $z$ then

$$\mathrm{OPT} \le z \le (1+\epsilon')\mathrm{OPT}$$

**How do we get good primal solution (not just the value)?**

- The constraints used when computing $z$ certify that the solution is feasible for $\mathrm{DUAL}'$.
- Suppose that we drop all unused constraints in $\mathrm{DUAL}$. We will compute the same solution feasible for $\mathrm{DUAL}'$.
- Let $\mathrm{DUAL}''$ be $\mathrm{DUAL}$ without unused constraints.
- The dual to $\mathrm{DUAL}''$ is $\mathrm{PRIMAL}$ where we ignore variables for which the corresponding dual constraint has not been used.
- The optimum value for $\mathrm{PRIMAL}''$ is at most $(1+\epsilon')\mathrm{OPT}$.
- We can compute the corresponding solution in polytime.

## Separation Oracle

If the value of the computed dual solution (which may be infeasible) is $z$ then

$$\text{OPT} \leq z \leq (1 + \epsilon')\text{OPT}$$

**How do we get good primal solution (not just the value)?**

- The constraints used when computing $z$ certify that the solution is feasible for $\text{DUAL}'$.
- Suppose that we drop all unused constraints in $\text{DUAL}$. We will compute the same solution feasible for $\text{DUAL}'$.
- Let $\text{DUAL}''$ be $\text{DUAL}$ without unused constraints.
- The dual to $\text{DUAL}''$ is $\text{PRIMAL}$ where we ignore variables for which the corresponding dual constraint has not been used.
- The optimum value for $\text{PRIMAL}''$ is at most $(1 + \epsilon')\text{OPT}$.
- We can compute the corresponding solution in polytime.

# Separation Oracle

If the value of the computed dual solution (which may be infeasible) is $z$ then

$$\mathrm{OPT} \leq z \leq (1+\epsilon')\mathrm{OPT}$$

**How do we get good primal solution (not just the value)?**

- The constraints used when computing $z$ certify that the solution is feasible for $\mathrm{DUAL}'$.
- Suppose that we drop all unused constraints in $\mathrm{DUAL}$. We will compute the same solution feasible for $\mathrm{DUAL}'$.
- Let $\mathrm{DUAL}''$ be $\mathrm{DUAL}$ without unused constraints.
- The dual to $\mathrm{DUAL}''$ is $\mathrm{PRIMAL}$ where we ignore variables for which the corresponding dual constraint has not been used.
- The optimum value for $\mathrm{PRIMAL}''$ is at most $(1+\epsilon')\mathrm{OPT}$.
- We can compute the corresponding solution in polytime.

# Separation Oracle

If the value of the computed dual solution (which may be infeasible) is $z$ then

$$\mathrm{OPT} \le z \le (1 + \epsilon')\mathrm{OPT}$$

**How do we get good primal solution (not just the value)?**

- The constraints used when computing $z$ certify that the solution is feasible for $\mathrm{DUAL}'$.
- Suppose that we drop all unused constraints in $\mathrm{DUAL}$. We will compute the same solution feasible for $\mathrm{DUAL}'$.
- Let $\mathrm{DUAL}''$ be $\mathrm{DUAL}$ without unused constraints.
- The dual to $\mathrm{DUAL}''$ is $\mathrm{PRIMAL}$ where we ignore variables for which the corresponding dual constraint has not been used.
- The optimum value for $\mathrm{PRIMAL}''$ is at most $(1 + \epsilon')\mathrm{OPT}$.
- We can compute the corresponding solution in polytime.

## Separation Oracle

If the value of the computed dual solution (which may be infeasible) is $z$ then

$$\mathrm{OPT} \leq z \leq (1 + \epsilon')\mathrm{OPT}$$

**How do we get good primal solution (not just the value)?**

- The constraints used when computing $z$ certify that the solution is feasible for $\mathrm{DUAL}'$.
- Suppose that we drop all unused constraints in $\mathrm{DUAL}$. We will compute the same solution feasible for $\mathrm{DUAL}'$.
- Let $\mathrm{DUAL}''$ be $\mathrm{DUAL}$ without unused constraints.
- The dual to $\mathrm{DUAL}''$ is $\mathrm{PRIMAL}$ where we ignore variables for which the corresponding dual constraint has not been used.
- The optimum value for $\mathrm{PRIMAL}''$ is at most $(1 + \epsilon')\mathrm{OPT}$.
- We can compute the corresponding solution in polytime.

This gives that overall we need at most

$$(1+\epsilon')\mathrm{OPT}_{\mathrm{LP}}(I) + \mathcal{O}(\log^2(\mathrm{SIZE}(I)))$$

bins.

We can choose $\epsilon' = \frac{1}{\mathrm{OPT}}$ as $\mathrm{OPT} \le$ #items and since we have a fully polynomial time approximation scheme (FPTAS) for knapsack.

This gives that overall we need at most

$$(1+\epsilon')\mathrm{OPT}_{\mathrm{LP}}(I) + \mathcal{O}(\log^2(\mathrm{SIZE}(I)))$$

bins.

We can choose $\epsilon' = \frac{1}{\mathrm{OPT}}$ as $\mathrm{OPT} \leq$ #items and since we have a fully polynomial time approximation scheme (FPTAS) for knapsack.

## Lemma 29 (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 30

*For* $0 \leq \delta \leq 1$ *we have that*

$$\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{U} \leq e^{-U\delta^2/3}$$

*and*

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

# Proof of Chernoff Bounds

**Markovs Inequality**:
Let $X$ be random variable taking non-negative values.
Then

$$\Pr[X \geq a] \leq \mathrm{E}[X]/a$$

Trivial!

# Proof of Chernoff Bounds

**Markovs Inequality**:

Let $X$ be random variable taking non-negative values.

Then

$$\Pr[X \geq a] \leq \mathrm{E}[X]/a$$

**Trivial!**

# Proof of Chernoff Bounds

**Hence:**

$$\Pr[X \geq (1+\delta)U] \leq \frac{\mathrm{E}[X]}{(1+\delta)U}$$

# Proof of Chernoff Bounds

**Hence:**

$$\Pr[X \geq (1+\delta)U] \leq \frac{\mathrm{E}[X]}{(1+\delta)U} \approx \frac{1}{1+\delta}$$

# Proof of Chernoff Bounds

**Hence:**

$$\Pr[X \geq (1+\delta)U] \leq \frac{\mathrm{E}[X]}{(1+\delta)U} \approx \frac{1}{1+\delta}$$

**That's awfully weak :(**

# Proof of Chernoff Bounds

Set $p_i = \Pr[X_i = 1]$. Assume $p_i > 0$ for all $i$.

# Proof of Chernoff Bounds

Set $p_i = \Pr[X_i = 1]$. Assume $p_i > 0$ for all $i$.

**Cool Trick:**

$$\Pr[X \geq (1+\delta)U] = \Pr[e^{tX} \geq e^{t(1+\delta)U}]$$

# Proof of Chernoff Bounds

Set $p_i = \Pr[X_i = 1]$. Assume $p_i > 0$ for all $i$.

**Cool Trick:**

$$\Pr[X \geq (1+\delta)U] = \Pr[e^{tX} \geq e^{t(1+\delta)U}]$$

Now, we apply Markov:

$$\Pr[e^{tX} \geq e^{t(1+\delta)U}] \leq \frac{\mathrm{E}[e^{tX}]}{e^{t(1+\delta)U}} \enspace .$$

# Proof of Chernoff Bounds

Set $p_i = \Pr[X_i = 1]$. Assume $p_i > 0$ for all $i$.

**Cool Trick:**

$$\Pr[X \geq (1+\delta)U] = \Pr[e^{tX} \geq e^{t(1+\delta)U}]$$

Now, we apply Markov:

$$\Pr[e^{tX} \geq e^{t(1+\delta)U}] \leq \frac{\mathrm{E}[e^{tX}]}{e^{t(1+\delta)U}} \enspace .$$

**This may be a lot better (!?)**

# Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right]$$

# Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right] = \mathrm{E}\left[e^{t\sum_i X_i}\right]$$

# Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right] = \mathrm{E}\left[e^{t\sum_i X_i}\right] = \mathrm{E}\left[\prod_i e^{tX_i}\right]$$

# Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right] = \mathrm{E}\left[e^{t\sum_i X_i}\right] = \mathrm{E}\left[\prod_i e^{tX_i}\right] = \prod_i \mathrm{E}\left[e^{tX_i}\right]$$

# Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right] = \mathrm{E}\left[e^{t\sum_i X_i}\right] = \mathrm{E}\left[\prod_i e^{tX_i}\right] = \prod_i \mathrm{E}\left[e^{tX_i}\right]$$

$$\mathrm{E}\left[e^{tX_i}\right]$$

# Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right] = \mathrm{E}\left[e^{t\sum_i X_i}\right] = \mathrm{E}\left[\prod_i e^{tX_i}\right] = \prod_i \mathrm{E}\left[e^{tX_i}\right]$$

$$\mathrm{E}\left[e^{tX_i}\right] = (1 - p_i) + p_i e^t$$

# Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right] = \mathrm{E}\left[e^{t\sum_i X_i}\right] = \mathrm{E}\left[\prod_i e^{tX_i}\right] = \prod_i \mathrm{E}\left[e^{tX_i}\right]$$

$$\mathrm{E}\left[e^{tX_i}\right] = (1-p_i) + p_i e^t = 1 + p_i(e^t - 1)$$

# Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right] = \mathrm{E}\left[e^{t\sum_i X_i}\right] = \mathrm{E}\left[\prod_i e^{tX_i}\right] = \prod_i \mathrm{E}\left[e^{tX_i}\right]$$

$$\mathrm{E}\left[e^{tX_i}\right] = (1 - p_i) + p_i e^t = 1 + p_i(e^t - 1) \leq e^{p_i(e^t - 1)}$$

# Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right] = \mathrm{E}\left[e^{t\sum_i X_i}\right] = \mathrm{E}\left[\prod_i e^{tX_i}\right] = \prod_i \mathrm{E}\left[e^{tX_i}\right]$$

$$\mathrm{E}\left[e^{tX_i}\right] = (1-p_i) + p_i e^t = 1 + p_i(e^t - 1) \leq e^{p_i(e^t-1)}$$

$$\prod_i \mathrm{E}\left[e^{tX_i}\right]$$

# Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right] = \mathrm{E}\left[e^{t\sum_i X_i}\right] = \mathrm{E}\left[\prod_i e^{tX_i}\right] = \prod_i \mathrm{E}\left[e^{tX_i}\right]$$

$$\mathrm{E}\left[e^{tX_i}\right] = (1-p_i) + p_i e^t = 1 + p_i(e^t - 1) \le e^{p_i(e^t-1)}$$

$$\prod_i \mathrm{E}\left[e^{tX_i}\right] \le \prod_i e^{p_i(e^t-1)}$$

# Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right] = \mathrm{E}\left[e^{t\sum_i X_i}\right] = \mathrm{E}\left[\prod_i e^{tX_i}\right] = \prod_i \mathrm{E}\left[e^{tX_i}\right]$$

$$\mathrm{E}\left[e^{tX_i}\right] = (1-p_i) + p_i e^t = 1 + p_i(e^t - 1) \le e^{p_i(e^t-1)}$$

$$\prod_i \mathrm{E}\left[e^{tX_i}\right] \le \prod_i e^{p_i(e^t-1)} = e^{\sum p_i(e^t-1)}$$

# Proof of Chernoff Bounds

$$\mathrm{E}\left[e^{tX}\right] = \mathrm{E}\left[e^{t\sum_i X_i}\right] = \mathrm{E}\left[\prod_i e^{tX_i}\right] = \prod_i \mathrm{E}\left[e^{tX_i}\right]$$

$$\mathrm{E}\left[e^{tX_i}\right] = (1 - p_i) + p_i e^t = 1 + p_i(e^t - 1) \le e^{p_i(e^t-1)}$$

$$\prod_i \mathrm{E}\left[e^{tX_i}\right] \le \prod_i e^{p_i(e^t-1)} = e^{\sum p_i(e^t-1)} = e^{(e^t-1)U}$$

Now, we apply Markov:

$$\begin{aligned}\Pr[X \geq (1+\delta)U] &= \Pr[e^{tX} \geq e^{t(1+\delta)U}] \\ &\leq \frac{\mathrm{E}[e^{tX}]}{e^{t(1+\delta)U}}\end{aligned}$$

Now, we apply Markov:

$$\begin{aligned}\Pr[X \geq (1+\delta)U] &= \Pr[e^{tX} \geq e^{t(1+\delta)U}] \\ &\leq \frac{\mathrm{E}[e^{tX}]}{e^{t(1+\delta)U}} \leq \frac{e^{(e^t-1)U}}{e^{t(1+\delta)U}}\end{aligned}$$

Now, we apply Markov:

$$\begin{aligned}\Pr[X \geq (1+\delta)U] &= \Pr[e^{tX} \geq e^{t(1+\delta)U}] \\ &\leq \frac{\mathrm{E}[e^{tX}]}{e^{t(1+\delta)U}} \leq \frac{e^{(e^t-1)U}}{e^{t(1+\delta)U}}\end{aligned}$$

We choose $t = \ln(1+\delta)$.

Now, we apply Markov:

$$\Pr[X \geq (1+\delta)U] = \Pr[e^{tX} \geq e^{t(1+\delta)U}]$$

$$\leq \frac{\mathrm{E}[e^{tX}]}{e^{t(1+\delta)U}} \leq \frac{e^{(e^t-1)U}}{e^{t(1+\delta)U}} \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^U$$

We choose $t = \ln(1+\delta)$.

## Lemma 31

*For* $0 \leq \delta \leq 1$ *we have that*

$$\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{U} \leq e^{-U\delta^2/3}$$

*and*

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

Show:

$$\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{U} \leq e^{-U\delta^2/3}$$

Show:

$$\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{U} \leq e^{-U\delta^2/3}$$

Take logarithms:

$$U(\delta - (1+\delta)\ln(1+\delta)) \leq -U\delta^2/3$$

Show:

$$\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{U} \le e^{-U\delta^2/3}$$

Take logarithms:

$$U(\delta - (1+\delta)\ln(1+\delta)) \le -U\delta^2/3$$

True for $\delta = 0$.

Show:

$$\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{U} \le e^{-U\delta^2/3}$$

Take logarithms:

$$U(\delta - (1+\delta)\ln(1+\delta)) \le -U\delta^2/3$$

True for $\delta = 0$. Divide by $U$ and take derivatives:

$$-\ln(1+\delta) \le -2\delta/3$$

**Reason:**

As long as derivative of left side is smaller than derivative of right side the inequality holds.

$$f(\delta) := -\ln(1+\delta) + 2\delta/3 \leq 0$$

$$f(\delta) := -\ln(1+\delta) + 2\delta/3 \le 0$$

A convex function ($f''(\delta) \ge 0$) on an interval takes maximum at the boundaries.

$$f(\delta) := -\ln(1+\delta) + 2\delta/3 \leq 0$$

A convex function ($f''(\delta) \geq 0$) on an interval takes maximum at the boundaries.

$$f'(\delta) = -\frac{1}{1+\delta} + 2/3$$

$$f(\delta) := -\ln(1+\delta) + 2\delta/3 \leq 0$$

A convex function ($f''(\delta) \geq 0$) on an interval takes maximum at the boundaries.

$$f'(\delta) = -\frac{1}{1+\delta} + 2/3 \qquad f''(\delta) = \frac{1}{(1+\delta)^2}$$

$$f(\delta) := -\ln(1+\delta) + 2\delta/3 \leq 0$$

A convex function ($f''(\delta) \geq 0$) on an interval takes maximum at the boundaries.

$$f'(\delta) = -\frac{1}{1+\delta} + 2/3 \qquad f''(\delta) = \frac{1}{(1+\delta)^2}$$

$f(0) = 0$ and $f(1) = -\ln(2) + 2/3 < 0$

For $\delta \geq 1$ we show

$$\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{U} \leq e^{-U\delta/3}$$

For $\delta \geq 1$ we show

$$\left( \frac{e^{\delta}}{(1+\delta)^{1+\delta}} \right)^{U} \leq e^{-U\delta/3}$$

Take logarithms:

$$U(\delta - (1+\delta)\ln(1+\delta)) \leq -U\delta/3$$

For $\delta \geq 1$ we show

$$\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{U} \leq e^{-U\delta/3}$$

Take logarithms:

$$U(\delta - (1+\delta)\ln(1+\delta)) \leq -U\delta/3$$

True for $\delta = 0$.

For $\delta \geq 1$ we show

$$\left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{U} \leq e^{-U\delta/3}$$

Take logarithms:

$$U(\delta - (1+\delta)\ln(1+\delta)) \leq -U\delta/3$$

True for $\delta = 0$. Divide by $U$ and take derivatives:

$$-\ln(1+\delta) \leq -1/3 \iff \ln(1+\delta) \geq 1/3 \quad \text{(true)}$$

**Reason:**
As long as derivative of left side is smaller than derivative of right side the inequality holds.

Show:

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

Show:

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

Take logarithms:

$$L(-\delta - (1-\delta)\ln(1-\delta)) \le -L\delta^2/2$$

Show:

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

Take logarithms:

$$L(-\delta - (1-\delta)\ln(1-\delta)) \le -L\delta^2/2$$

True for $\delta = 0$.

Show:

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

Take logarithms:

$$L(-\delta - (1-\delta)\ln(1-\delta)) \le -L\delta^2/2$$

True for $\delta = 0$. Divide by $L$ and take derivatives:

$$\ln(1-\delta) \le -\delta$$

**Reason:**

As long as derivative of left side is smaller than derivative of right side the inequality holds.

$$\ln(1-\delta) \le -\delta$$

$$\ln(1-\delta) \le -\delta$$

True for $\delta = 0$.

$$\ln(1-\delta) \le -\delta$$

True for $\delta = 0$. Take derivatives:

$$-\frac{1}{1-\delta} \le -1$$

$$\ln(1 - \delta) \le -\delta$$

True for $\delta = 0$. Take derivatives:

$$-\frac{1}{1 - \delta} \le -1$$

This holds for $0 \le \delta < 1$.

# Integer Multicommodity Flows

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

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

# Integer Multicommodity Flows

**Randomized Rounding:**
For each $i$ choose one path from the set $\mathcal{P}_i$ at random according to the probability distribution given by the Linear Programming solution.

## Theorem 32

*If $W^* \geq c \ln n$ for some constant $c$, then with probability at least $n^{-c/3}$ the total number of paths using any edge is at most $W^* + \sqrt{cW^* \ln n}$.*

## Theorem 33

*With probability at least $n^{-c/3}$ the total number of paths using any edge is at most $W^* + c \ln n$.*

# Integer Multicommodity Flows

Let $X_e^i$ be a random variable that indicates whether the path for $s_i$-$t_i$ uses edge $e$.

Then the number of paths using edge $e$ is $Y_e = \sum_i X_e^i$.

$$E[Y_e] = \sum_i \sum_{p \in \mathcal{P}_i : e \in p} x_p^* = \sum_{p : e \in P} x_p^* \le W^*$$

# Integer Multicommodity Flows

Let $X_e^i$ be a random variable that indicates whether the path for $s_i$-$t_i$ uses edge $e$.

Then the number of paths using edge $e$ is $Y_e = \sum_i X_e^i$.

$$E[Y_e] = \sum_i \sum_{p \in \mathcal{P}_i : e \in p} x_p^* = \sum_{p : e \in P} x_p^* \le W^*$$

# Integer Multicommodity Flows

Let $X_e^i$ be a random variable that indicates whether the path for $s_i$-$t_i$ uses edge $e$.

Then the number of paths using edge $e$ is $Y_e = \sum_i X_e^i$.

# Integer Multicommodity Flows

Let $X_e^i$ be a random variable that indicates whether the path for $s_i$-$t_i$ uses edge $e$.

Then the number of paths using edge $e$ is $Y_e = \sum_i X_e^i$.

$$E[Y_e] = \sum_i \sum_{p \in \mathcal{P}_i : e \in p} x_p^* = \sum_{p : e \in P} x_p^* \leq W^*$$

# Integer Multicommodity Flows

Let $X_e^i$ be a random variable that indicates whether the path for $s_i$-$t_i$ uses edge $e$.

Then the number of paths using edge $e$ is $Y_e = \sum_i X_e^i$.

$$E[Y_e] = \sum_i \sum_{p \in \mathcal{P}_i : e \in p} x_p^* = \sum_{p : e \in P} x_p^* \le W^*$$

# Integer Multicommodity Flows

Let $X_e^i$ be a random variable that indicates whether the path for $s_i$-$t_i$ uses edge $e$.

Then the number of paths using edge $e$ is $Y_e = \sum_i X_e^i$.

$$E[Y_e] = \sum_i \sum_{p \in \mathcal{P}_i : e \in p} x_p^* = \sum_{p : e \in P} x_p^* \le W^*$$

# Integer Multicommodity Flows

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

Then

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

# Integer Multicommodity Flows

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

Then

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

# 18 MAXSAT

**Problem definition:**

- $n$ Boolean variables
- $m$ clauses $C_1, \dots, C_m$. For example

$$C_7 = x_3 \vee \bar{x}_5 \vee \bar{x}_9$$

- Non-negative weight $w_j$ for each clause $C_j$.
- Find an assignment of true/false to the variables sucht that the total weight of clauses that are satisfied is maximum.

# 18 MAXSAT

**Problem definition:**

- $n$ Boolean variables
- $m$ clauses $C_1, \dots, C_m$. For example

$$C_7 = x_3 \vee \bar{x}_5 \vee \bar{x}_9$$

- Non-negative weight $w_j$ for each clause $C_j$.
- Find an assignment of true/false to the variables sucht that the total weight of clauses that are satisfied is maximum.

# 18 MAXSAT

**Problem definition:**

- $n$ Boolean variables
- $m$ clauses $C_1, \dots, C_m$. For example

$$C_7 = x_3 \vee \bar{x}_5 \vee \bar{x}_9$$

- Non-negative weight $w_j$ for each clause $C_j$.
- Find an assignment of true/false to the variables sucht that the total weight of clauses that are satisfied is maximum.

# 18 MAXSAT

**Problem definition:**

- $n$ Boolean variables
- $m$ clauses $C_1, \dots, C_m$. For example

$$C_7 = x_3 \vee \bar{x}_5 \vee \bar{x}_9$$

- Non-negative weight $w_j$ for each clause $C_j$.
- Find an assignment of true/false to the variables sucht that the total weight of clauses that are satisfied is maximum.

# 18 MAXSAT

**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.

# 18 MAXSAT

**Terminology:**

- A variable $x_i$ and its negation $\bar{x}_i$ are called literals.
- Hence, each clause consists of a set of literals (i.e., no duplications: $x_i \vee x_i \vee \bar{x}_j$ is **not** a clause).
- We assume a clause does not contain $x_i$ and $\bar{x}_i$ for any $i$.
- $x_i$ is called a positive literal while the negation $\bar{x}_i$ is called a negative literal.
- For a given clause $C_j$ the number of its literals is called its length or size and denoted with $\ell_j$.
- Clauses of length one are called unit clauses.

# 18 MAXSAT

**Terminology:**

- A variable $x_i$ and its negation $\bar{x}_i$ are called literals.
- Hence, each clause consists of a set of literals (i.e., no duplications: $x_i \vee x_i \vee \bar{x}_j$ is **not** a clause).
- We assume a clause does not contain $x_i$ and $\bar{x}_i$ for any $i$.
- $x_i$ is called a positive literal while the negation $\bar{x}_i$ is called a negative literal.
- For a given clause $C_j$ the number of its literals is called its length or size and denoted with $\ell_j$.
- Clauses of length one are called unit clauses.

# 18 MAXSAT

**Terminology:**

- A variable $x_i$ and its negation $\bar{x}_i$ are called literals.
- Hence, each clause consists of a set of literals (i.e., no duplications: $x_i \vee x_i \vee \bar{x}_j$ is **not** a clause).
- We assume a clause does not contain $x_i$ and $\bar{x}_i$ for any $i$.
- $x_i$ is called a positive literal while the negation $\bar{x}_i$ is called a negative literal.
- For a given clause $C_j$ the number of its literals is called its length or size and denoted with $\ell_j$.
- Clauses of length one are called unit clauses.

# 18 MAXSAT

**Terminology:**

- A variable $x_i$ and its negation $\bar{x}_i$ are called literals.
- Hence, each clause consists of a set of literals (i.e., no duplications: $x_i \vee x_i \vee \bar{x}_j$ is **not** a clause).
- We assume a clause does not contain $x_i$ and $\bar{x}_i$ for any $i$.
- $x_i$ is called a positive literal while the negation $\bar{x}_i$ is called a negative literal.
- For a given clause $C_j$ the number of its literals is called its length or size and denoted with $\ell_j$.
- Clauses of length one are called unit clauses.

# 18 MAXSAT

**Terminology:**

- A variable $x_i$ and its negation $\bar{x}_i$ are called literals.
- Hence, each clause consists of a set of literals (i.e., no duplications: $x_i \vee x_i \vee \bar{x}_j$ is **not** a clause).
- We assume a clause does not contain $x_i$ and $\bar{x}_i$ for any $i$.
- $x_i$ is called a positive literal while the negation $\bar{x}_i$ is called a negative literal.
- For a given clause $C_j$ the number of its literals is called its length or size and denoted with $\ell_j$.
- 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

$$W = \sum_j w_j X_j$$

Define random variable $X_j$ with

$$X_j = \begin{cases} 1 & \text{if } C_j \text{ satisfied} \\ 0 & \text{otw.} \end{cases}$$

Then the total weight $W$ of satisfied clauses is given by

$$W = \sum_j w_j X_j$$

$E[W]$

$$E[W] = \sum_j w_j E[X_j]$$

$$
\begin{aligned}
E[W] &= \sum_j w_j E[X_j] \\
&= \sum_j w_j \Pr[C_j \text{ is satisified}]
\end{aligned}
$$

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

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

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

# MAXSAT: LP formulation

- Let for a clause $C_j$, $P_j$ be the set of positive literals and $N_j$ the set of negative literals.

$$C_j = \bigvee_{j \in P_j} x_i \vee \bigvee_{j \in N_j} \bar{x}_i$$

$$\begin{array}{llrcl} \max & & \multicolumn{3}{c}{\sum_j w_j z_j} \\ \text{s.t.} & \forall j & \sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) & \geq & z_j \\ & \forall i & y_i & \in & \{0,1\} \\ & \forall j & z_j & \leq & 1 \end{array}$$

# MAXSAT: LP formulation

- Let for a clause $C_j$, $P_j$ be the set of positive literals and $N_j$ the set of negative literals.

$$C_j = \bigvee_{j \in P_j} x_i \vee \bigvee_{j \in N_j} \bar{x}_i$$

$$\begin{array}{llrcl} \max & & \sum_j w_j z_j & & \\ \text{s.t.} & \forall j & \sum_{i \in P_j} y_i + \sum_{i \in N_j} (1 - y_i) & \geq & z_j \\ & \forall i & y_i & \in & \{0, 1\} \\ & \forall j & z_j & \leq & 1 \end{array}$$

# MAXSAT: Randomized Rounding

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

**Lemma 34 (Geometric Mean $\leq$ Arithmetic Mean)**

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

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

## Definition 35

A function $f$ on an interval $I$ is concave if for any two points $s$ and $r$ from $I$ and any $\lambda \in [0,1]$ we have

$$f(\lambda s + (1-\lambda)r) \geq \lambda f(s) + (1-\lambda)f(r)$$

## Lemma 36

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

$$\begin{aligned} f(\lambda) &= f((1-\lambda)0 + \lambda 1) \\ &\geq (1-\lambda)f(0) + \lambda f(1) \\ &= a + \lambda b \end{aligned}$$

*for $\lambda \in [0,1]$.*

## Definition 35

A function $f$ on an interval $I$ is concave if for any two points $s$ and $r$ from $I$ and any $\lambda \in [0,1]$ we have

$$f(\lambda s + (1-\lambda)r) \geq \lambda f(s) + (1-\lambda)f(r)$$

## Lemma 36

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

$$\begin{aligned} f(\lambda) &= f((1-\lambda)0 + \lambda 1) \\ &\geq (1-\lambda)f(0) + \lambda f(1) \\ &= a + \lambda b \end{aligned}$$

*for $\lambda \in [0,1]$.*

## Definition 35

A function $f$ on an interval $I$ is concave if for any two points $s$ and $r$ from $I$ and any $\lambda \in [0,1]$ we have

$$f(\lambda s + (1-\lambda)r) \geq \lambda f(s) + (1-\lambda)f(r)$$

## Lemma 36

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

$$\begin{aligned} f(\lambda) &= f((1-\lambda)0 + \lambda 1) \\ &\geq (1-\lambda)f(0) + \lambda f(1) \\ &= a + \lambda b \end{aligned}$$

*for $\lambda \in [0,1]$.*

## Definition 35

A function $f$ on an interval $I$ is concave if for any two points $s$ and $r$ from $I$ and any $\lambda \in [0,1]$ we have

$$f(\lambda s + (1-\lambda)r) \geq \lambda f(s) + (1-\lambda)f(r)$$

## Lemma 36

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

$$\begin{aligned} f(\lambda) &= f((1-\lambda)0 + \lambda 1) \\ &\geq (1-\lambda)f(0) + \lambda f(1) \\ &= a + \lambda b \end{aligned}$$

*for $\lambda \in [0,1]$.*

$\Pr[C_j \text{ not satisfied}]$

$$\Pr[C_j \text{ not satisfied}] = \prod_{i \in P_j} (1 - y_i) \prod_{i \in N_j} y_i$$

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

$$\Pr[C_j \text{ not satisfied}] = \prod_{i\in P_j}(1-y_i)\prod_{i\in N_j} y_i$$

$$\leq \left[\frac{1}{\ell_j}\left(\sum_{i\in P_j}(1-y_i) + \sum_{i\in N_j} y_i\right)\right]^{\ell_j}$$

$$= \left[1 - \frac{1}{\ell_j}\left(\sum_{i\in P_j} y_i + \sum_{i\in N_j}(1-y_i)\right)\right]^{\ell_j}$$

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

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

$$\Pr[C_j \text{ satisfied}]$$

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

$$\Pr[C_j \text{ satisfied}] \geq 1 - \left(1 - \frac{z_j}{\ell_j}\right)^{\ell_j}$$

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

$$\Pr[C_j \text{ satisfied}] \geq 1 - \left(1 - \frac{z_j}{\ell_j}\right)^{\ell_j}$$
$$\geq \left[1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right] \cdot z_j \ .$$

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

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

$f''(z) = -\frac{\ell-1}{\ell}\left[1 - \frac{z}{\ell}\right]^{\ell-2} \leq 0$ for $z \in [0,1]$. Therefore, $f$ is concave.

$$E[W]$$

$$E[W] = \sum_j w_j \Pr[C_j \text{ is satisfied}]$$

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

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

# MAXSAT: The better of two

**Theorem 37**

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

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

$$E[\max\{W_1, W_2\}]$$

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

$$
\begin{aligned}
E[\max\{W_1, W_2\}] \\
&\geq E[\tfrac{1}{2}W_1 + \tfrac{1}{2}W_2]
\end{aligned}
$$

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

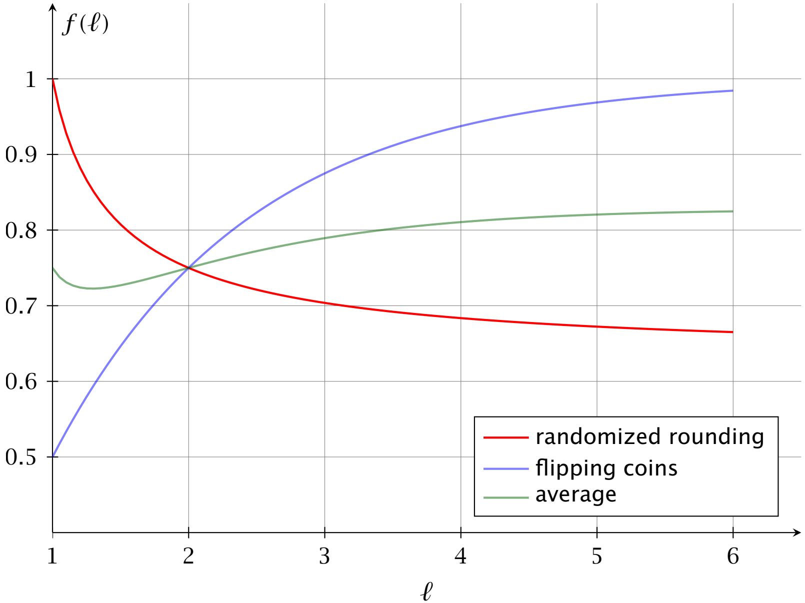
$$\begin{aligned} E[\max\{W_1, W_2\}] & \\ & \geq E[\tfrac{1}{2}W_1 + \tfrac{1}{2}W_2] \\ & \geq \frac{1}{2}\sum_j w_j z_j \left[1 - \left(1 - \frac{1}{\ell_j}\right)^{\ell_j}\right] + \frac{1}{2}\sum_j w_j \left(1 - \left(\frac{1}{2}\right)^{\ell_j}\right) \end{aligned}$$

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

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

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

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


$f(\ell)$
1
0.9
0.8
0.7
0.6
0.5
1
2
3
4
5
6
$\ell$
randomized rounding
flipping coins
average

# MAXSAT: Nonlinear Randomized Rounding

So far we used linear randomized rounding, i.e., the probability that a variable is set to $1$/true was exactly the value of the corresponding variable in the linear program.

We could define a function $f : [0,1] \to [0,1]$ and set $x_i$ to true with probability $f(y_i)$.

# MAXSAT: Nonlinear Randomized Rounding

So far we used linear randomized rounding, i.e., the probability that a variable is set to $1$/true was exactly the value of the corresponding variable in the linear program.

We could define a function $f : [0,1] \rightarrow [0,1]$ and set $x_i$ to true with probability $f(y_i)$.

# MAXSAT: Nonlinear Randomized Rounding

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

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

**Theorem 38**

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

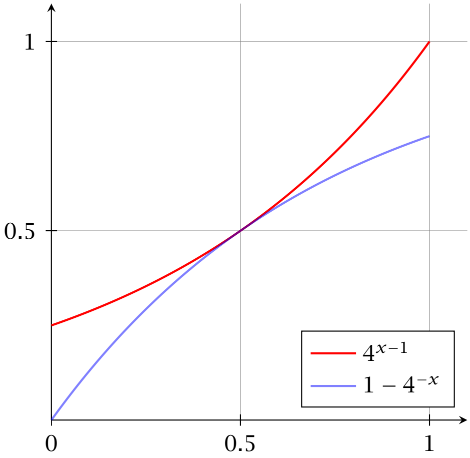
# MAXSAT: Nonlinear Randomized Rounding

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

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

**Theorem 38**

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


1
0.5
0
0.5
1
$4^{x-1}$
$1-4^{-x}$

$\Pr[C_j \text{ not satisfied}]$

$$\Pr[C_j \text{ not satisfied}] = \prod_{i \in P_j} (1 - f(y_i)) \prod_{i \in N_j} f(y_i)$$

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

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

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

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

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

$$\Pr[C_j \text{ satisfied}]$$

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

$$\Pr[C_j \text{ satisfied}] \geq 1 - 4^{-z_j}$$

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

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

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

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

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

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

Therefore,

$$E[W]$$

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

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

Therefore,

$$E[W] = \sum_j w_j \Pr[C_j \text{ satisfied}]$$

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

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

Therefore,

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

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

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

Therefore,

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

## Can we do better?

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

**Definition 39 (Integrality Gap)**

The integrality gap for an ILP is the worst-case ratio over all instances of the problem of the value of an optimal IP-solution to the value of an optimal solution to its linear programming relaxation.

Note that the integrality is less than one for maximization problems and larger than one for minimization problems (of course, equality is possible).

Note that an integrality gap only holds for one specific ILP formulation.

## Can we do better?

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

**Definition 39 (Integrality Gap)**

The integrality gap for an ILP is the worst-case ratio over all instances of the problem of the value of an optimal IP-solution to the value of an optimal solution to its linear programming relaxation.

Note that the integrality is less than one for maximization problems and larger than one for minimization problems (of course, equality is possible).

Note that an integrality gap only holds for one specific ILP formulation.

**Can we do better?**

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

**Definition 39 (Integrality Gap)**

The integrality gap for an ILP is the worst-case ratio over all instances of the problem of the value of an optimal IP-solution to the value of an optimal solution to its linear programming relaxation.

Note that the integrality is less than one for maximization problems and larger than one for minimization problems (of course, equality is possible).

Note that an integrality gap only holds for one specific ILP formulation.

**Can we do better?**

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

### Definition 39 (Integrality Gap)

The integrality gap for an ILP is the worst-case ratio over all instances of the problem of the value of an optimal IP-solution to the value of an optimal solution to its linear programming relaxation.

Note that the integrality is less than one for maximization problems and larger than one for minimization problems (of course, equality is possible).

Note that an integrality gap only holds for one specific ILP formulation.

**Can we do better?**

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

**Definition 39 (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 40

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

$$\begin{array}{llrcl} \max & & \sum_j w_j z_j & & \\ \text{s.t.} & \forall j & \sum_{i\in P_j} y_i + \sum_{i\in N_j}(1-y_i) & \geq & z_j \\ & \forall i & y_i & \in & \{0,1\} \\ & \forall j & z_j & \leq & 1 \end{array}$$

Consider: $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_2)$

- any solution can satisfy at most 3 clauses
- we can set $y_1 = y_2 = 1/2$ in the LP; this allows to set $z_1 = z_2 = z_3 = z_4 = 1$
- hence, the LP has value 4.

## Lemma 40

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

$$\begin{array}{llrcl} \max & & \sum_j w_j z_j & & \\ \text{s.t.} & \forall j & \sum_{i\in P_j} y_i + \sum_{i\in N_j}(1-y_i) & \geq & z_j \\ & \forall i & y_i & \in & \{0,1\} \\ & \forall j & z_j & \leq & 1 \end{array}$$

Consider: $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_2)$

- any solution can satisfy at most 3 clauses
- we can set $y_1 = y_2 = 1/2$ in the LP; this allows to set $z_1 = z_2 = z_3 = z_4 = 1$
- hence, the LP has value $4$.

# Repetition: Primal Dual for Set Cover

**Primal Relaxation:**

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

**Dual Formulation:**

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

# Repetition: Primal Dual for Set Cover

**Primal Relaxation:**

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

**Dual Formulation:**

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

# Repetition: Primal Dual for Set Cover

**Algorithm:**

- Start with $y = 0$ (feasible dual solution).
  Start with $x = 0$ (integral primal solution that may be infeasible).
- While $x$ not feasible
  - Identify an element $e$ that is not covered in current primal integral solution.
  - Increase dual variable $y_e$ until a dual constraint becomes tight (maybe increase by 0!).
  - If this is the constraint for set $S_j$ set $x_j = 1$ (add this set to your solution).

# Repetition: Primal Dual for Set Cover

**Algorithm:**

- Start with $y = 0$ (feasible dual solution).
  Start with $x = 0$ (integral primal solution that may be infeasible).
- While $x$ not feasible
  - Identify an element $e$ that is not covered in current primal integral solution.
  - Increase dual variable $y_e$ until a dual constraint becomes tight (maybe increase by 0!).
  - If this is the constraint for set $S_j$ set $x_j = 1$ (add this set to your solution).

# Repetition: Primal Dual for Set Cover

**Algorithm:**

- Start with $y = 0$ (feasible dual solution).
  Start with $x = 0$ (integral primal solution that may be infeasible).
- While $x$ not feasible
  - Identify an element $e$ that is not covered in current primal integral solution.
  - Increase dual variable $y_e$ until a dual constraint becomes tight (maybe increase by 0!).
  - If this is the constraint for set $S_j$ set $x_j = 1$ (add this set to your solution).

# Repetition: Primal Dual for Set Cover

**Algorithm:**

- Start with $y = 0$ (feasible dual solution).
  Start with $x = 0$ (integral primal solution that may be infeasible).
- While $x$ not feasible
  - Identify an element $e$ that is not covered in current primal integral solution.
  - Increase dual variable $y_e$ until a dual constraint becomes tight (maybe increase by $0$!).
  - If this is the constraint for set $S_j$ set $x_j = 1$ (add this set to your solution).

# Repetition: Primal Dual for Set Cover

**Algorithm:**

- Start with $y = 0$ (feasible dual solution).
  Start with $x = 0$ (integral primal solution that may be infeasible).
- While $x$ not feasible
  - Identify an element $e$ that is not covered in current primal integral solution.
  - Increase dual variable $y_e$ until a dual constraint becomes tight (maybe increase by $0$!).
  - If this is the constraint for set $S_j$ set $x_j = 1$ (add this set to your solution).

# Repetition: Primal Dual for Set Cover

**Analysis:**

# Repetition: Primal Dual for Set Cover

**Analysis:**

- For every set $S_j$ with $x_j = 1$ we have

$$\sum_{e \in S_j} y_e = w_j$$

# Repetition: Primal Dual for Set Cover

**Analysis:**

- For every set $S_j$ with $x_j = 1$ we have

$$\sum_{e \in S_j} y_e = w_j$$

- Hence our cost is

# Repetition: Primal Dual for Set Cover

**Analysis:**

- For every set $S_j$ with $x_j = 1$ we have

$$\sum_{e \in S_j} y_e = w_j$$

- Hence our cost is

$$\sum_j w_j$$

# Repetition: Primal Dual for Set Cover

**Analysis:**

- For every set $S_j$ with $x_j = 1$ we have

$$\sum_{e \in S_j} y_e = w_j$$

- Hence our cost is

$$\sum_j w_j = \sum_j \sum_{e \in S_j} y_e$$

# Repetition: Primal Dual for Set Cover

**Analysis:**

- For every set $S_j$ with $x_j = 1$ we have
$$\sum_{e \in S_j} y_e = w_j$$
- Hence our cost is
$$\sum_j w_j = \sum_j \sum_{e \in S_j} y_e = \sum_e |\{j : e \in S_j\}| \cdot y_e$$

# Repetition: Primal Dual for Set Cover

**Analysis:**

- For every set $S_j$ with $x_j = 1$ we have

$$\sum_{e \in S_j} y_e = w_j$$

- Hence our cost is

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

Note that the constructed pair of primal and dual solution fulfills primal slackness conditions.

Note that the constructed pair of primal and dual solution fulfills primal slackness conditions.

This means

$$x_j > 0 \Rightarrow \sum_{e \in S_j} y_e = w_j$$

Note that the constructed pair of primal and dual solution fulfills primal slackness conditions.

This means

$$x_j > 0 \Rightarrow \sum_{e \in S_j} y_e = w_j$$

If we would also fulfill dual slackness conditions

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

then the solution would be optimal!!!

We don't fulfill these constraint but we fulfill an approximate version:

We don’t fulfill these constraint but we fulfill an approximate version:

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

We don't fulfill these constraint but we fulfill an approximate version:

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

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

Suppose we have a primal/dual pair

$$\begin{array}{llll} \min & & \sum_j c_j x_j & \\ \text{s.t.} & \forall i & \sum_{j:} a_{ij} x_j & \geq & b_i \\ & \forall j & x_j & \geq & 0 \end{array}$$

$$\begin{array}{llll} \max & & \sum_i b_i y_i & \\ \text{s.t.} & \forall j & \sum_i a_{ij} y_i & \leq & c_j \\ & \forall i & y_i & \geq & 0 \end{array}$$

Suppose we have a primal/dual pair

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

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

and solutions that fulfill approximate slackness conditions:

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

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

Then

$$\sum_j c_j x_j$$

Then

$$\sum_j c_j x_j$$

primal cost

Then

right hand side of $j$-th dual constraint

$$\sum_j c_j x_j$$

primal cost

Then

$$\boxed{\sum_j c_j x_j} \leq \alpha \sum_j \left( \sum_i a_{ij} y_i \right) x_j$$

primal cost

Then

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

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

primal cost

Then

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

primal cost

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

$$\le \alpha\beta \cdot \sum_i b_i y_i$$

Then

$$
\begin{aligned}
\boxed{\sum_j c_j x_j} &\leq \alpha \sum_j \left( \sum_i a_{ij} y_i \right) x_j \\
&= \alpha \sum_i \left( \sum_j a_{ij} x_j \right) y_i \\
&\leq \alpha\beta \cdot \boxed{\sum_i b_i y_i}
\end{aligned}
$$

primal cost

dual objective

# Feedback Vertex Set for Undirected Graphs

- Given a graph $G = (V, E)$ and non-negative weights $w_v \geq 0$ for vertex $v \in V$.

# Feedback Vertex Set for Undirected Graphs

- Given a graph $G = (V, E)$ and non-negative weights $w_v \geq 0$ for vertex $v \in V$.
- Choose a minimum cost subset of vertices s.t. every cycle contains at least one vertex.

We can encode this as an instance of Set Cover

- Each vertex can be viewed as a set that contains some cycles.

We can encode this as an instance of Set Cover

- Each vertex can be viewed as a set that contains some cycles.
- However, this encoding gives a Set Cover instance of non-polynomial size.

We can encode this as an instance of Set Cover

- Each vertex can be viewed as a set that contains some cycles.
- However, this encoding gives a Set Cover instance of non-polynomial size.
- The $O(\log n)$-approximation for Set Cover does not help us to get a good solution.

Let $C$ denote the set of all cycles (where a cycle is identified by its set of vertices)

Let $\mathcal{C}$ denote the set of all cycles (where a cycle is identified by its set of vertices)

**Primal Relaxation:**

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

**Dual Formulation:**

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

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

- Start with $x = 0$ and $y = 0$

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

- Start with $x = 0$ and $y = 0$
- While there is a cycle $C$ that is not covered (does not contain a chosen vertex).

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

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

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

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

Then

$$\sum_v w_v x_v$$

Then

$$\sum_v w_v x_v = \sum_v \sum_{C:v\in C} y_C x_v$$

Then

$$\sum_v w_v x_v = \sum_v \sum_{C:v\in C} y_C x_v$$
$$= \sum_{v\in S} \sum_{C:v\in C} y_C$$

where $S$ is the set of vertices we choose.

Then

$$
\begin{aligned}
\sum_v w_v x_v &= \sum_v \sum_{C:v\in C} y_C x_v \\
&= \sum_{v\in S} \sum_{C:v\in C} y_C \\
&= \sum_C |S\cap C| \cdot y_C
\end{aligned}
$$

where $S$ is the set of vertices we choose.

Then

$$\begin{aligned}\sum_v w_v x_v &= \sum_v \sum_{C: v \in C} y_C x_v \\ &= \sum_{v \in S} \sum_{C: v \in C} y_C \\ &= \sum_C |S \cap C| \cdot y_C\end{aligned}$$

where $S$ is the set of vertices we choose.

If every cycle is short we get a good approximation ratio, but this is unrealistic.

**Algorithm 1** FeedbackVertexSet

$y \leftarrow 0$
$x \leftarrow 0$
**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.

**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 a $2\alpha$-approximation.

**Observation:**
If we always choose a cycle for which the number of vertices of degree at least 3 is at most $\alpha$ we get a $2\alpha$-approximation.

## Theorem 41

*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 $c$.

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

Here $\delta(S)$ denotes the set of edges with exactly one end-point in $S$, and $\mathcal{S} = \{S \subseteq V : s \in S, t \notin S\}$.

# Primal Dual for Shortest Path

Given a graph $G = (V, E)$ with two nodes $s, t \in V$ and edge-weights $c : E \to \mathbb{R}^+$ find a shortest path between $s$ and $t$ w.r.t. edge-weights $c$.

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

Here $\delta(S)$ denotes the set of edges with exactly one end-point in $S$, and $\mathcal{S} = \{S \subseteq V : s \in S, t \notin S\}$.

# Primal Dual for Shortest Path

**The Dual:**

$$\begin{array}{llrcl} \max & & \sum_S y_S & & \\ \text{s.t.} & \forall e \in E & \sum_{S: e \in \delta(S)} y_S & \leq & c(e) \\ & \forall S \in \mathcal{S} & y_S & \geq & 0 \end{array}$$

Here $\delta(S)$ denotes the set of edges with exactly one end-point in $S$, and $\mathcal{S} = \{S \subseteq V : s \in S, t \notin S\}$.

# Primal Dual for Shortest Path

**The Dual:**

$$
\begin{array}{llrcl}
\max & & \sum_S y_S & & \\
\text{s.t.} & \forall e \in E & \sum_{S: e \in \delta(S)} y_S & \le & c(e) \\
& \forall S \in \mathcal{S} & y_S & \ge & 0
\end{array}
$$

Here $\delta(S)$ denotes the set of edges with exactly one end-point in $S$, and $\mathcal{S} = \{S \subseteq V : s \in S, t \notin S\}$.

# Primal Dual for Shortest Path

We can interpret the value $y_S$ as the width of a moat surounding the set $S$.

Each set can have its own moat but all moats must be disjoint.

An edge cannot be shorter than all the moats that it has to cross.

# Primal Dual for Shortest Path

We can interpret the value $y_S$ as the width of a moat surounding the set $S$.

Each set can have its own moat but all moats must be disjoint.

An edge cannot be shorter than all the moats that it has to cross.

# Primal Dual for Shortest Path

We can interpret the value $y_S$ as the width of a moat surounding the set $S$.

Each set can have its own moat but all moats must be disjoint.

An edge cannot be shorter than all the moats that it has to cross.

# Primal Dual for Shortest Path

We can interpret the value $y_S$ as the width of a moat surounding the set $S$.

Each set can have its own moat but all moats must be disjoint.

An edge cannot be shorter than all the moats that it has to cross.

**Algorithm 1** PrimalDualShortestPath

```
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 42

*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 $r$ (call this component $C$) and add some edge from $\delta(C)$ to $F$.
- Since, at most one end-point of the new edge is in $C$ the edge cannot close a cycle.

## Lemma 42

*At each point in time the set $F$ forms a tree.*

**Proof:**

- In each iteration we take the current connected component from $(V, F)$ that contains $s$ (call this component $C$) and add some edge from $\delta(C)$ to $F$.
- Since, at most one end-point of the new edge is in $C$ the edge cannot close a cycle.

## Lemma 42

*At each point in time the set $F$ forms a tree.*

**Proof:**

- In each iteration we take the current connected component from $(V, F)$ that contains $s$ (call this component $C$) and add some edge from $\delta(C)$ to $F$.
- Since, at most one end-point of the new edge is in $C$ the edge cannot close a cycle.

$$\sum_{e \in P} c(e)$$

$$\sum_{e \in P} c(e) = \sum_{e \in P} \sum_{S : e \in \delta(S)} y_S$$

$$\sum_{e\in P} c(e) = \sum_{e\in P}\sum_{S:e\in\delta(S)} y_S$$

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

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

If we can show that $y_S > 0$ implies $|P \cap \delta(S)| = 1$ gives

$$\sum_{e\in P} c(e) = \sum_S y_S \le \mathrm{OPT}$$

by weak duality.

$$\sum_{e\in P} c(e) = \sum_{e\in P} \sum_{S:e\in\delta(S)} y_S$$

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

If we can show that $y_S > 0$ implies $|P \cap \delta(S)| = 1$ gives

$$\sum_{e\in P} c(e) = \sum_{S} y_S \le \mathrm{OPT}$$

by weak duality.

Hence, we find a shortest path.

If $S$ contains two edges from $P$ then there must exist a subpath $P'$ of $P$ that starts and ends with a vertex from $S$ (and all interior vertices are not in $S$).

When we increased $y_S$, $S$ was a connected component of the set of edges $F'$ that we had chosen till this point.

$F' \cup P'$ contains a cycle. Hence, also the final set of edges contains a cycle.

This is a contradiction.

If $S$ contains two edges from $P$ then there must exist a subpath $P'$ of $P$ that starts and ends with a vertex from $S$ (and all interior vertices are not in $S$).

When we increased $y_S$, $S$ was a connected component of the set of edges $F'$ that we had chosen till this point.

$F' \cup P'$ contains a cycle. Hence, also the final set of edges contains a cycle.

This is a contradiction.

If $S$ contains two edges from $P$ then there must exist a subpath $P'$ of $P$ that starts and ends with a vertex from $S$ (and all interior vertices are not in $S$).

When we increased $y_S$, $S$ was a connected component of the set of edges $F'$ that we had chosen till this point.

$F' \cup P'$ contains a cycle. Hence, also the final set of edges contains a cycle.

This is a contradiction.

If $S$ contains two edges from $P$ then there must exist a subpath $P'$ of $P$ that starts and ends with a vertex from $S$ (and all interior vertices are not in $S$).

When we increased $y_S$, $S$ was a connected component of the set of edges $F'$ that we had chosen till this point.

$F' \cup P'$ contains a cycle. Hence, also the final set of edges contains a cycle.

This is a contradiction.

If $S$ contains two edges from $P$ then there must exist a subpath $P'$ of $P$ that starts and ends with a vertex from $S$ (and all interior vertices are not in $S$).

When we increased $y_S$, $S$ was a connected component of the set of edges $F'$ that we had chosen till this point.

$F' \cup P'$ contains a cycle. Hence, also the final set of edges contains a cycle.

This is a contradiction.

**Steiner Forest Problem:**

Given a graph $G = (V, E)$, together with source-target pairs $s_i, t_i, i = 1, \ldots, k$, 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 $i \in \{1, \ldots, k\}$ there is a path between $s_i$ and $t_i$ only using edges in $F$.

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

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

**Steiner Forest Problem:**

Given a graph $G = (V, E)$, together with source-target pairs $s_i, t_i, i = 1, \dots, k$, and a cost function $c : E \to \mathbb{R}^+$ on the edges. Find a subset $F \subseteq E$ of the edges such that for every $i \in \{1, \dots, k\}$ there is a path between $s_i$ and $t_i$ only using edges in $F$.

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

Here $\mathcal{S}_i$ contains all sets $S$ such that $s_i \in S$ and $t_i \notin S$.

**Steiner Forest Problem:**

Given a graph $G = (V, E)$, together with source-target pairs $s_i, t_i, i = 1, \dots, k$, and a cost function $c : E \to \mathbb{R}^+$ on the edges. Find a subset $F \subseteq E$ of the edges such that for every $i \in \{1, \dots, k\}$ there is a path between $s_i$ and $t_i$ only using edges in $F$.

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

Here $\mathcal{S}_i$ contains all sets $S$ such that $s_i \in S$ and $t_i \notin S$.

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

The difference to the dual of the shortest path problem is that we have many more variables (sets for which we can generate a moat of non-zero width).

**Algorithm 1** FirstTry

```
y ← 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'}
return ⋃_i P_i
```

$y \leftarrow 0$
$F \leftarrow \emptyset$
**while** not all $s_i$-$t_i$ pairs connected in $F$ **do**
Let $C$ be some connected component of $(V,F)$ such that $|C \cap \{s_i, t_i\}| = 1$ for some $i$.
Increase $y_C$ until there is an edge $e' \in \delta(C)$ s.t. $\sum_{S \in S_i : e' \in \delta(S)} y_S = c_{e'}$
$F \leftarrow F \cup \{e'\}$
**return** $\bigcup_i P_i$

$$\sum_{e \in F} c(e)$$

$$\sum_{e \in F} c(e) = \sum_{e \in F} \sum_{S : e \in \delta(S)} y_S$$

$$\sum_{e\in F} c(e) = \sum_{e\in F} \sum_{S:e\in\delta(S)} y_S = \sum_S |\delta(S)\cap F| \cdot y_S \enspace .$$

$$\sum_{e\in F} c(e) = \sum_{e\in F} \sum_{S:e\in\delta(S)} y_S = \sum_{S} |\delta(S)\cap F| \cdot y_S \enspace .$$

$$\sum_{e\in F} c(e) = \sum_{e\in F} \sum_{S:e\in\delta(S)} y_S = \sum_{S} |\delta(S)\cap F| \cdot y_S \enspace .$$

If we show that $y_S > 0$ implies that $|\delta(S)\cap F| \le \alpha$ we are in good shape.

However, this is not true:

- Take a complete graph on $k+1$ vertices $v_0, v_1, \dots, v_k$.

$$\sum_{e\in F} c(e) = \sum_{e\in F} \sum_{S:e\in\delta(S)} y_S = \sum_S |\delta(S)\cap F| \cdot y_S \enspace .$$

If we show that $y_S > 0$ implies that $|\delta(S)\cap F| \le \alpha$ we are in good shape.

However, this is not true:

- Take a complete graph on $k+1$ vertices $v_0, v_1, \dots, v_k$.
- The $i$-th pair is $v_0$-$v_i$.

$$\sum_{e\in F} c(e) = \sum_{e\in F}\sum_{S:e\in\delta(S)} y_S = \sum_S |\delta(S)\cap F|\cdot y_S \enspace .$$

If we show that $y_S > 0$ implies that $|\delta(S)\cap F| \le \alpha$ we are in good shape.

However, this is not true:

- Take a complete graph on $k+1$ vertices $v_0, v_1, \ldots, v_k$.
- The $i$-th pair is $v_0$-$v_i$.
- The first component $C$ could be $\{v_0\}$.

$$\sum_{e\in F} c(e) = \sum_{e\in F}\sum_{S:e\in\delta(S)} y_S = \sum_S |\delta(S)\cap F|\cdot y_S\ .$$

If we show that $y_S > 0$ implies that $|\delta(S)\cap F| \leq \alpha$ we are in good shape.

However, this is not true:

- Take a complete graph on $k+1$ vertices $v_0, v_1, \dots, v_k$.
- 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 $0$.

$$\sum_{e\in F} c(e) = \sum_{e\in F} \sum_{S:e\in\delta(S)} y_S = \sum_S |\delta(S)\cap F| \cdot y_S \ .$$

If we show that $y_S > 0$ implies that $|\delta(S)\cap F| \leq \alpha$ we are in good shape.

However, this is not true:

- Take a complete graph on $k+1$ vertices $v_0, v_1, \ldots, v_k$.
- The $i$-th pair is $v_0$-$v_i$.
- The first component $C$ could be $\{v_0\}$.
- We only set $y_{\{v_0\}} = 1$. All other dual variables stay $0$.
- The final set $F$ contains all edges $\{v_0, v_i\}$, $i = 1, \ldots, k$.

$$\sum_{e\in F} c(e) = \sum_{e\in F} \sum_{S:e\in\delta(S)} y_S = \sum_S |\delta(S)\cap F| \cdot y_S \enspace .$$

If we show that $y_S > 0$ implies that $|\delta(S)\cap F| \le \alpha$ we are in good shape.

However, this is not true:

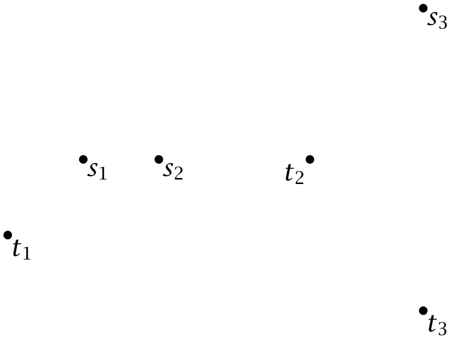
- Take a complete graph on $k+1$ vertices $v_0, v_1, \ldots, v_k$.
- The $i$-th pair is $v_0$-$v_i$.
- The first component $C$ could be $\{v_0\}$.
- We only set $y_{\{v_0\}} = 1$. All other dual variables stay $0$.
- The final set $F$ contains all edges $\{v_0, v_i\}$, $i = 1, \ldots, k$.
- $y_{\{v_0\}} > 0$ but $|\delta(\{v_0\}) \cap F| = k$.

**Algorithm 1** SecondTry

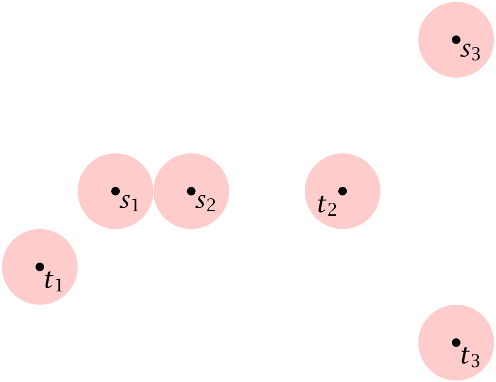
$y \leftarrow 0$; $F \leftarrow \emptyset$; $\ell \leftarrow 0$
**while** not all $s_i$-$t_i$ pairs connected in $F$ **do**
    $\ell \leftarrow \ell + 1$
    Let $\mathcal{C}$ be set of all connected components $C$ of $(V, F)$ such that $|C \cap \{s_i, t_i\}| = 1$ for some $i$.
    Increase $y_C$ for all $C \in \mathcal{C}$ uniformly until for some edge $e_\ell \in \delta(C')$, $C' \in \mathcal{C}$ s.t. $\sum_{S: e_\ell \in \delta(S)} y_S = c_{e_\ell}$
    $F \leftarrow F \cup \{e_\ell\}$
$F' \leftarrow F$
**for** $k \leftarrow \ell$ downto $1$ **do** // reverse deletion
    **if** $F' - e_k$ is feasible solution **then**
        remove $e_k$ from $F'$
**return** $F'$

The reverse deletion step is not strictly necessary this way. It would also be sufficient to simply delete all unnecessary edges in any order.

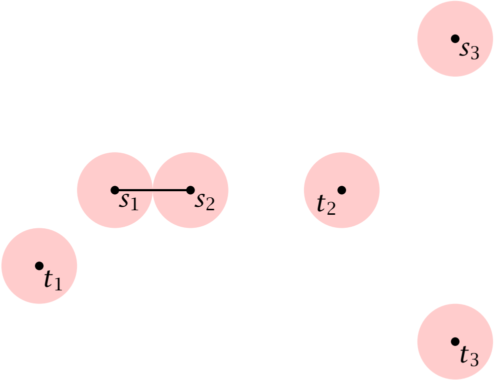
# Example



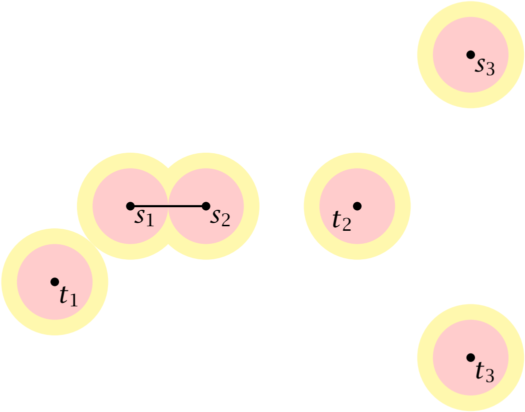
# Example



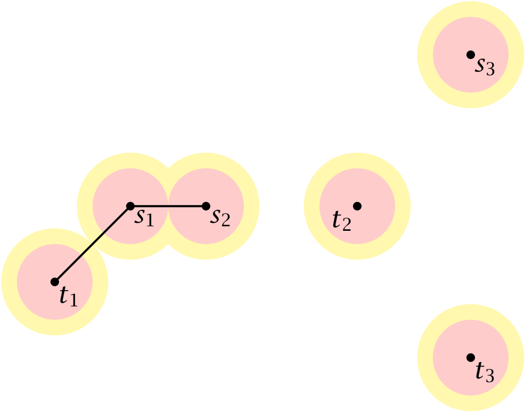
# Example



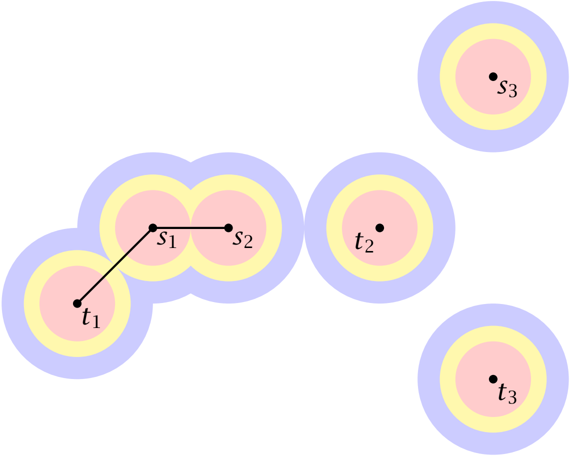
# Example



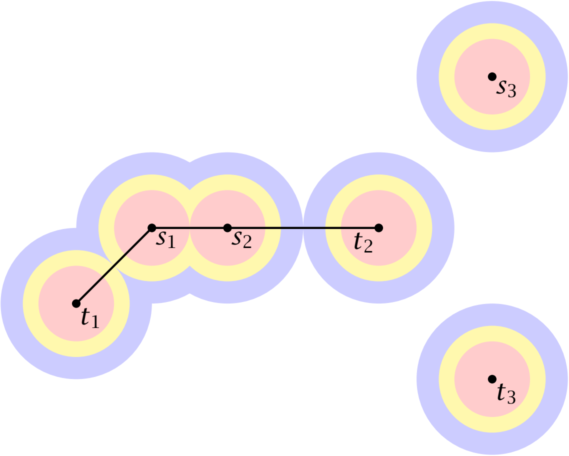
# Example



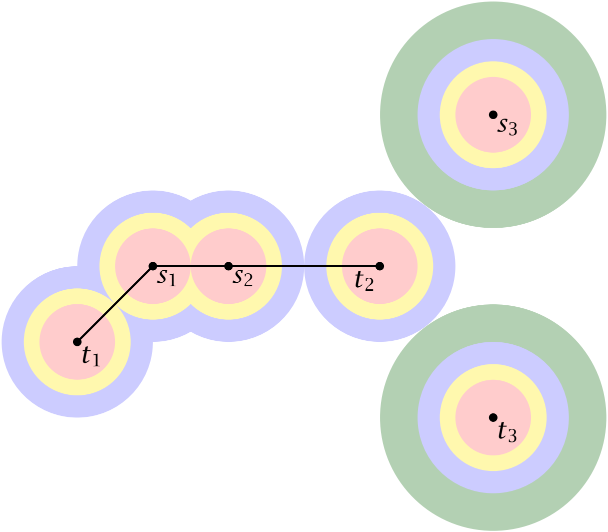
# Example



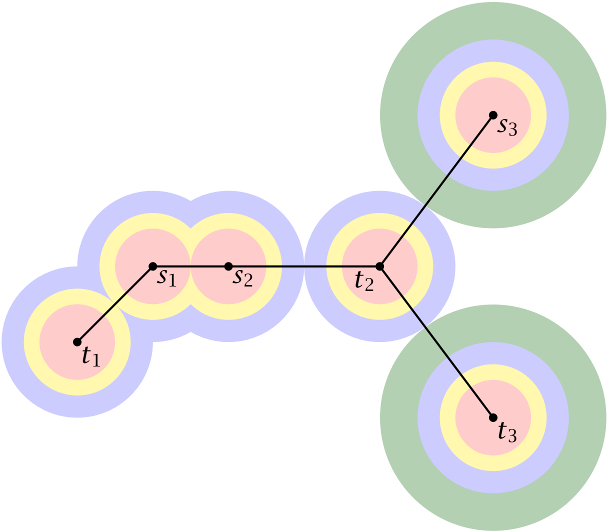
# Example



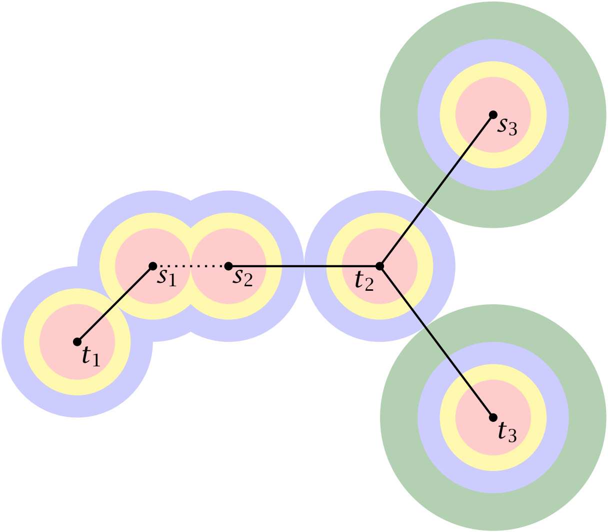
# Example



# Example



# Example



## Lemma 43

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

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

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.

$$\sum_{e\in F'} c_e = \sum_{e\in F'} \sum_{S:e\in\delta(S)} y_S = \sum_S |F' \cap \delta(S)| \cdot y_S \ .$$

We want to show that

$$\sum_S |F' \cap \delta(S)| \cdot y_S \le 2 \sum_S y_S$$

- In the $i$-th iteration the increase of the left-hand side is

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

and the increase of the right hand side is $2\epsilon|\mathcal{C}|$.

- Hence, by the previous lemma the inequality holds after the iteration if it holds in the beginning of the iteration.

$$\sum_{e \in F'} c_e = \sum_{e \in F'} \sum_{S : e \in \delta(S)} y_S = \sum_S |F' \cap \delta(S)| \cdot y_S \ .$$

We want to show that

$$\sum_S |F' \cap \delta(S)| \cdot y_S \le 2 \sum_S y_S$$

- In the $i$-th iteration the increase of the left hand side is

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

and the increase of the right hand side is $2\epsilon|\mathcal{C}|$.

- Hence, by the previous lemma the inequality holds after the iteration if it holds in the beginning of the iteration.

$$\sum_{e \in F'} c_e = \sum_{e \in F'} \sum_{S : e \in \delta(S)} y_S = \sum_S |F' \cap \delta(S)| \cdot y_S \ .$$

We want to show that

$$\sum_S |F' \cap \delta(S)| \cdot y_S \leq 2 \sum_S y_S$$

In the $i$-th iteration the increase of the left-hand side is

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

and the increase of the right hand side is $2\epsilon|\mathcal{C}|$.

Hence, by the previous lemma the inequality holds after the iteration if it holds in the beginning of the iteration.

$$\sum_{e\in F'} c_e = \sum_{e\in F'} \sum_{S:e\in\delta(S)} y_S = \sum_S |F' \cap \delta(S)| \cdot y_S \ .$$

We want to show that

$$\sum_S |F' \cap \delta(S)| \cdot y_S \le 2 \sum_S y_S$$

- In the $i$-th iteration the increase of the left-hand side is

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

  and the increase of the right hand side is $2\epsilon|C|$.
- Hence, by the previous lemma the inequality holds after the iteration if it holds in the beginning of the iteration.

$$\sum_{e\in F'} c_e = \sum_{e\in F'} \sum_{S:e\in\delta(S)} y_S = \sum_S |F' \cap \delta(S)| \cdot y_S \enspace .$$

We want to show that

$$\sum_S |F' \cap \delta(S)| \cdot y_S \le 2 \sum_S y_S$$

- In the $i$-th iteration the increase of the left-hand side is

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

  and the increase of the right hand side is $2\epsilon|C|$.
- Hence, by the previous lemma the inequality holds after the iteration if it holds in the beginning of the iteration.

## Lemma 44

*For any set of connected components $\mathcal{C}$ in any iteration of the algorithm*

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

**Proof:**

- At any point during the algorithm the set of edges forms a forest (why?).
- Fix iteration $i$. $e_i$ is the set we add to $F$. Let $F_i$ be the set of edges in $F$ at the beginning of the iteration.
- Let $H = F' - F_i$.
- All edges in $H$ are necessary for the solution.

## Lemma 44

*For any set of connected components $\mathcal{C}$ in any iteration of the algorithm*

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

**Proof:**

- At any point during the algorithm the set of edges forms a forest (why?).
- Fix iteration $i$. $e_i$ is the set we add to $F$. Let $F_i$ be the set of edges in $F$ at the beginning of the iteration.
- Let $H = F' - F_i$.
- All edges in $H$ are necessary for the solution.

## Lemma 44

*For any set of connected components $\mathcal{C}$ in any iteration of the algorithm*

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

**Proof:**

- At any point during the algorithm the set of edges forms a forest (why?).
- Fix iteration $i$. $e_i$ is the set we add to $F$. Let $F_i$ be the set of edges in $F$ at the beginning of the iteration.
- Let $H = F' - F_i$.
- All edges in $H$ are necessary for the solution.

## Lemma 44

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

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

**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.

## Lemma 44

*For any set of connected components $\mathcal{C}$ in any iteration of the algorithm*

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

**Proof:**

- At any point during the algorithm the set of edges forms a forest (why?).
- Fix iteration $i$. $e_i$ is the set we add to $F$. Let $F_i$ be the set of edges in $F$ at the beginning of the iteration.
- Let $H = F' - F_i$.
- All edges in $H$ are necessary for the solution.

- Contract all edges in $F_i$ into single vertices $V'$.
- We can consider the forest $H$ on the set of vertices $V'$.
- Let $\deg(v)$ be the degree of a vertex $v \in V'$ within this forest.
- Color a vertex $v \in V'$ red if it corresponds to a component from $C$ (an active component). Otw. color it blue. (Let $B$ the set of blue vertices (with non-zero degree) and $R$ the set of red vertices)
- We have

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

- Contract all edges in $F_i$ into single vertices $V'$.
- We can consider the forest $H$ on the set of vertices $V'$.
- Let $\deg(v)$ be the degree of a vertex $v \in V'$ within this forest.
- Color a vertex $v \in V'$ red if it corresponds to a component from $C$ (an active component). Otw. color it blue. (Let $B$ the set of blue vertices (with non-zero degree) and $R$ the set of red vertices)
- We have

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

- Contract all edges in $F_i$ into single vertices $V'$.
- We can consider the forest $H$ on the set of vertices $V'$.
- Let $\deg(v)$ be the degree of a vertex $v \in V'$ within this forest.
- Color a vertex $v \in V'$ red if it corresponds to a component from $C$ (an active component). Otw. color it blue. (Let $B$ the set of blue vertices (with non-zero degree) and $R$ the set of red vertices)
- We have

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

- Contract all edges in $F_i$ into single vertices $V'$.
- We can consider the forest $H$ on the set of vertices $V'$.
- Let $\deg(v)$ be the degree of a vertex $v \in V'$ within this forest.
- Color a vertex $v \in V'$ red if it corresponds to a component from $C$ (an active component). Otw. color it blue. (Let $B$ the set of blue vertices (with non-zero degree) and $R$ the set of red vertices)
- We have

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

- Contract all edges in $F_i$ into single vertices $V'$.
- We can consider the forest $H$ on the set of vertices $V'$.
- Let $\deg(v)$ be the degree of a vertex $v \in V'$ within this forest.
- Color a vertex $v \in V'$ red if it corresponds to a component from $C$ (an active component). Otw. color it blue. (Let $B$ the set of blue vertices (with non-zero degree) and $R$ the set of red vertices)
- We have

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

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

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

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

$$\sum_{v \in R} \deg(v)$$

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

$$\sum_{v \in R} \deg(v) = \sum_{v \in R \cup B} \deg(v) - \sum_{v \in B} \deg(v)$$

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

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

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

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

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

$$\sum_{v\in R}\deg(v) = \sum_{v\in R\cup B}\deg(v) - \sum_{v\in B}\deg(v)$$
$$\le 2(|R| + |B|) - 2|B| = 2|R|$$

- Every blue vertex with non-zero degree must have degree at least two.

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

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

- Every blue vertex with non-zero degree must have degree at least two.
  - Suppose not. The single edge connecting $b \in B$ comes from $H$, and, hence, is necessary.

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

$$\sum_{v \in R} \deg(v) = \sum_{v \in R \cup B} \deg(v) - \sum_{v \in B} \deg(v) \\ \le 2(|R| + |B|) - 2|B| = 2|R|$$

- Every blue vertex with non-zero degree must have degree at least two.
  - Suppose not. The single edge connecting $b \in B$ comes from $H$, and, hence, is necessary.
  - But this means that the cluster corresponding to $b$ must separate a source-target pair.

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

$$\sum_{v \in R} \deg(v) = \sum_{v \in R \cup B} \deg(v) - \sum_{v \in B} \deg(v)$$
$$\le 2(|R| + |B|) - 2|B| = 2|R|$$

- Every blue vertex with non-zero degree must have degree at least two.
  - Suppose not. The single edge connecting $b \in B$ comes from $H$, and, hence, is necessary.
  - But this means that the cluster corresponding to $b$ must separate a source-target pair.
  - But then it must be a red node.

# 20 Cuts & Metrics

## Shortest Path

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

$\mathcal{S}$ is the set of subsets that separate $s$ from $t$.

## The Dual:

$$\begin{array}{llrcl} \max & & \sum_S y_S & & \\ \text{s.t.} & \forall e \in E & \sum_{S:e\in\delta(S)} y_S & \leq & c(e) \\ & \forall S \in \mathcal{S} & y_S & \geq & 0 \end{array}$$

The Separation Problem for the Shortest Path LP is the Minimum Cut Problem.

# 20 Cuts & Metrics

## Shortest Path

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

$\mathcal{S}$ is the set of subsets that separate $s$ from $t$.

## The Dual:

$$\begin{array}{llrcl} \max & & \sum_S y_S & & \\ \text{s.t.} & \forall e \in E & \sum_{S:e\in\delta(S)} y_S & \leq & c(e) \\ & \forall S \in \mathcal{S} & y_S & \geq & 0 \end{array}$$

The Separation Problem for the Shortest Path LP is the Minimum Cut Problem.

# 20 Cuts & Metrics

**Shortest Path**

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

$\mathcal{S}$ is the set of subsets that separate $s$ from $t$.

**The Dual:**

$$\begin{array}{llrcl} \max & & \sum_S y_S & & \\ \text{s.t.} & \forall e \in E & \sum_{S:e\in\delta(S)} y_S & \leq & c(e) \\ & \forall S \in \mathcal{S} & y_S & \geq & 0 \end{array}$$

The Separation Problem for the Shortest Path LP is the Minimum Cut Problem.

# 20 Cuts & Metrics

### Minimum Cut

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

$\mathcal{P}$ is the set of path that connect $s$ and $t$.

### The Dual:

$$
\begin{array}{llrcl}
\max & & \sum_P y_P & & \\
\text{s.t.} & \forall e \in E & \sum_{P: e \in P} y_P & \leq & c(e) \\
& \forall P \in \mathcal{P} & y_P & \geq & 0
\end{array}
$$

The Separation Problem for the Minimum Cut LP is the Shortest Path Problem.

# 20 Cuts & Metrics

## Minimum Cut

$$\begin{array}{llrcl} \min & & \sum_e c(e)x_e & & \\ \text{s.t.} & \forall P \in \mathcal{P} & \sum_{e \in P} x_e & \geq & 1 \\ & \forall e \in E & x_e & \geq & 0 \end{array}$$

$\mathcal{P}$ is the set of path that connect $s$ and $t$.

## The Dual:

$$\begin{array}{llrcl} \max & & \sum_P y_P & & \\ \text{s.t.} & \forall e \in E & \sum_{P: e \in P} y_P & \leq & c(e) \\ & \forall P \in \mathcal{P} & y_P & \geq & 0 \end{array}$$

The Separation Problem for the Minimum Cut LP is the Shortest Path Problem.

# 20 Cuts & Metrics

**Minimum Cut**

$$\begin{array}{llrcl} \min & & \sum_e c(e)x_e & & \\ \text{s.t.} & \forall P \in \mathcal{P} & \sum_{e \in P} x_e & \geq & 1 \\ & \forall e \in E & x_e & \geq & 0 \end{array}$$

$\mathcal{P}$ is the set of path that connect $s$ and $t$.

**The Dual:**

$$\begin{array}{llrcl} \max & & \sum_P y_P & & \\ \text{s.t.} & \forall e \in E & \sum_{P: e \in P} y_P & \leq & c(e) \\ & \forall P \in \mathcal{P} & y_P & \geq & 0 \end{array}$$

The Separation Problem for the Minimum Cut LP is the Shortest Path Problem.

# 20 Cuts & Metrics

**Minimum Cut**

$$\begin{array}{llrcl} \min & & \sum_e c(e)\ell_e & & \\ \text{s.t.} & \forall P \in \mathcal{P} & \sum_{e\in P} \ell_e & \geq & 1 \\ & \forall e \in E & \ell_e & \geq & 0 \end{array}$$

$\mathcal{P}$ is the set of path that connect $s$ and $t$.

**The Dual:**

$$\begin{array}{llrcl} \max & & \sum_P f_P & & \\ \text{s.t.} & \forall e \in E & \sum_{P: e\in P} f_P & \leq & c(e) \\ & \forall P \in \mathcal{P} & f_P & \geq & 0 \end{array}$$

The Separation Problem for the Minimum Cut LP is the Shortest Path Problem.

# 20 Cuts & Metrics

**Observations:**

Suppose that $\ell_e$-values are solution to Minimum Cut LP.

- We can view $\ell_e$ as defining the length of an edge.
- Define $d(u,v) = \min_{\text{path } P \text{ btw. } u \text{ and } v} \sum_{e\in P} \ell_e$ as the Shortest Path Metric induced by $\ell_e$.
- We have $d(u,v) = \ell_e$ for every edge $e = (u,v)$, as otw. we could reduce $\ell_e$ without affecting the distance between $s$ and $t$.

**Remark for bean-counters:**

$d$ is not a metric on $V$ but a semimetric as two nodes $u$ and $v$ could have distance zero.

# 20 Cuts & Metrics

**Observations:**

Suppose that $\ell_e$-values are solution to Minimum Cut LP.

- We can view $\ell_e$ as defining the length of an edge.
- Define $d(u,v) = \min_{\text{path } P \text{ btw. } u \text{ and } v} \sum_{e \in P} \ell_e$ as the Shortest Path Metric induced by $\ell_e$.
- We have $d(u,v) = \ell_e$ for every edge $e = (u,v)$, as otw. we could reduce $\ell_e$ without affecting the distance between $s$ and $t$.

**Remark for bean-counters:**

$d$ is not a metric on $V$ but a semimetric as two nodes $u$ and $v$ could have distance zero.

# 20 Cuts & Metrics

**Observations:**

Suppose that $\ell_e$-values are solution to Minimum Cut LP.

- We can view $\ell_e$ as defining the length of an edge.
- Define $d(u,v) = \min_{\text{path } P \text{ btw. } u \text{ and } v} \sum_{e \in P} \ell_e$ as the Shortest Path Metric induced by $\ell_e$.
- We have $d(u,v) = \ell_e$ for every edge $e = (u,v)$, as otw. we could reduce $\ell_e$ without affecting the distance between $s$ and $t$.

**Remark for bean-counters:**

$d$ is not a metric on $V$ but a semimetric as two nodes $u$ and $v$ could have distance zero.

# 20 Cuts & Metrics

**Observations:**

Suppose that $\ell_e$-values are solution to Minimum Cut LP.

- We can view $\ell_e$ as defining the length of an edge.
- Define $d(u,v) = \min_{\text{path } P \text{ btw. } u \text{ and } v} \sum_{e\in P} \ell_e$ as the Shortest Path Metric induced by $\ell_e$.
- We have $d(u,v) = \ell_e$ for every edge $e = (u,v)$, as otw. we could reduce $\ell_e$ without affecting the distance between $s$ and $t$.

**Remark for bean-counters:**
$d$ is not a metric on $V$ but a semimetric as two nodes $u$ and $v$ could have distance zero.

## How do we round the LP?

- Let $B(s,r)$ be the ball of radius $r$ around $s$ (w.r.t. metric $d$). Formally:

$$B = \{v \in V \mid d(s,v) \le r\}$$

- For $0 \le r < 1$, $B(s,r)$ is an $s$-$t$-cut.

**Which value of $r$ should we choose? choose randomly!!!**

Formally:
choose $r$ u.a.r. (uniformly at random) from interval $[0,1)$

## How do we round the LP?

- Let $B(s,r)$ be the ball of radius $r$ around $s$ (w.r.t. metric $d$). Formally:

$$B = \{v \in V \mid d(s,v) \le r\}$$

- For $0 \le r < 1$, $B(s,r)$ is an $s$-$t$-cut.

## Which value of $r$ should we choose? choose randomly!!!

Formally:
choose $r$ u.a.r. (uniformly at random) from interval $[0,1)$

## How do we round the LP?

- Let $B(s,r)$ be the ball of radius $r$ around $s$ (w.r.t. metric $d$). Formally:

$$B = \{v \in V \mid d(s,v) \le r\}$$

- For $0 \le r < 1$, $B(s,r)$ is an $s$-$t$-cut.

**Which value of $r$ should we choose? choose randomly!!!**

Formally:

choose $r$ u.a.r. (uniformly at random) from interval $[0,1)$

**How do we round the LP?**

- Let $B(s,r)$ be the ball of radius $r$ around $s$ (w.r.t. metric $d$). Formally:

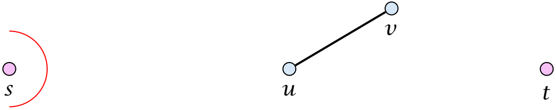
$$B = \{v \in V \mid d(s,v) \leq r\}$$

- For $0 \leq r < 1$, $B(s,r)$ is an $s$-$t$-cut.

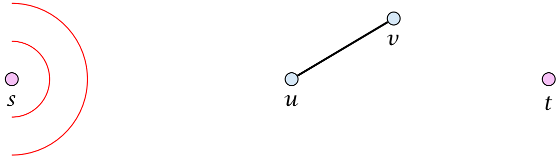
**Which value of $r$ should we choose? choose randomly!!!**

Formally:
choose $r$ u.a.r. (uniformly at random) from interval $[0,1)$

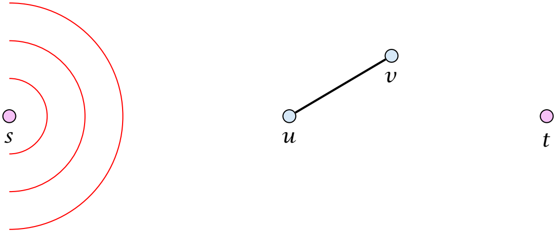
## What is the probability that an edge $(u, v)$ is in the cut?



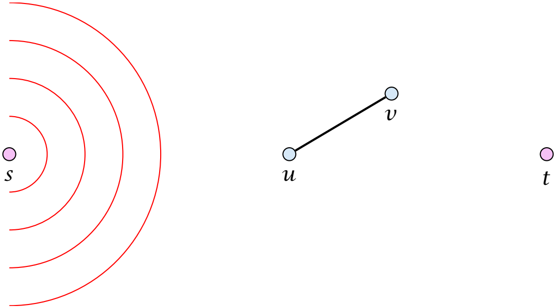
## What is the probability that an edge $(u, v)$ is in the cut?



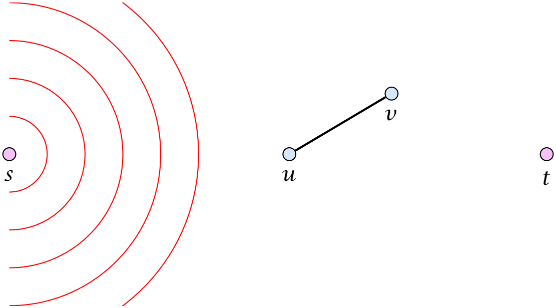
## What is the probability that an edge $(u, v)$ is in the cut?



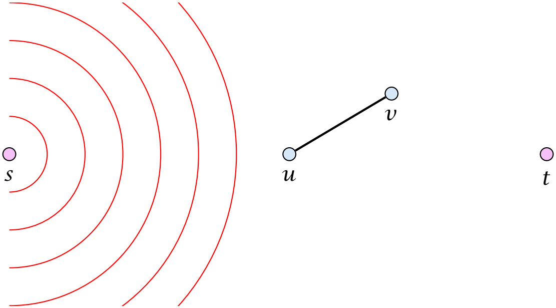
## What is the probability that an edge $(u, v)$ is in the cut?



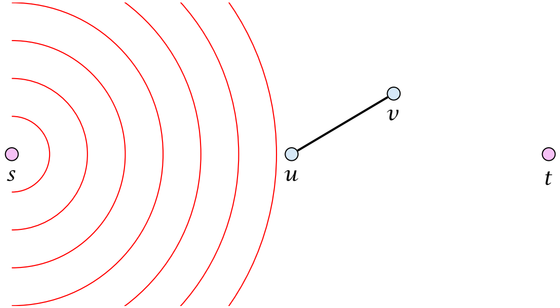
## What is the probability that an edge $(u, v)$ is in the cut?



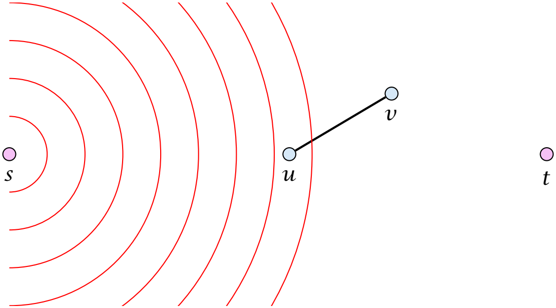
## What is the probability that an edge $(u, v)$ is in the cut?



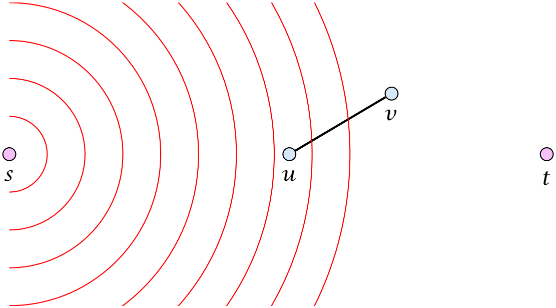
## What is the probability that an edge $(u, v)$ is in the cut?



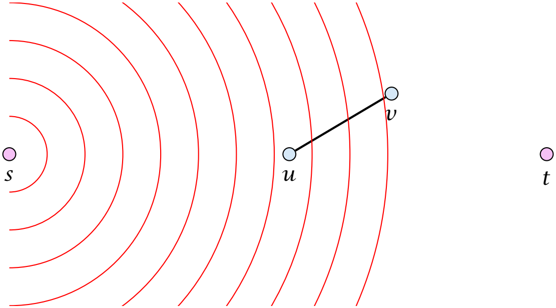
# What is the probability that an edge $(u, v)$ is in the cut?



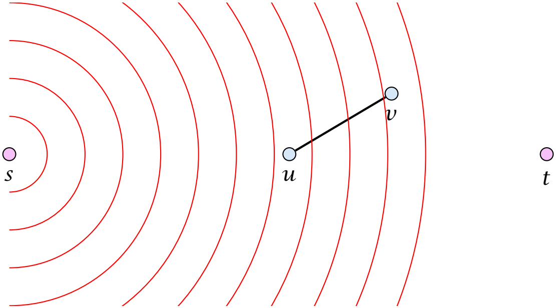
## What is the probability that an edge $(u, v)$ is in the cut?



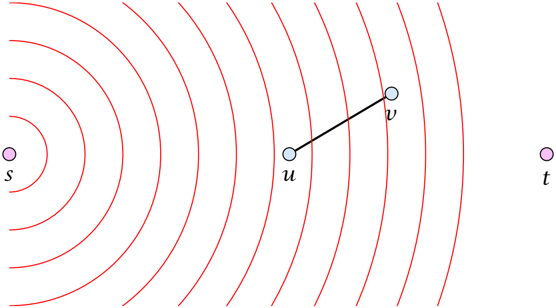
## What is the probability that an edge $(u, v)$ is in the cut?



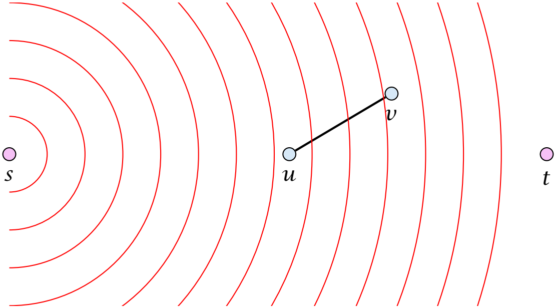
## What is the probability that an edge $(u, v)$ is in the cut?



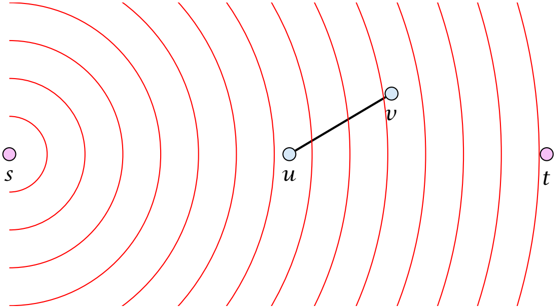
## What is the probability that an edge $(u, v)$ is in the cut?



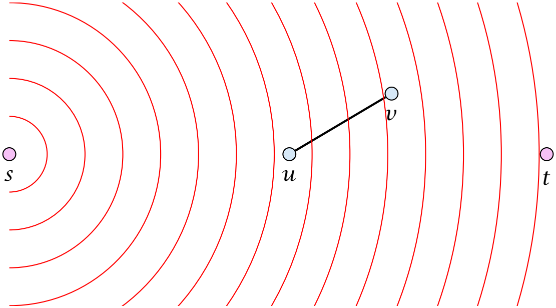
## What is the probability that an edge $(u, v)$ is in the cut?



## What is the probability that an edge $(u, v)$ is in the cut?



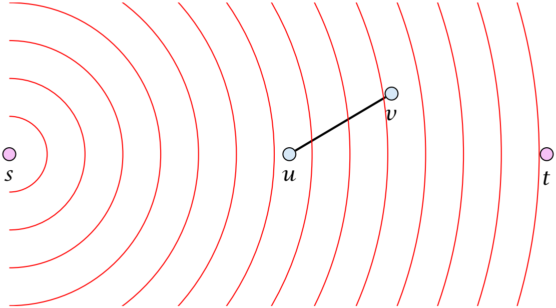
## What is the probability that an edge $(u, v)$ is in the cut?




- asssume wlog. $d(s,u) \le d(s,v)$

$$\Pr[e \text{ is cut}]$$

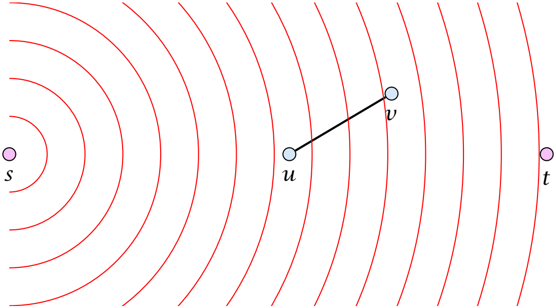
## What is the probability that an edge $(u, v)$ is in the cut?




- asssume wlog. $d(s,u) \leq d(s,v)$

$$\Pr[e \text{ is cut}] = \Pr[r \in [d(s,u), d(s,v))]$$

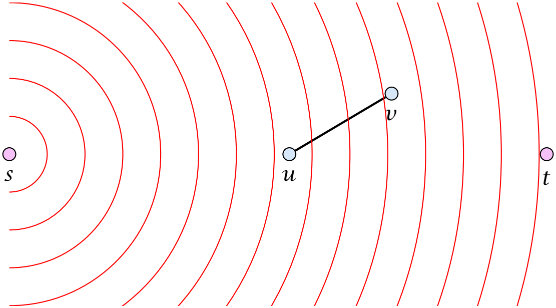
## What is the probability that an edge $(u, v)$ is in the cut?




- asssume wlog. $d(s,u) \le d(s,v)$

$$\Pr[e \text{ is cut}] = \Pr[r \in [d(s,u), d(s,v))] \le \frac{d(s,v) - d(s,u)}{1 - 0}$$

## What is the probability that an edge $(u, v)$ is in the cut?




- asssume wlog. $d(s,u) \le d(s,v)$

$$\Pr[e \text{ is cut}] = \Pr[r \in [d(s,u), d(s,v))] \le \frac{d(s,v) - d(s,u)}{1-0}$$
$$\le \ell_e$$

## What is the expected size of a cut?

$$\begin{aligned} \mathrm{E}[\text{size of cut}] &= \mathrm{E}[\textstyle\sum_e c(e)\Pr[e \text{ is cut}]] \\ &\le \textstyle\sum_e c(e)\ell_e \end{aligned}$$

On the other hand:

$$\textstyle\sum_e c(e)\ell_e \le \text{size of mincut}$$

as the $\ell_e$ are the solution to the Mincut LP *relaxation.*

Hence, our rounding gives an optimal solution.

**What is the expected size of a cut?**

$$\mathrm{E}[\text{size of cut}] = \mathrm{E}\Big[\sum_e c(e)\Pr[e \text{ is cut}]\Big]$$
$$\leq \sum_e c(e)\ell_e$$

On the other hand:

$$\sum_e c(e)\ell_e \leq \text{size of mincut}$$

as the $\ell_e$ are the solution to the Mincut LP *relaxation*.

Hence, our rounding gives an optimal solution.

**What is the expected size of a cut?**

$$\begin{aligned}\mathrm{E}[\text{size of cut}] &= \mathrm{E}[\textstyle\sum_e c(e)\Pr[e \text{ is cut}]] \\ &\le \textstyle\sum_e c(e)\ell_e\end{aligned}$$

On the other hand:

$$\textstyle\sum_e c(e)\ell_e \le \text{size of mincut}$$

as the $\ell_e$ are the solution to the Mincut LP *relaxation*.

Hence, our rounding gives an optimal solution.

**Minimum Multicut:**

Given a graph $G = (V, E)$, together with source-target pairs $s_i, t_i$, $i = 1, \ldots, k$, and a capacity function $c : E \to \mathbb{R}^+$ on the edges. Find a subset $F \subseteq E$ of the edges such that all $s_i$-$t_i$ pairs lie in different components in $G = (V, E \setminus F)$.

$$\begin{array}{llrcl} \min & & \sum_e c(e)\ell_e & & \\ \text{s.t.} & \forall P \in \mathcal{P}_i \text{ for some } i & \sum_{e \in P} \ell_e & \geq & 1 \\ & \forall e \in E & \ell_e & \in & \{0,1\} \end{array}$$

Here $\mathcal{P}_i$ contains all path $P$ between $s_i$ and $t_i$.

**Minimum Multicut:**

Given a graph $G = (V, E)$, together with source-target pairs $s_i, t_i$, $i = 1, \ldots, k$, and a capacity function $c : E \to \mathbb{R}^+$ on the edges. Find a subset $F \subseteq E$ of the edges such that all $s_i$-$t_i$ pairs lie in different components in $G = (V, E \setminus F)$.

$$
\begin{array}{llrcl}
\min & & \sum_e c(e)\ell_e & & \\
\text{s.t.} & \forall P \in \mathcal{P}_i \text{ for some } i & \sum_{e \in P} \ell_e & \geq & 1 \\
 & \forall e \in E & \ell_e & \in & \{0, 1\}
\end{array}
$$

Here $\mathcal{P}_i$ contains all path $P$ between $s_i$ and $t_i$.

**Minimum Multicut:**
Given a graph $G = (V, E)$, together with source-target pairs $s_i, t_i$, $i = 1, \dots, k$, and a capacity function $c : E \to \mathbb{R}^+$ on the edges. Find a subset $F \subseteq E$ of the edges such that all $s_i$-$t_i$ pairs lie in different components in $G = (V, E \setminus F)$.

$$\begin{array}{llrcl} \min & & \sum_e c(e)\ell_e & & \\ \text{s.t.} & \forall P \in \mathcal{P}_i \text{ for some } i & \sum_{e \in P} \ell_e & \geq & 1 \\ & \forall e \in E & \ell_e & \in & \{0,1\} \end{array}$$

Here $\mathcal{P}_i$ contains all path $P$ between $s_i$ and $t_i$.

**Re-using the analysis for the single-commodity case is difficult.**

$$\Pr[e \text{ is cut}] \leq ?$$

- If for some $R$ the balls $B(s_i, R)$ are disjoint between different sources, we get a $1/R$ approximation.
- However, this cannot be guaranteed.

**Re-using the analysis for the single-commodity case is difficult.**

$$\Pr[e \text{ is cut}] \leq \textbf{?}$$

- If for some $R$ the balls $B(s_i, R)$ are disjoint between different sources, we get a $1/R$ approximation.
- However, this cannot be guaranteed.

**Re-using the analysis for the single-commodity case is difficult.**

$$\Pr[e \text{ is cut}] \leq \textbf{?}$$

- If for some $R$ the balls $B(s_i, R)$ are disjoint between different sources, we get a $1/R$ approximation.
- However, this cannot be guaranteed.

- Assume for simplicity that all edge-length $\ell_e$ are multiples of $\delta \ll 1$.
- Replace the graph $G$ by a graph $G'$, where an edge of length $\ell_e$ is replaced by $\ell_e/\delta$ edges of length $\delta$.
- Let $B(s_i, z)$ be the ball in $G'$ that contains nodes $v$ with distance $d(s_i, v) \le z\delta$.

**Algorithm 1** RegionGrowing$(s_i, p)$

```
z ← 0
repeat
    flip a coin (Pr[heads] = p)
    z ← z + 1
until heads
return B(s_i, z)
```

- Assume for simplicity that all edge-length $\ell_e$ are multiples of $\delta \ll 1$.
- Replace the graph $G$ by a graph $G'$, where an edge of length $\ell_e$ is replaced by $\ell_e/\delta$ edges of length $\delta$.
- Let $B(s_i, z)$ be the ball in $G'$ that contains nodes $v$ with distance $d(s_i, v) \leq z\delta$.

**Algorithm 1** RegionGrowing$(s_i, p)$

```
z ← 0
repeat
    flip a coin (Pr[heads] = p)
    z ← z + 1
until heads
return B(s_i, z)
```

- Assume for simplicity that all edge-length $\ell_e$ are multiples of $\delta \ll 1$.
- Replace the graph $G$ by a graph $G'$, where an edge of length $\ell_e$ is replaced by $\ell_e/\delta$ edges of length $\delta$.
- Let $B(s_i, z)$ be the ball in $G'$ that contains nodes $v$ with distance $d(s_i, v) \le z\delta$.

**Algorithm 1** RegionGrowing($s_i, p$)

$z \leftarrow 0$
**repeat**
    flip a coin ($\Pr[\text{heads}] = p$)
    $z \leftarrow z + 1$
**until** heads
**return** $B(s_i, z)$

- Assume for simplicity that all edge-length $\ell_e$ are multiples of $\delta \ll 1$.
- Replace the graph $G$ by a graph $G'$, where an edge of length $\ell_e$ is replaced by $\ell_e/\delta$ edges of length $\delta$.
- Let $B(s_i, z)$ be the ball in $G'$ that contains nodes $v$ with distance $d(s_i, v) \leq z\delta$.

**Algorithm 1** RegionGrowing$(s_i, p)$

```
z ← 0
repeat
    flip a coin (Pr[heads] = p)
    z ← z + 1
until heads
return B(s_i, z)
```

**Algorithm 1** Multicut($G'$)

**while** $\exists s_i$-$t_i$ pair in $G'$ **do**
    $C \leftarrow$ RegionGrowing($s_i, p$)
    $G' = G' \setminus C$ // cuts edges leaving $C$
**return** $B(s_i, z)$

- probability of cutting an edge is only $p$
- a source either does not reach an edge during Region Growing; then it is not cut
- if it reaches the edge then it either cuts the edge or protects the edge from being cut by other sources
- if we choose $p = \delta$ the probability of cutting an edge is only its LP-value; our expected cost are at most OPT.

**Algorithm 1** Multicut($G'$)

**while** $\exists s_i$-$t_i$ pair in $G'$ **do**
    $C \leftarrow$ RegionGrowing$(s_i, p)$
    $G' = G' \setminus C$ // cuts edges leaving $C$
**return** $B(s_i, z)$

- probability of cutting an edge is only $p$
- a source either does not reach an edge during Region Growing; then it is not cut
- if it reaches the edge then it either cuts the edge or protects the edge from being cut by other sources
- if we choose $p = \delta$ the probability of cutting an edge is only its LP-value; our expected cost are at most OPT.

**Algorithm 1** Multicut($G'$)

```
while ∃s_i-t_i pair in G' do
    C ← RegionGrowing(s_i, p)
    G' = G' \ C // cuts edges leaving C
return B(s_i, z)
```

- probability of cutting an edge is only $p$
- a source either does not reach an edge during Region Growing; then it is not cut
- if it reaches the edge then it either cuts the edge or protects the edge from being cut by other sources
- if we choose $p = \delta$ the probability of cutting an edge is only its LP-value; our expected cost are at most OPT.

**Algorithm 1** Multicut($G'$)

```
while ∃s_i-t_i pair in G' do
    C ← RegionGrowing(s_i, p)
    G' = G' \ C // cuts edges leaving C
return B(s_i, z)
```

- probability of cutting an edge is only $\boldsymbol{p}$
- a source either does not reach an edge during Region Growing; then it is not cut
- if it reaches the edge then it either cuts the edge or protects the edge from being cut by other sources
- if we choose $p = \delta$ the probability of cutting an edge is only its LP-value; our expected cost are at most OPT.

**Algorithm 1** Multicut($G'$)

```
while ∃s_i-t_i pair in G' do
    C ← RegionGrowing(s_i, p)
    G' = G' \ C // cuts edges leaving C
return B(s_i, z)
```

- probability of cutting an edge is only $\boldsymbol{p}$
- a source either does not reach an edge during Region Growing; then it is not cut
- if it reaches the edge then it either cuts the edge or protects the edge from being cut by other sources
- if we choose $p = \delta$ the probability of cutting an edge is only its LP-value; our expected cost are at most $\mathrm{OPT}$.

**Problem:**

We may not cut all source-target pairs.

A component that we remove may contain an $s_i$-$t_i$ pair.

If we ensure that we cut before reaching radius $1/2$ we are in good shape.

**Problem:**

We may not cut all source-target pairs.

A component that we remove may contain an $s_i$-$t_i$ pair.

If we ensure that we cut before reaching radius 1/2 we are in good shape.

**Problem:**
We may not cut all source-target pairs.

A component that we remove may contain an $s_i$-$t_i$ pair.

If we ensure that we cut before reaching radius $1/2$ we are in good shape.

- choose $p = 6 \ln k \cdot \delta$
- we make $\frac{1}{2\delta}$ trials before reaching radius $1/2$.
- we say a Region Growing is not successful if it does not terminate before reaching radius $1/2$.

$$\Pr[\text{not successful}] \le (1-p)^{\frac{1}{2\delta}} = \left((1-p)^{1/p}\right)^{\frac{p}{2\delta}} \le e^{-\frac{p}{2\delta}} \le \frac{1}{k^3}$$

- Hence,

$$\Pr[\exists i \text{ that is not successful}] \le \frac{1}{k^2}$$

- choose $p = 6 \ln k \cdot \delta$
- we make $\frac{1}{2\delta}$ trials before reaching radius $1/2$.
- we say a Region Growing is not successful if it does not terminate before reaching radius $1/2$.

$$\Pr[\text{not successful}] \le (1-p)^{\frac{1}{2\delta}} = \left((1-p)^{1/p}\right)^{\frac{p}{2\delta}} \le e^{-\frac{p}{2\delta}} \le \frac{1}{k^3}$$

- Hence,

$$\Pr[\exists i \text{ that is not successful}] \le \frac{1}{k^2}$$

- choose $p = 6 \ln k \cdot \delta$
- we make $\frac{1}{2\delta}$ trials before reaching radius $1/2$.
- we say a Region Growing is not successful if it does not terminate before reaching radius $1/2$.

$$\Pr[\text{not successful}] \le (1-p)^{\frac{1}{2\delta}} = \left((1-p)^{1/p}\right)^{\frac{p}{2\delta}} \le e^{-\frac{p}{2\delta}} \le \frac{1}{k^3}$$

- Hence,

$$\Pr[\exists i \text{ that is not successful}] \le \frac{1}{k^2}$$

- choose $p = 6 \ln k \cdot \delta$
- we make $\frac{1}{2\delta}$ trials before reaching radius $1/2$.
- we say a Region Growing is not successful if it does not terminate before reaching radius $1/2$.

$$\Pr[\text{not successful}] \le (1-p)^{\frac{1}{2\delta}} = \left((1-p)^{1/p}\right)^{\frac{p}{2\delta}} \le e^{-\frac{p}{2\delta}} \le \frac{1}{k^3}$$

- Hence,

$$\Pr[\exists i \text{ that is not successful}] \le \frac{1}{k^2}$$

## What is expected cost?

$$\begin{aligned} \text{E}[\text{cutsize}] &= \Pr[\text{success}] \cdot \text{E}[\text{cutsize} \mid \text{success}] \\ &\quad + \Pr[\text{no success}] \cdot \text{E}[\text{cutsize} \mid \text{no success}] \end{aligned}$$

## What is expected cost?

$$\begin{aligned} \mathrm{E}[\text{cutsize}] &= \Pr[\text{success}] \cdot \mathrm{E}[\text{cutsize} \mid \text{success}] \\ &\quad + \Pr[\text{no success}] \cdot \mathrm{E}[\text{cutsize} \mid \text{no success}] \end{aligned}$$

$$\begin{aligned} \mathrm{E}[\text{cutsize} \mid \text{succ.}] &= \frac{\mathrm{E}[\text{cutsize}] - \Pr[\text{no succ.}] \cdot \mathrm{E}[\text{cutsize} \mid \text{no succ.}]}{\Pr[\text{success}]} \\ &\le \frac{\mathrm{E}[\text{cutsize}]}{\Pr[\text{success}]} \le \frac{1}{1-\frac{1}{k^2}} 6 \ln k \cdot \mathrm{OPT} \le 8 \ln k \cdot \mathrm{OPT} \end{aligned}$$

Note: success means all source-target pairs separated

We assume $k \ge 2$.

## What is expected cost?

$$\begin{aligned}\mathrm{E}[\text{cutsize}] &= \Pr[\text{success}] \cdot \mathrm{E}[\text{cutsize} \mid \text{success}] \\ &\quad + \Pr[\text{no success}] \cdot \mathrm{E}[\text{cutsize} \mid \text{no success}]\end{aligned}$$

$$\begin{aligned}\mathrm{E}[\text{cutsize} \mid \text{succ.}] &= \frac{\mathrm{E}[\text{cutsize}] - \Pr[\text{no succ.}] \cdot \mathrm{E}[\text{cutsize} \mid \text{no succ.}]}{\Pr[\text{success}]} \\ &\le \frac{\mathrm{E}[\text{cutsize}]}{\Pr[\text{success}]} \le \frac{1}{1 - \frac{1}{k^2}} 6 \ln k \cdot \mathrm{OPT} \le 8 \ln k \cdot \mathrm{OPT}\end{aligned}$$

Note: success means all source-target pairs separated

We assume $k \ge 2$.

## What is expected cost?

$$\begin{aligned}\mathrm{E}[\text{cutsize}] &= \Pr[\text{success}] \cdot \mathrm{E}[\text{cutsize} \mid \text{success}] \\ &\quad + \Pr[\text{no success}] \cdot \mathrm{E}[\text{cutsize} \mid \text{no success}]\end{aligned}$$

$$\begin{aligned}\mathrm{E}[\text{cutsize} \mid \text{succ.}] &= \frac{\mathrm{E}[\text{cutsize}] - \Pr[\text{no succ.}] \cdot \mathrm{E}[\text{cutsize} \mid \text{no succ.}]}{\Pr[\text{success}]} \\ &\le \frac{\mathrm{E}[\text{cutsize}]}{\Pr[\text{success}]} \le \frac{1}{1-\frac{1}{k^2}} 6 \ln k \cdot \mathrm{OPT} \le 8 \ln k \cdot \mathrm{OPT}\end{aligned}$$

Note: success means all source-target pairs separated

We assume $k \ge 2$.

## What is expected cost?

$$\begin{aligned} \text{E}[\text{cutsize}] &= \Pr[\text{success}] \cdot \text{E}[\text{cutsize} \mid \text{success}] \\ &\quad + \Pr[\text{no success}] \cdot \text{E}[\text{cutsize} \mid \text{no success}] \end{aligned}$$

$$\begin{aligned} \text{E}[\text{cutsize} \mid \text{succ.}] &= \frac{\text{E}[\text{cutsize}] - \Pr[\text{no succ.}] \cdot \text{E}[\text{cutsize} \mid \text{no succ.}]}{\Pr[\text{success}]} \\ &\le \frac{\text{E}[\text{cutsize}]}{\Pr[\text{success}]} \le \frac{1}{1-\frac{1}{k^2}} 6 \ln k \cdot \text{OPT} \le 8 \ln k \cdot \text{OPT} \end{aligned}$$

Note: success means all source-target pairs separated

We assume $k \ge 2$.

## What is expected cost?

$$\begin{aligned}\mathrm{E}[\mathrm{cutsize}] &= \Pr[\mathrm{success}] \cdot \mathrm{E}[\mathrm{cutsize} \mid \mathrm{success}] \\ &\quad + \Pr[\mathrm{no\ success}] \cdot \mathrm{E}[\mathrm{cutsize} \mid \mathrm{no\ success}]\end{aligned}$$

$$\begin{aligned}\mathrm{E}[\mathrm{cutsize} \mid \mathrm{succ.}] &= \frac{\mathrm{E}[\mathrm{cutsize}] - \Pr[\mathrm{no\ succ.}] \cdot \mathrm{E}[\mathrm{cutsize} \mid \mathrm{no\ succ.}]}{\Pr[\mathrm{success}]} \\ &\leq \frac{\mathrm{E}[\mathrm{cutsize}]}{\Pr[\mathrm{success}]} \leq \frac{1}{1-\frac{1}{k^2}} 6 \ln k \cdot \mathrm{OPT} \leq 8 \ln k \cdot \mathrm{OPT}\end{aligned}$$

Note: success means all source-target pairs separated

We assume $k \geq 2$.

## What is expected cost?

$$\begin{aligned}\mathrm{E}[\text{cutsize}] &= \Pr[\text{success}] \cdot \mathrm{E}[\text{cutsize} \mid \text{success}] \\ &\quad + \Pr[\text{no success}] \cdot \mathrm{E}[\text{cutsize} \mid \text{no success}]\end{aligned}$$

$$\begin{aligned}\mathrm{E}[\text{cutsize} \mid \text{succ.}] &= \frac{\mathrm{E}[\text{cutsize}] - \Pr[\text{no succ.}] \cdot \mathrm{E}[\text{cutsize} \mid \text{no succ.}]}{\Pr[\text{success}]} \\ &\le \frac{\mathrm{E}[\text{cutsize}]}{\Pr[\text{success}]} \le \frac{1}{1 - \frac{1}{k^2}} 6 \ln k \cdot \mathrm{OPT} \le 8 \ln k \cdot \mathrm{OPT}\end{aligned}$$

Note: success means all source-target pairs separated

We assume $k \ge 2$.

If we are not successful we simply perform a trivial $k$-approximation.

This only increases the expected cost by at most $\frac{1}{k^2} \cdot k\mathrm{OPT} \leq \mathrm{OPT}/k$.

Hence, our final cost is $\mathcal{O}(\ln k) \cdot \mathrm{OPT}$ in expectation.

**Definition 45 (NP)**

A language $L \in \mathrm{NP}$ if there exists a polynomial time, deterministic verifier $V$ (a Turing machine), s.t.

**[$x \in L$]** There exists a proof string $y$, $|y| = \mathrm{poly}(|x|)$, s.t. $V(x, y) =$ "accept".

**[$x \notin L$]** For any proof string $y$, $V(x, y) =$ "reject".

Note that requiring $|y| = \mathrm{poly}(|x|)$ for $x \notin L$ does not make a difference (**why?**).

## Definition 45 (NP)

A language $L \in \mathrm{NP}$ if there exists a polynomial time, deterministic verifier $V$ (a Turing machine), s.t.

**[$x \in L$]** There exists a proof string $y$, $|y| = \mathrm{poly}(|x|)$, s.t. $V(x, y) =$ "accept".

**[$x \notin L$]** For any proof string $y$, $V(x, y) =$ "reject".

Note that requiring $|y| = \mathrm{poly}(|x|)$ for $x \notin L$ does not make a difference (**why?**).

# Probabilistic Proof Verification

**Definition 46 (IP)**

In an interactive proof system a randomized polynomial-time verifier $V$ (with private coin tosses) interacts with an all powerful prover $P$ in polynomially many rounds. $L \in \mathrm{IP}$ if

**[$x \in L$]** There exists a strategy for $P$ s.t. $V$ accepts with probability $1$.

**[$x \notin L$]** Regardless of $P$'s strategy $V$ accepts with probability at most $1/2$.

# Probabilistic Checkable Proofs

**Definition 47 (PCP)**

A language $L \in \mathrm{PCP}_{c(n),s(n)}(r(n), q(n))$ if there exists a polynomial time, non-adaptive, randomized verifier $V$ (an Oracle Turing Machine), s.t.

**[$x \in L$]** There exists a proof string $y$, s.t. $V^{\pi_y}(x) =$ "accept" with proability $\geq c(n)$.

**[$x \notin L$]** For any proof string $y$, $V^{\pi_y}(x) =$ "accept" with probability $\leq s(n)$.

The verifier uses at most $r(n)$ random bits and makes at most $q(n)$ oracle queries.

# Probabilistic Checkable Proofs

An Oracle Turing Machine $M$ is a Turing machine that has access to an oracle.

Such an oracle allows $M$ to solve some problem in a single step.

For example having access to a TSP-oracle $\pi_{TSP}$ would allow $M$ to write a TSP-instance $x$ on a special oracle tape and obtain the answer (yes or no) in a single step.

For such TMs one looks in addition to running time also at query complexity, i.e., how often the machine queries the oracle.

For a proof string $y$, $\pi_y$ is an oracle that upon given an index $i$ returns the $i$-th character $y_i$ of $y$.

$c(n)$ is called the completeness. If not specified otw. $c(n) = 1$. Probability of accepting a correct proof.

$s(n) < c(n)$ is called the soundness. If not specified otw. $s(n) = 1/2$. Probability of accepting a wrong proof.

$r(n)$ is called the randomness complexity, i.e., how many random bits the (randomized) verifier uses.

$q(n)$ is the query complexity of the verifier.

$$\mathrm{IP} \subseteq \mathrm{PCP}_{1,1/2}(\mathrm{poly}(n), \mathrm{poly}(n))$$

We can view non-adadpative $\mathrm{PCP}_{1,1/2}(\mathrm{poly}(n), \mathrm{poly}(n))$ as the version of IP in which the prover has written down his answers to all possible queries (beforehand).

This makes it harder for the prover to cheat.

The non-cheating prover does not loose power.

**Note that the above is not a proof!**

- $\mathrm{PCP}(0,0) = \mathrm{P}$
- $\mathrm{PCP}(\mathcal{O}(\log n), 0) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\log n)) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\mathrm{poly}(n)), 0) = \mathrm{coRP}$
  randomized polynomial time with one sided error (positive probability of accepting a false statement)
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(1)) = \mathrm{NP}$ (the PCP theorem)

- $\mathrm{PCP}(0, 0) = \mathrm{P}$
- $\mathrm{PCP}(\mathcal{O}(\log n), 0) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\log n)) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\mathrm{poly}(n)), 0) = \mathrm{coRP}$
  randomized polynomial time with one sided error (positive probability of accepting a false statement)
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(1)) = \mathrm{NP}$ (the PCP theorem)

- $\mathrm{PCP}(0,0) = \mathrm{P}$
- $\mathrm{PCP}(\mathcal{O}(\log n), 0) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\log n)) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\mathrm{poly}(n)), 0) = \mathrm{coRP}$
  randomized polynomial time with one sided error (positive probability of accepting a false statement)
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(1)) = \mathrm{NP}$ (the PCP theorem)

- $\mathrm{PCP}(0,0) = \mathrm{P}$
- $\mathrm{PCP}(\mathcal{O}(\log n), 0) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\log n)) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\mathrm{poly}(n)), 0) = \mathrm{coRP}$
  randomized polynomial time with one sided error (positive probability of accepting a false statement)
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(1)) = \mathrm{NP}$ (the PCP theorem)

- $\mathrm{PCP}(0,0) = \mathrm{P}$
- $\mathrm{PCP}(\mathcal{O}(\log n), 0) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\log n)) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\mathrm{poly}(n)), 0) = \mathrm{coRP}$
  randomized polynomial time with one sided error (positive probability of accepting a false statement)
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(1)) = \mathrm{NP}$ (the PCP theorem)

- $\mathrm{PCP}(0,0) = \mathrm{P}$
- $\mathrm{PCP}(\mathcal{O}(\log n), 0) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\log n)) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\mathrm{poly}(n)), 0) = \mathrm{coRP}$
  randomized polynomial time with one sided error (positive probability of accepting a false statement)
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(1)) = \mathrm{NP}$ (the PCP theorem)

- $\mathrm{PCP}(0,0) = \mathrm{P}$
- $\mathrm{PCP}(\mathcal{O}(\log n), 0) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\log n)) = \mathrm{P}$
- $\mathrm{PCP}(0, \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(\mathrm{poly}(n))) = \mathrm{NP}$
- $\mathrm{PCP}(\mathcal{O}(\mathrm{poly}(n)), 0) = \mathrm{coRP}$
  randomized polynomial time with one sided error (positive probability of accepting a false statement)
- $\mathrm{PCP}(\mathcal{O}(\log n), \mathcal{O}(1)) = \mathrm{NP}$ (the PCP theorem)

# NP $\subseteq$ PCP(poly($n$), 1)

PCP(poly($n$), 1) means that we have a potentially exponentially long proof but we only read a constant number of bits from the proof.

The idea is to encode an NP-witness/proof (e.g. a satisfying assignment (say $n$ bits)) by a code whose code-words have $2^n$ bits.

A wrong proof is either

- a code-word whose pre-image does not correspond to a satisfying assignment
- or, a sequence of bits that does not correspond to a code-word

We can detect both cases by querying a few positions.

# $\text{NP} \subseteq \text{PCP}(\text{poly}(n), 1)$

$\text{PCP}(\text{poly}(n), 1)$ means that we have a potentially exponentially long proof but we only read a constant number of bits from the proof.

The idea is to encode an NP-witness/proof (e.g. a satisfying assignment (say $n$ bits)) by a code whose code-words have $2^n$ bits.

A wrong proof is either

- a code-word whose pre-image does not correspond to a satisfying assignment
- or, a sequence of bits that does not correspond to a code-word

We can detect both cases by querying a few positions.

# $\text{NP} \subseteq \text{PCP}(\text{poly}(n), 1)$

$\text{PCP}(\text{poly}(n), 1)$ means that we have a potentially exponentially long proof but we only read a constant number of bits from the proof.

The idea is to encode an NP-witness/proof (e.g. a satisfying assignment (say $n$ bits)) by a code whose code-words have $2^n$ bits.

A wrong proof is either

- a code-word whose pre-image does not correspond to a satisfying assignment
- or, a sequence of bits that does not correspond to a code-word

We can detect both cases by querying a few positions.

# The Code

$u \in \{0,1\}^n$ (satisfying assignment)

**Walsh-Hadamard Code:**
$\mathrm{WH}_u : \{0,1\}^n \to \{0,1\}, x \mapsto x^T u$ (over $\mathrm{GF}(2)$)

The code-word for $u$ is $\mathrm{WH}_u$. We identify this function by a bit-vector of length $2^n$.

# The Code

**Lemma 48**
*If $u \neq u'$ then $\mathrm{WH}_u$ and $\mathrm{WH}_{u'}$ differ in at least $2^{n-1}$ bits.*

Suppose that $u - u' \neq 0$. Then

$$\mathrm{WH}_u(x) \neq \mathrm{WH}_{u'}(x) \iff (u - u')^T x \neq 0$$

This holds for $2^{n-1}$ different vectors $x$.

# The Code

**Lemma 48**
*If* $u \neq u'$ *then* $\mathrm{WH}_u$ *and* $\mathrm{WH}_{u'}$ *differ in at least* $2^{n-1}$ *bits.*

Suppose that $u - u' \neq 0$. Then

$$\mathrm{WH}_u(x) \neq \mathrm{WH}_{u'}(x) \iff (u - u')^T x \neq 0$$

This holds for $2^{n-1}$ different vectors $x$.

# The Code

Suppose we are given access to a function $f : \{0,1\}^n \rightarrow \{0,1\}$ and want to check whether it is a codeword.

Since the set of codewords is the set of all linear functions $\{0,1\}^n$ to $\{0,1\}$ we can check

$$f(x + y) = f(x) + f(y)$$

for all $2^{2n}$ pairs $x, y$. But that's not very efficient.

# The Code

Suppose we are given access to a function $f : \{0,1\}^n \to \{0,1\}$ and want to check whether it is a codeword.

Since the set of codewords is the set of all linear functions $\{0,1\}^n$ to $\{0,1\}$ we can check

$$f(x+y) = f(x) + f(y)$$

for all $2^{2n}$ pairs $x, y$. But that's not very efficient.

Can we just check a constant number of positions?

## Definition 49

Let $\rho \in [0,1]$. We say that $f, g : \{0,1\}^n \to \{0,1\}$ are $\rho$-close if

$$\Pr_{x \in \{0,1\}^n}[f(x) = g(x)] \geq \rho \ .$$

## Theorem 50

*Let $f : \{0,1\}^n \to \{0,1\}$ with*

$$\Pr_{x,y \in \{0,1\}^n}\Big[f(x) + f(y) = f(x+y)\Big] \geq \rho > \frac{1}{2} \ .$$

*Then there is a linear function $\tilde{f}$ such that $f$ and $\tilde{f}$ are $\rho$-close.*

## Definition 49

Let $\rho \in [0,1]$. We say that $f, g : \{0,1\}^n \to \{0,1\}$ are $\rho$-close if

$$\Pr_{x\in\{0,1\}^n}[f(x) = g(x)] \geq \rho \ .$$

## Theorem 50

*Let* $f : \{0,1\}^n \to \{0,1\}$ *with*

$$\Pr_{x,y\in\{0,1\}^n}\Big[f(x) + f(y) = f(x+y)\Big] \geq \rho > \frac{1}{2} \ .$$

*Then there is a linear function* $\tilde{f}$ *such that* $f$ *and* $\tilde{f}$ *are* $\rho$*-close.*

We need $\mathcal{O}(1/\delta)$ trials to be sure that $f$ is $(1-\delta)$-close to a linear function with (arbitrary) constant probability.

Suppose for $\delta < 1/4$ $f$ is $(1-\delta)$-close to some linear function $\tilde{f}$.

$\tilde{f}$ is uniquely defined by $f$, since linear functions differ on at least half their inputs.

Suppose we are given $x \in \{0,1\}^n$ and access to $f$. Can we compute $\tilde{f}(x)$ using only constant number of queries?

Suppose for $\delta < 1/4$ $f$ is $(1-\delta)$-close to some linear function $\tilde{f}$.

$\tilde{f}$ is uniquely defined by $f$, since linear functions differ on at least half their inputs.

Suppose we are given $x \in \{0,1\}^n$ and access to $f$. Can we compute $\tilde{f}(x)$ using only constant number of queries?

Suppose for $\delta < 1/4$ $f$ is $(1-\delta)$-close to some linear function $\tilde{f}$.

$\tilde{f}$ is uniquely defined by $f$, since linear functions differ on at least half their inputs.

Suppose we are given $x \in \{0,1\}^n$ and access to $f$. Can we compute $\tilde{f}(x)$ using only constant number of queries?

Suppose we are given $x \in \{0,1\}^n$ and access to $f$. Can we compute $\tilde{f}(x)$ using only constant number of queries?

1. Choose $x' \in \{0,1\}^n$ u.a.r.
2. Set $x'' := x + x'$.
3. Let $y' = f(x')$ and $y'' = f(x'')$.
4. Output $y' + y''$.

$x'$ and $x''$ are uniformly distributed (albeit dependent). With probability at least $1 - 2\delta$ we have $f(x') = \tilde{f}(x')$ and $f(x'') = \tilde{f}(x'')$.

Then we can compute $\tilde{f}(x)$.

This technique is known as local decoding of the Walsh-Hadamard code.

Suppose we are given $x \in \{0,1\}^n$ and access to $f$. Can we compute $\tilde{f}(x)$ using only constant number of queries?

1. Choose $x' \in \{0,1\}^n$ u.a.r.
2. Set $x'' := x + x'$.
3. Let $y' = f(x')$ and $y'' = f(x'')$.
4. Output $y' + y''$.

$x'$ and $x''$ are uniformly distributed (albeit dependent). With probability at least $1 - 2\delta$ we have $f(x') = \tilde{f}(x')$ and $f(x'') = \tilde{f}(x'')$.

Then we can compute $\tilde{f}(x)$.

This technique is known as local decoding of the Walsh-Hadamard code.

Suppose we are given $x \in \{0,1\}^n$ and access to $f$. Can we compute $\tilde{f}(x)$ using only constant number of queries?

1. Choose $x' \in \{0,1\}^n$ u.a.r.
2. Set $x'' := x + x'$.
3. Let $y' = f(x')$ and $y'' = f(x'')$.
4. Output $y' + y''$.

$x'$ and $x''$ are uniformly distributed (albeit dependent). With probability at least $1 - 2\delta$ we have $f(x') = \tilde{f}(x')$ and $f(x'') = \tilde{f}(x'')$.

Then we can compute $\tilde{f}(x)$.

This technique is known as local decoding of the Walsh-Hadamard code.

# $\mathrm{NP} \subseteq \mathrm{PCP}(\mathrm{poly}(n), 1)$

We show that $\mathrm{QUADEQ} \in \mathrm{PCP}(\mathrm{poly}(n), 1)$. The theorem follows since any $\mathrm{PCP}$-class is closed under polynomial time reductions.

introduce $\mathrm{QUADEQ}$...

prove NP-completeness...

Let $A$, $b$ be an instance of QUADEQ. Let $u$ be a satisfying assignment.

The correct PCP-proof will be the Walsh-Hadamard encodings of $u$ and $u \otimes u$. The verifier will accept such a proof with probability 1.

We have to make sure that we reject proofs that do not correspond to codewords for vectors of the form $u$, and $u \otimes u$.

We also have to reject proofs that correspond to codewords for vectors of the form $z$, and $z \otimes z$, where $z$ is not a satisfying assignment.

**Step 1. Linearity Test.**

The proof contains $2^n + 2^{n^2}$ bits. This is interpreted as a pair of functions $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^{n^2} \to \{0,1\}$.

We do a $0.99$-linearity test for both functions (requires a constant number of queries).

We also assume that the remaining constant number of (random) accesses only hit points where $f(x) = \tilde{f}(x)$.

Hence, our proof will only see $\tilde{f}$ and therefore we use $f$ for $\tilde{f}$, in the following (similar for $g$, $\tilde{g}$).

**Step 1. Linearity Test.**
The proof contains $2^n + 2^{n^2}$ bits. This is interpreted as a pair of functions $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^{n^2} \to \{0,1\}$.

We do a $0.99$-linearity test for both functions (requires a constant number of queries).

We also assume that the remaining constant number of (random) accesses only hit points where $f(x) = \tilde{f}(x)$.

Hence, our proof will only see $\tilde{f}$ and therefore we use $f$ for $\tilde{f}$, in the following (similar for $g$, $\tilde{g}$).

**Step 1. Linearity Test.**
The proof contains $2^n + 2^{n^2}$ bits. This is interpreted as a pair of functions $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^{n^2} \to \{0,1\}$.

We do a $0.99$-linearity test for both functions (requires a constant number of queries).

We also assume that the remaining constant number of (random) accesses only hit points where $f(x) = \tilde{f}(x)$.

Hence, our proof will only see $\tilde{f}$ and therefore we use $f$ for $\tilde{f}$, in the following (similar for $g$, $\tilde{g}$).

**Step 1. Linearity Test.**

The proof contains $2^n + 2^{n^2}$ bits. This is interpreted as a pair of functions $f : \{0,1\}^n \to \{0,1\}$ and $g : \{0,1\}^{n^2} \to \{0,1\}$.

We do a $0.99$-linearity test for both functions (requires a constant number of queries).

We also assume that the remaining constant number of (random) accesses only hit points where $f(x) = \tilde{f}(x)$.

Hence, our proof will only see $\tilde{f}$ and therefore we use $f$ for $\tilde{f}$, in the following (similar for $g$, $\tilde{g}$).

**Step 2. Verify that $g$ encodes $u \otimes u$ where $u$ is string encoded by $f$.**

$f(r) = u^T r$ and $g(z) = w^T z$ since $f, g$ are linear.

- choose $r, r'$ independently, u.a.r. from $\{0,1\}^n$
- if $f(r)f(r') \neq g(r \otimes r')$ reject
- repeat 3 times

A correct proof survives the test

$$f(r) \cdot f(r')$$

A correct proof survives the test

$$f(r) \cdot f(r') = u^T r \cdot u^T r'$$

A correct proof survives the test

$$f(r) \cdot f(r') = u^T r \cdot u^T r' = \Big( \sum_i u_i r_i \Big) \cdot \Big( \sum_j u_j r'_j \Big)$$

A correct proof survives the test

$$f(r) \cdot f(r') = u^T r \cdot u^T r' = \Big(\sum_i u_i r_i\Big) \cdot \Big(\sum_j u_j r'_j\Big)$$

$$= \sum_{ij} u_i u_j r_i r'_j$$

A correct proof survives the test

$$
\begin{aligned}
f(r) \cdot f(r') = u^T r \cdot u^T r' &= \Big(\sum_i u_i r_i\Big) \cdot \Big(\sum_j u_j r'_j\Big) \\
&= \sum_{ij} u_i u_j r_i r'_j = (u \otimes u)^T (r \otimes r')
\end{aligned}
$$

A correct proof survives the test

$$
\begin{aligned}
f(r) \cdot f(r') &= u^T r \cdot u^T r' = \Big( \sum_i u_i r_i \Big) \cdot \Big( \sum_j u_j r_j' \Big) \\
&= \sum_{ij} u_i u_j r_i r_j' = (u \otimes u)^T (r \otimes r') = g(r \otimes r')
\end{aligned}
$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let $W$ be $n \times n$-matrix with entries from $w$. Let $U$ be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let $W$ be $n \times n$-matrix with entries from $w$. Let $U$ be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r')$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let $W$ be $n \times n$-matrix with entries from $w$. Let $U$ be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r') = w^T(r \otimes r')$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let $W$ be $n \times n$-matrix with entries from $w$. Let $U$ be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r') = w^T(r \otimes r') = \sum_{ij} w_{ij} r_i r'_j$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let $W$ be $n \times n$-matrix with entries from $w$. Let $U$ be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r') = w^T(r \otimes r') = \sum_{ij} w_{ij} r_i r'_j = r^T W r'$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let $W$ be $n \times n$-matrix with entries from $w$. Let $U$ be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r') = w^T(r \otimes r') = \sum_{ij} w_{ij} r_i r'_j = r^T W r'$$

$$f(r)f(r')$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let $W$ be $n \times n$-matrix with entries from $w$. Let $U$ be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r') = w^T(r \otimes r') = \sum_{ij} w_{ij} r_i r'_j = r^T W r'$$

$$f(r) f(r') = u^T r \cdot u^T r'$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let $W$ be $n \times n$-matrix with entries from $w$. Let $U$ be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r') = w^T(r \otimes r') = \sum_{ij} w_{ij} r_i r'_j = r^T W r'$$

$$f(r) f(r') = u^T r \cdot u^T r' = r^T U r'$$

Suppose that the proof is not correct and $w \neq u \otimes u$.

Let $W$ be $n \times n$-matrix with entries from $w$. Let $U$ be matrix with $U_{ij} = u_i \cdot u_j$ (entries from $u \otimes u$).

$$g(r \otimes r') = w^T(r \otimes r') = \sum_{ij} w_{ij} r_i r'_j = r^T W r'$$

$$f(r) f(r') = u^T r \cdot u^T r' = r^T U r'$$

If $U \neq W$ then $Wr' \neq Ur'$ with probability at least 1/2. Then $r^T W r' \neq r^T U r'$ with probability at least 1/4.

**Step 3. Verify that $f$ encodes satisfying assignment.**

We need to check

$$A_k(u \otimes u) = b_k$$

where $A_k$ is the $k$-th row of the constraint matrix. But the left hand side is just $g(A_k^T)$.

We can handle this by a single query but checking all constraints would take $\mathcal{O}(m)$ steps.

We compute $rA$, where $r \in_R \{0,1\}^m$. If $u$ is not a satisfying assignment then with probability 1/2 the vector $r$ will hit an odd number of violated constraint.

In this case $rA(u \otimes u) \neq rb_k$. The left hand side is equal to $g(A^T r^T)$.

**Step 3. Verify that $f$ encodes satisfying assignment.**

We need to check

$$A_k(u \otimes u) = b_k$$

where $A_k$ is the $k$-th row of the constraint matrix. But the left hand side is just $g(A_k^T)$.

We can handle this by a single query but checking all constraints would take $\mathcal{O}(m)$ steps.

We compute $rA$, where $r \in_R \{0,1\}^m$. If $u$ is not a satisfying assignment then with probability 1/2 the vector $r$ will hit an odd number of violated constraint.

In this case $rA(u \otimes u) \neq rb_k$. The left hand side is equal to $g(A^T r^T)$.

**Step 3. Verify that $f$ encodes satisfying assignment.**

We need to check

$$A_k(u \otimes u) = b_k$$

where $A_k$ is the $k$-th row of the constraint matrix. But the left hand side is just $g(A_k^T)$.

We can handle this by a single query but checking all constraints would take $\mathcal{O}(m)$ steps.

We compute $rA$, where $r \in_R \{0,1\}^m$. If $u$ is not a satisfying assignment then with probability 1/2 the vector $r$ will hit an odd number of violated constraint.

In this case $rA(u \otimes u) \neq rb_k$. The left hand side is equal to $g(A^T r^T)$.

**Step 3. Verify that $f$ encodes satisfying assignment.**

We need to check

$$A_k(u \otimes u) = b_k$$

where $A_k$ is the $k$-th row of the constraint matrix. But the left hand side is just $g(A_k^T)$.

We can handle this by a single query but checking all constraints would take $\mathcal{O}(m)$ steps.

We compute $rA$, where $r \in_R \{0,1\}^m$. If $u$ is not a satisfying assignment then with probability 1/2 the vector $r$ will hit an odd number of violated constraint.

In this case $rA(u \otimes u) \neq rb_k$. The left hand side is equal to $g(A^T r^T)$.

## Theorem 50

*Let* $f : \{0,1\}^n \to \{0,1\}$ *with*

$$\Pr_{x,y\in\{0,1\}^n}\Big[f(x) + f(y) = f(x+y)\Big] \geq \rho > \frac{1}{2}\ .$$

*Then there is a linear function* $\tilde{f}$ *such that* $f$ *and* $\tilde{f}$ *are* $\rho$*-close.*

## Fourier Transform over GF(2)

In the following we use $\{-1, 1\}$ instead of $\{0, 1\}$. We map $b \in \{0, 1\}$ to $(-1)^b$.

This turns summation into multiplication.

The set of function $f : \{-1, 1\} \to \mathbb{R}$ form a $2^n$-dimensional Hilbert space.

## Hilbert space

- addition $(f+g)(x) = f(x) + g(x)$
- scalar multiplication $(\alpha f)(x) = \alpha f(x)$
- inner product $\langle f, g\rangle = E_{x\in\{0,1\}^n}[f(x)g(x)]$
  (bilinear, $\langle f, f\rangle \geq 0$, and $\langle f, f\rangle = 0 \Rightarrow f = 0$)
- completeness: any sequence $x_k$ of vectors for which

$$\sum_{k=1}^{\infty} \|x_k\| < \infty \text{ fulfills } \left\| L - \sum_{k=1}^{N} x_k \right\| \to 0$$

  for some vector $L$.

**standard basis**

$$e_x(y) = \begin{cases} 1 & x = y \\ 0 & \text{otw.} \end{cases}$$

Then, $f(x) = \sum_x \alpha_x e_x$ where $\alpha_x = f(x)$, this means the functions $e_x$ form a basis. This basis is orthonormal.

## fourier basis

For $\alpha \subseteq [n]$ define

$$\chi_\alpha(x) = \prod_{i \in \alpha} x_i$$

## fourier basis

For $\alpha \subseteq [n]$ define

$$\chi_\alpha(x) = \prod_{i \in \alpha} x_i$$

Note that

$$\langle \chi_\alpha, \chi_\beta \rangle$$

## fourier basis

For $\alpha \subseteq [n]$ define

$$\chi_\alpha(x) = \prod_{i \in \alpha} x_i$$

Note that

$$\langle \chi_\alpha, \chi_\beta \rangle = E_x\Big[\chi_\alpha(x)\chi_\beta(x)\Big]$$

## fourier basis

For $\alpha \subseteq [n]$ define

$$\chi_\alpha(x) = \prod_{i\in\alpha} x_i$$

Note that

$$\langle \chi_\alpha, \chi_\beta \rangle = E_x\Big[\chi_\alpha(x)\chi_\beta(x)\Big] = E_x\Big[\chi_{\alpha\triangle\beta}(x)\Big]$$

## fourier basis

For $\alpha \subseteq [n]$ define

$$\chi_\alpha(x) = \prod_{i \in \alpha} x_i$$

Note that

$$\langle \chi_\alpha, \chi_\beta \rangle = E_x\Big[\chi_\alpha(x)\chi_\beta(x)\Big] = E_x\Big[\chi_{\alpha \triangle \beta}(x)\Big] = \begin{cases} 1 & \alpha = \beta \\ 0 & \text{otw.} \end{cases}$$

## fourier basis

For $\alpha \subseteq [n]$ define

$$\chi_\alpha(x) = \prod_{i \in \alpha} x_i$$

Note that

$$\langle \chi_\alpha, \chi_\beta \rangle = E_x\Big[\chi_\alpha(x)\chi_\beta(x)\Big] = E_x\Big[\chi_{\alpha \triangle \beta}(x)\Big] = \begin{cases} 1 & \alpha = \beta \\ 0 & \text{otw.} \end{cases}$$

This means the $\chi_\alpha$'s also define an orthonormal basis. (since we have $2^n$ orthonormal vectors...)

A function $\chi_\alpha$ multiplies a set of $x_i$'s. Back in the $\mathrm{GF}(2)$-world this means summing a set of $z_i$'s where $x_i = (-1)^{z_i}$.

This means the function $\chi_\alpha$ correspond to linear functions in the $\mathrm{GF}(2)$ world.

We can write any function $f : \{-1, 1\}^n \to \mathbb{R}$ as

$$f = \sum_{\alpha} \hat{f}_\alpha \chi_\alpha$$

We call $\hat{f}_\alpha$ the $\alpha^{th}$ Fourier coefficient.

## Lemma 51

1. $\langle f, g\rangle = \sum_\alpha f_\alpha g_\alpha$
2. $\langle f, f\rangle = \sum_\alpha f_\alpha^2$

Note that for Boolean functions $f : \{-1, 1\}^n \to \{-1, 1\}$, $\langle f, f\rangle = 1$.

# Linearity Test

**GF(2)**

We want to show that if $\Pr_{x,y}[f(x) + f(y) = f(x + y)]$ is large than $f$ has a large agreement with a linear function.

# Linearity Test

**GF(2)**
We want to show that if $\Pr_{x,y}[f(x) + f(y) = f(x + y)]$ is large than $f$ has a large agreement with a linear function.

**Hilbert space** (we prove)
Suppose that $f : \{+1, -1\}^n \to \{-1, 1\}$ satisfies $\Pr_{x,y}[f(x)f(y) = f(xy)] \geq \frac{1}{2} + \epsilon$. Then there is some $\alpha \subseteq [n]$, s.t. $\hat{f}_\alpha \geq 2\epsilon$.

# Linearity Test

**GF(2)**
We want to show that if $\Pr_{x,y}[f(x) + f(y) = f(x+y)]$ is large than $f$ has a large agreement with a linear function.

**Hilbert space** (we prove)
Suppose that $f : \{+1,-1\}^n \to \{-1,1\}$ satisfies $\Pr_{x,y}[f(x)f(y) = f(xy)] \geq \frac{1}{2} + \epsilon$. Then there is some $\alpha \subseteq [n]$, s.t. $\hat{f}_\alpha \geq 2\epsilon$.

For Boolean functions $\langle f, g \rangle$ is the fraction of inputs on which $f, g$ agree **minus** the fraction of inputs on which they disagree.

# Linearity Test

**GF(2)**
We want to show that if $\Pr_{x,y}[f(x)+f(y)=f(x+y)]$ is large than $f$ has a large agreement with a linear function.

**Hilbert space** (we prove)
Suppose that $f:\{+1,-1\}^n \rightarrow \{-1,1\}$ satisfies $\Pr_{x,y}[f(x)f(y)=f(xy)] \geq \frac{1}{2}+\epsilon$. Then there is some $\alpha \subseteq [n]$, s.t. $\hat{f}_\alpha \geq 2\epsilon$.

For Boolean functions $\langle f,g \rangle$ is the fraction of inputs on which $f,g$ agree **minus** the fraction of inputs on which they disagree.

$$2\epsilon \leq \hat{f}_\alpha$$

# Linearity Test

**GF(2)**
We want to show that if $\Pr_{x,y}[f(x) + f(y) = f(x+y)]$ is large than $f$ has a large agreement with a linear function.

**Hilbert space** (we prove)
Suppose that $f : \{+1,-1\}^n \to \{-1,1\}$ satisfies $\Pr_{x,y}[f(x)f(y) = f(xy)] \geq \frac{1}{2} + \epsilon$. Then there is some $\alpha \subseteq [n]$, s.t. $\hat{f}_\alpha \geq 2\epsilon$.

For Boolean functions $\langle f, g\rangle$ is the fraction of inputs on which $f, g$ agree **minus** the fraction of inputs on which they disagree.

$$2\epsilon \leq \hat{f}_\alpha = \langle f, \chi_\alpha \rangle$$

# Linearity Test

**GF(2)**
We want to show that if $\Pr_{x,y}[f(x) + f(y) = f(x+y)]$ is large than $f$ has a large agreement with a linear function.

**Hilbert space** (we prove)
Suppose that $f : \{+1,-1\}^n \to \{-1,1\}$ satisfies $\Pr_{x,y}[f(x)f(y) = f(xy)] \geq \frac{1}{2} + \epsilon$. Then there is some $\alpha \subseteq [n]$, s.t. $\hat{f}_\alpha \geq 2\epsilon$.

For Boolean functions $\langle f, g\rangle$ is the fraction of inputs on which $f, g$ agree **minus** the fraction of inputs on which they disagree.

$$2\epsilon \leq \hat{f}_\alpha = \langle f, \chi_\alpha\rangle = \text{agree} - \text{disagree}$$

# Linearity Test

**GF(2)**
We want to show that if $\Pr_{x,y}[f(x) + f(y) = f(x+y)]$ is large than $f$ has a large agreement with a linear function.

**Hilbert space** (we prove)
Suppose that $f : \{+1,-1\}^n \to \{-1,1\}$ satisfies $\Pr_{x,y}[f(x)f(y) = f(xy)] \geq \frac{1}{2} + \epsilon$. Then there is some $\alpha \subseteq [n]$, s.t. $\hat{f}_\alpha \geq 2\epsilon$.

For Boolean functions $\langle f, g\rangle$ is the fraction of inputs on which $f, g$ agree **minus** the fraction of inputs on which they disagree.

$$2\epsilon \leq \hat{f}_\alpha = \langle f, \chi_\alpha\rangle = \text{agree} - \text{disagree} = 2\text{agree} - 1$$

# Linearity Test

**GF(2)**
We want to show that if $\Pr_{x,y}[f(x) + f(y) = f(x+y)]$ is large than $f$ has a large agreement with a linear function.

**Hilbert space** (we prove)
Suppose that $f : \{+1,-1\}^n \to \{-1,1\}$ satisfies $\Pr_{x,y}[f(x)f(y) = f(xy)] \geq \frac{1}{2} + \epsilon$. Then there is some $\alpha \subseteq [n]$, s.t. $\hat{f}_\alpha \geq 2\epsilon$.

For Boolean functions $\langle f, g\rangle$ is the fraction of inputs on which $f, g$ agree **minus** the fraction of inputs on which they disagree.

$$2\epsilon \leq \hat{f}_\alpha = \langle f, \chi_\alpha\rangle = \text{agree} - \text{disagree} = 2\text{agree} - 1$$

This gives that the agreement between $f$ and $\chi_\alpha$ is at least $\frac{1}{2} + \epsilon$.

# Linearity Test

$$\Pr_{x,y}[f(xy) = f(x)f(y)] \geq \frac{1}{2} + \epsilon$$

is equivalent to

$$E_{x,y}[f(xy)f(x)f(y)] = \text{agreement} - \text{disagreement} \geq 2\epsilon$$

$$2\epsilon \le E_{x,y}\left[f(xy)f(x)f(y)\right]$$

$$2\epsilon \leq E_{x,y}\Big[f(xy)f(x)f(y)\Big]$$

$$= E_{x,y}\Big[\Big(\sum_\alpha \hat{f}_\alpha\chi_\alpha(xy)\Big)\cdot\Big(\sum_\beta \hat{f}_\beta\chi_\beta(x)\Big)\cdot\Big(\sum_\gamma \hat{f}_\gamma\chi_\gamma(y)\Big)\Big]$$

$$
\begin{aligned}
2\epsilon &\leq E_{x,y}\Big[f(xy)f(x)f(y)\Big] \\
&= E_{x,y}\Big[\Big(\sum\nolimits_{\alpha}\hat{f}_{\alpha}\chi_{\alpha}(xy)\Big)\cdot\Big(\sum\nolimits_{\beta}\hat{f}_{\beta}\chi_{\beta}(x)\Big)\cdot\Big(\sum\nolimits_{\gamma}\hat{f}_{\gamma}\chi_{\gamma}(y)\Big)\Big] \\
&= E_{x,y}\Big[\sum\nolimits_{\alpha,\beta,\gamma}\hat{f}_{\alpha}\hat{f}_{\beta}\hat{f}_{\gamma}\chi_{\alpha}(x)\chi_{\alpha}(y)\chi_{\beta}(x)\chi_{\gamma}(y)\Big]
\end{aligned}
$$

$$
\begin{aligned}
2\epsilon &\le E_{x,y}\Big[f(xy)f(x)f(y)\Big] \\
&= E_{x,y}\Big[\Big(\sum_\alpha \hat{f}_\alpha \chi_\alpha(xy)\Big)\cdot\Big(\sum_\beta \hat{f}_\beta \chi_\beta(x)\Big)\cdot\Big(\sum_\gamma \hat{f}_\gamma \chi_\gamma(y)\Big)\Big] \\
&= E_{x,y}\Big[\sum_{\alpha,\beta,\gamma} \hat{f}_\alpha \hat{f}_\beta \hat{f}_\gamma \chi_\alpha(x)\chi_\alpha(y)\chi_\beta(x)\chi_\gamma(y)\Big] \\
&= \sum_{\alpha,\beta,\gamma} \hat{f}_\alpha \hat{f}_\beta \hat{f}_\gamma \cdot E_x\Big[\chi_\alpha(x)\chi_\beta(x)\Big]E_y\Big[\chi_\alpha(y)\chi_\gamma(y)\Big]
\end{aligned}
$$

$$
\begin{aligned}
2\epsilon &\le E_{x,y}\left[ f(xy)f(x)f(y) \right] \\
&= E_{x,y}\left[ \left(\sum\nolimits_\alpha \hat{f}_\alpha \chi_\alpha(xy)\right) \cdot \left(\sum\nolimits_\beta \hat{f}_\beta \chi_\beta(x)\right) \cdot \left(\sum\nolimits_\gamma \hat{f}_\gamma \chi_\gamma(y)\right) \right] \\
&= E_{x,y}\left[ \sum\nolimits_{\alpha,\beta,\gamma} \hat{f}_\alpha \hat{f}_\beta \hat{f}_\gamma \chi_\alpha(x)\chi_\alpha(y)\chi_\beta(x)\chi_\gamma(y) \right] \\
&= \sum\nolimits_{\alpha,\beta,\gamma} \hat{f}_\alpha \hat{f}_\beta \hat{f}_\gamma \cdot E_x\left[\chi_\alpha(x)\chi_\beta(x)\right] E_y\left[\chi_\alpha(y)\chi_\gamma(y)\right] \\
&= \sum_\alpha \hat{f}_\alpha^3
\end{aligned}
$$

$$
\begin{aligned}
2\epsilon &\le E_{x,y}\Big[f(xy)f(x)f(y)\Big] \\
&= E_{x,y}\Big[\Big(\textstyle\sum_\alpha \hat{f}_\alpha \chi_\alpha(xy)\Big)\cdot\Big(\sum_\beta \hat{f}_\beta \chi_\beta(x)\Big)\cdot\Big(\sum_\gamma \hat{f}_\gamma \chi_\gamma(y)\Big)\Big] \\
&= E_{x,y}\Big[\textstyle\sum_{\alpha,\beta,\gamma} \hat{f}_\alpha \hat{f}_\beta \hat{f}_\gamma \chi_\alpha(x)\chi_\alpha(y)\chi_\beta(x)\chi_\gamma(y)\Big] \\
&= \textstyle\sum_{\alpha,\beta,\gamma} \hat{f}_\alpha \hat{f}_\beta \hat{f}_\gamma \cdot E_x\Big[\chi_\alpha(x)\chi_\beta(x)\Big] E_y\Big[\chi_\alpha(y)\chi_\gamma(y)\Big] \\
&= \sum_\alpha \hat{f}_\alpha^3 \\
&\le \max_\alpha \hat{f}_\alpha \cdot \sum_\alpha \hat{f}_\alpha^2 = \max_\alpha \hat{f}_\alpha
\end{aligned}
$$

# Probabilistic proof for Graph NonIsomorphism

GNI is the language of pairs of non-isomorphic graphs

# Probabilistic proof for Graph NonIsomorphism

GNI is the language of pairs of non-isomorphic graphs

Verifier gets input $(G_0, G_1)$ (two graphs with $n$-nodes)

# Probabilistic proof for Graph NonIsomorphism

GNI is the language of pairs of non-isomorphic graphs

Verifier gets input $(G_0, G_1)$ (two graphs with $n$-nodes)

It expects a proof of the following form:

- For any labeled $n$-node graph $H$ the $H$'s bit $P[H]$ of the proof fulfills

$$\begin{aligned} G_0 \equiv H &\implies P[H] = 0 \\ G_1 \equiv H &\implies P[H] = 1 \\ G_0, G_1 \not\equiv H &\implies P[H] = \text{arbitrary} \end{aligned}$$

# Probabilistic proof for Graph NonIsomorphism

**Verifier:**

- choose $b \in \{0,1\}$ at random
- take graph $G_b$ and apply a random permutation to obtain a labeled graph $H$
- check whether $P[H] = b$

# Probabilistic proof for Graph NonIsomorphism

**Verifier:**

- choose $b \in \{0, 1\}$ at random
- take graph $G_b$ and apply a random permutation to obtain a labeled graph $H$
- check whether $P[H] = b$

If $G_0 \not\equiv G_1$ then by using the obvious proof the verifier will always accept.

# Probabilistic proof for Graph NonIsomorphism

**Verifier:**

- choose $b \in \{0,1\}$ at random
- take graph $G_b$ and apply a random permutation to obtain a labeled graph $H$
- check whether $P[H] = b$

If $G_0 \not\equiv G_1$ then by using the obvious proof the verifier will always accept.

If $G_0 \not\equiv G_1$ a proof only accepts with probability $1/2$.

- suppose $\pi(G_0) = G_1$
- if we accept for $b = 1$ and permutation $\pi_{\text{rand}}$ we reject for permutation $b = 0$ and $\pi_{\text{rand}} \circ \pi$

# How to show Harndess of Approximation?

**Decision version of optimization problems:**

Suppose we have some maximization problem.

# How to show Harndess of Approximation?

**Decision version of optimization problems:**
Suppose we have some maximization problem.

The corresponding decision problem equips each instance with a parameter $k$ and asks whether we can obtain a solution value of at least $k$. (where infeasible solutions are assumed to have value $-\infty$)

# How to show Harndess of Approximation?

**Decision version of optimization problems:**
Suppose we have some maximization problem.

The corresponding decision problem equips each instance with a parameter $k$ and asks whether we can obtain a solution value of at least $k$. (where infeasible solutions are assumed to have value $-\infty$)

(Analogous for minimization problems.)

# How to show Harndess of Approximation?

**Decision version of optimization problems:**
Suppose we have some maximization problem.

The corresponding decision problem equips each instance with a parameter $k$ and asks whether we can obtain a solution value of at least $k$. (where infeasible solutions are assumed to have value $-\infty$)

(Analogous for minimization problems.)

This is the standard way to show that some optimization problem is e.g. NP-hard.

# How to show Harndess of Approximation?

**Gap version of optimization problems:**

Suppose we have some maximization problem.

# How to show Harndess of Approximation?

**Gap version of optimization problems:**

Suppose we have some maximization problem.

The corresponding $(\alpha, \beta)$-gap problem asks the following:

# How to show Harndess of Approximation?

**Gap version of optimization problems:**

Suppose we have some maximization problem.

The corresponding $(\alpha, \beta)$-gap problem asks the following:

Suppose we are given an instance $I$ and a promise that either $\text{opt}(I) \geq \beta$ or $\text{opt}(I) \leq \alpha$. Can we differentiate between these two cases?

# How to show Harndess of Approximation?

**Gap version of optimization problems:**
Suppose we have some maximization problem.

The corresponding $(\alpha, \beta)$-gap problem asks the following:

Suppose we are given an instance $I$ and a promise that either $\text{opt}(I) \geq \beta$ or $\text{opt}(I) \leq \alpha$. Can we differentiate between these two cases?

An algorithm $A$ has to output

- $A(I) = 1$ if $\text{opt}(I) \geq \beta$
- $A(I) = 0$ if $\text{opt}(I) \leq \alpha$
- $A(I) =$ arbitrary, otw

# How to show Harndess of Approximation?

**Gap version of optimization problems:**

Suppose we have some maximization problem.

The corresponding $(\alpha, \beta)$-gap problem asks the following:

Suppose we are given an instance $I$ and a promise that either $\text{opt}(I) \geq \beta$ or $\text{opt}(I) \leq \alpha$. Can we differentiate between these two cases?

An algorithm $A$ has to output

- $A(I) = 1$ if $\text{opt}(I) \geq \beta$
- $A(I) = 0$ if $\text{opt}(I) \leq \alpha$
- $A(I) =$ arbitrary, otw

Note that this is not a decision problem

An approximation algorithm with approximation guarantee $c \leq \beta/\alpha$ can solve an $(\alpha, \beta)$-gap problem.

# Constraint Satisfaction Problem

A $q$CSP $\phi$ consists of $m$ $n$-ary Boolean functions $\phi_1, \dots, \phi_m$ (constraints), where each function only depends on $q$ inputs. The goal is to maximize the number of satisifed constraints.

- $u \in \{0,1\}^n$ satsifies constraint $\phi_i$ if $\phi_i(u) = 1$
- $r(u) := \sum_i \phi_i(u)/m$ is fraction of satisfied constraints
- $\text{value}(\phi) = \max_u r(u)$
- $\phi$ is satisfiable if $\text{value}(\phi) = 1$.

3SAT is a constraint satsifaction problem with $q = 3$.

# Constraint Satisfaction Problem

**GAP version:**
A $\rho$GAP$q$CSP $\phi$ consists of $m$ $n$-ary Boolean functions $\phi_1, \dots, \phi_m$ (constraints), where each function only depends on $q$ inputs. We know that either $\phi$ is satisfiable or $\text{value}(\phi) < \rho$, and want to differentiate between these cases.

$\rho$GAP$q$CSP is NP-hard if for any $L \in \text{NP}$ there is a polytime computable function $f$ mapping strings to instances of $q$CSP s.t.

- $x \in L \Longrightarrow \text{value}(f(x)) = 1$
- $x \notin L \Longrightarrow \text{value}(f(x)) < \rho$

## Theorem 52

*There exists constants $q, \rho$ such that $\rho$GAP$q$CSP is NP-hard.*

We know that $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$.

We reduce 3SAT to $\rho$GAP$q$CSP.

3SAT has a PCP system in which the verifier makes a constant number of queries ($q$), and uses $c \log n$ random bits (for some $c$).

For input $x$ and $r \in \{0,1\}^{c \log n}$ define

- $V_{x,r}$ as function that maps a proof $\pi$ to the result (0/1) computed by the verifier when using proof $\pi$, instance $x$ and random coins $r$.
- $V_{x,r}$ only depends on $q$ bits of the proof

For any $x$ the collection $\phi$ of the $V_{x,r}$'s over all $r$ is polynomial size $q$CSP.

$\phi$ can be computed in polynomial time.

We know that $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$.

We reduce 3SAT to $\rho$GAP$q$CSP.

3SAT has a PCP system in which the verifier makes a constant number of queries ($q$), and uses $c \log n$ random bits (for some $c$).

For input $x$ and $r \in \{0,1\}^{c \log n}$ define

- $V_{x,r}$ as function that maps a proof $\pi$ to the result (0/1) computed by the verifier when using proof $\pi$, instance $x$ and random coins $r$.
- $V_{x,r}$ only depends on $q$ bits of the proof

For any $x$ the collection $\phi$ of the $V_{x,r}$'s over all $r$ is polynomial size $q$CSP.

$\phi$ can be computed in polynomial time.

We know that $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$.

We reduce 3SAT to $\rho$GAP$q$CSP.

3SAT has a PCP system in which the verifier makes a constant number of queries ($q$), and uses $c \log n$ random bits (for some $c$).

For input $x$ and $r \in \{0,1\}^{c \log n}$ define

- $V_{x,r}$ as function that maps a proof $\pi$ to the result (0/1) computed by the verifier when using proof $\pi$, instance $x$ and random coins $r$.
- $V_{x,r}$ only depends on $q$ bits of the proof

For any $x$ the collection $\phi$ of the $V_{x,r}$'s over all $r$ is polynomial size $q$CSP.

$\phi$ can be computed in polynomial time.

We know that $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$.

We reduce 3SAT to $\rho$GAP$q$CSP.

3SAT has a PCP system in which the verifier makes a constant number of queries ($q$), and uses $c \log n$ random bits (for some $c$).

For input $x$ and $r \in \{0,1\}^{c \log n}$ define

- $V_{x,r}$ as function that maps a proof $\pi$ to the result (0/1) computed by the verifier when using proof $\pi$, instance $x$ and random coins $r$.
- $V_{x,r}$ only depends on $q$ bits of the proof

For any $x$ the collection $\phi$ of the $V_{x,r}$'s over all $r$ is polynomial size $q$CSP.

$\phi$ can be computed in polynomial time.

We know that $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$.

We reduce 3SAT to $\rho$GAP$q$CSP.

3SAT has a PCP system in which the verifier makes a constant number of queries ($q$), and uses $c \log n$ random bits (for some $c$).

For input $x$ and $r \in \{0,1\}^{c \log n}$ define

- $V_{x,r}$ as function that maps a proof $\pi$ to the result (0/1) computed by the verifier when using proof $\pi$, instance $x$ and random coins $r$.
- $V_{x,r}$ only depends on $q$ bits of the proof

For any $x$ the collection $\phi$ of the $V_{x,r}$'s over all $r$ is polynomial size $q$CSP.

$\phi$ can be computed in polynomial time.

We know that $\mathrm{NP} \subseteq \mathrm{PCP}(\log n, 1)$.

We reduce 3SAT to $\rho$GAP$q$CSP.

3SAT has a PCP system in which the verifier makes a constant number of queries ($q$), and uses $c \log n$ random bits (for some $c$).

For input $x$ and $r \in \{0,1\}^{c \log n}$ define

- $V_{x,r}$ as function that maps a proof $\pi$ to the result (0/1) computed by the verifier when using proof $\pi$, instance $x$ and random coins $r$.
- $V_{x,r}$ only depends on $q$ bits of the proof

For any $x$ the collection $\phi$ of the $V_{x,r}$'s over all $r$ is polynomial size $q$CSP.

$\phi$ can be computed in polynomial time.

$$x \in 3\text{SAT} \implies \phi \text{ is satisfiable}$$

$$x \notin 3\text{SAT} \implies \text{value}(\phi) \leq \frac{1}{2}$$

This means that $\rho$GAP$q$CSP is NP-hard.

$$x \in \text{3SAT} \implies \phi \text{ is satisfiable}$$

$$x \notin \text{3SAT} \implies \text{value}(\phi) \leq \frac{1}{2}$$

This means that $\rho$GAP$q$CSP is NP-hard.

Suppose that $\rho$GAP$q$CSP is NP-hard for some constants $q, \rho$ ($\rho < 1$).

Suppose you get an input $x$, and have to decide whether $x \in L$.

We get a verifier as follows.

We use the reduction to map an input $x$ into an instance $\phi$ of $q$CSP.

The proof is considered to be an assignment to the variables.

We can check a random constraint $\phi_i$ by making $q$ queries. If $x \in L$ the verifier accepts with probability 1.

Otw. at most a $\rho$ fraction of constraints are satisfied by the proof, and the verifier accepts with probability at most $\rho$.

Hence, $L \in \mathrm{PCP}_{1,\rho}(\log_2 m, q)$, where $m$ is the number of constraints.

Suppose that $\rho$GAP$q$CSP is NP-hard for some constants $q, \rho$ ($\rho < 1$).

Suppose you get an input $x$, and have to decide whether $x \in L$.

We get a verifier as follows.

We use the reduction to map an input $x$ into an instance $\phi$ of $q$CSP.

The proof is considered to be an assignment to the variables.

We can check a random constraint $\phi_i$ by making $q$ queries. If $x \in L$ the verifier accepts with probability 1.

Otw. at most a $\rho$ fraction of constraints are satisfied by the proof, and the verifier accepts with probability at most $\rho$.

Hence, $L \in \mathrm{PCP}_{1,\rho}(\log_2 m, q)$, where $m$ is the number of constraints.

Suppose that $\rho$GAP$q$CSP is NP-hard for some constants $q, \rho$ ($\rho < 1$).

Suppose you get an input $x$, and have to decide whether $x \in L$.

We get a verifier as follows.

We use the reduction to map an input $x$ into an instance $\phi$ of $q$CSP.

The proof is considered to be an assignment to the variables.

We can check a random constraint $\phi_i$ by making $q$ queries. If $x \in L$ the verifier accepts with probability 1.

Otw. at most a $\rho$ fraction of constraints are satisfied by the proof, and the verifier accepts with probability at most $\rho$.

Hence, $L \in \mathrm{PCP}_{1,\rho}(\log_2 m, q)$, where $m$ is the number of constraints.

Suppose that $\rho$GAP$q$CSP is NP-hard for some constants $q, \rho$ ($\rho < 1$).

Suppose you get an input $x$, and have to decide whether $x \in L$.

We get a verifier as follows.

We use the reduction to map an input $x$ into an instance $\phi$ of $q$CSP.

The proof is considered to be an assignment to the variables.

We can check a random constraint $\phi_i$ by making $q$ queries. If $x \in L$ the verifier accepts with probability 1.

Otw. at most a $\rho$ fraction of constraints are satisfied by the proof, and the verifier accepts with probability at most $\rho$.

Hence, $L \in \mathrm{PCP}_{1,\rho}(\log_2 m, q)$, where $m$ is the number of constraints.

Suppose that $\rho$GAP$q$CSP is NP-hard for some constants $q, \rho$ ($\rho < 1$).

Suppose you get an input $x$, and have to decide whether $x \in L$.

We get a verifier as follows.

We use the reduction to map an input $x$ into an instance $\phi$ of $q$CSP.

The proof is considered to be an assignment to the variables.

We can check a random constraint $\phi_i$ by making $q$ queries. If $x \in L$ the verifier accepts with probability 1.

Otw. at most a $\rho$ fraction of constraints are satisfied by the proof, and the verifier accepts with probability at most $\rho$.

Hence, $L \in \mathrm{PCP}_{1,\rho}(\log_2 m, q)$, where $m$ is the number of constraints.

Suppose that $\rho$GAP$q$CSP is NP-hard for some constants $q, \rho$ ($\rho < 1$).

Suppose you get an input $x$, and have to decide whether $x \in L$.

We get a verifier as follows.

We use the reduction to map an input $x$ into an instance $\phi$ of $q$CSP.

The proof is considered to be an assignment to the variables.

We can check a random constraint $\phi_i$ by making $q$ queries. If $x \in L$ the verifier accepts with probability 1.

Otw. at most a $\rho$ fraction of constraints are satisfied by the proof, and the verifier accepts with probability at most $\rho$.

Hence, $L \in \mathrm{PCP}_{1,\rho}(\log_2 m, q)$, where $m$ is the number of constraints.

Suppose that $\rho$GAP$q$CSP is NP-hard for some constants $q, \rho$ $(\rho < 1)$.

Suppose you get an input $x$, and have to decide whether $x \in L$.

We get a verifier as follows.

We use the reduction to map an input $x$ into an instance $\phi$ of $q$CSP.

The proof is considered to be an assignment to the variables.

We can check a random constraint $\phi_i$ by making $q$ queries. If $x \in L$ the verifier accepts with probability 1.

Otw. at most a $\rho$ fraction of constraints are satisfied by the proof, and the verifier accepts with probability at most $\rho$.

Hence, $L \in \mathrm{PCP}_{1,\rho}(\log_2 m, q)$, where $m$ is the number of constraints.

## Theorem 53

*For any positive constants $\epsilon, \delta > 0$, it is the case that $\mathrm{NP} \subseteq \mathrm{PCP}_{1-\epsilon, 1/2+\delta}(\log n, 3)$, and the verifier is restricted to use only the functions odd and even.*

It is NP-hard to approximate an ODD/EVEN constraint satisfaction problem by a factor better than $1/2 + \delta$, for any constant $\delta$.

## Theorem 54

*For any positive constant $\delta > 0$, $\mathrm{NP} \subseteq \mathrm{PCP}_{1, 7/8+\delta}(\mathcal{O}(\log n), 3)$ and the verifier is restricted to use only functions that check the OR of three bits or their negations.*

It is NP-hard to approximate 3SAT better than $7/8 + \delta$.

The following GAP-problem is NP-hard for any $\epsilon > 0$.

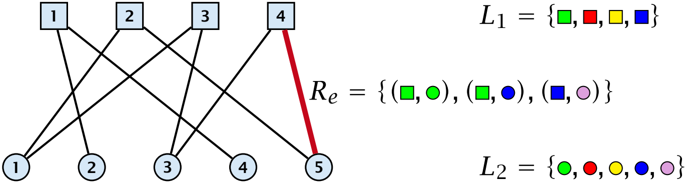
Given a graph $G = (V, E)$ composed of $m$ independent sets of size $3$ ($|V| = 3m$). Distinguish between

- the graph has a CLIQUE of size $m$
- the largest CLIQUE has size at most $(7/8 + \epsilon)m$

# Label Cover

**Input:**

- bipartite graph $G = (V_1, V_2, E)$
- label sets $L_1, L_2$
- for every edge $(u, v) \in E$ a relation $R_{u,v} \subseteq L_1 \times L_2$ that describe assignments that make the edge *happy*.
- maximize number of happy edges



# Label Cover

- an instance of label cover is $(d_1, d_2)$-regular if every vertex in $L_1$ has degree $d_1$ and every vertex in $L_2$ has degree $d_2$.
- if every vertex has the same degree $d$ the instance is called $d$-regular

**Minimization version:**

- assign a set $L_x \subseteq L_1$ of labels to every node $x \in L_1$ and a set $L_y \subseteq L_2$ to every node $x \in L_2$
- make sure that for every edge $(x, y)$ there is $\ell_x \in L_x$ and $\ell_y \in L_y$ s.t. $(\ell_x, \ell_y) \in R_{x,y}$
- minimize $\sum_{x \in L_1} |L_x| + \sum_{y \in L_2} |L_y|$ (total labels used)

# MAX E3SAT via Label Cover

instance:

$\Phi(x) = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_4 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4)$

corresponding graph:




label sets: $L_1 = \{T, F\}^3, L_2 = \{T, F\}$ ($T$=true, $F$=false)

relation: $R_{C,x_i} = \{((u_i, u_j, u_k), u_i)\}$, where the clause $C$ is over variables $x_i, x_j, x_k$ and assignment $(u_i, u_j, u_k)$ satisfies $C$

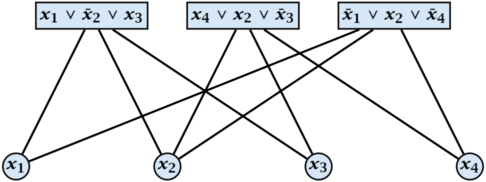
$$R = \{((F,F,F),F), ((F,T,F),F), ((F,F,T),T), ((F,T,T),T), ((T,T,T),T), ((T,T,F),F), ((T,F,F),F)\}$$

# MAX E3SAT via Label Cover

instance:

$\Phi(x) = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_4 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4)$

corresponding graph:




label sets: $L_1 = \{T, F\}^3, L_2 = \{T, F\}$ ($T$=true, $F$=false)

relation: $R_{C,x_i} = \{((u_i, u_j, u_k), u_i)\}$, where the clause $C$ is over variables $x_i, x_j, x_k$ and assignment $(u_i, u_j, u_k)$ satisfies $C$

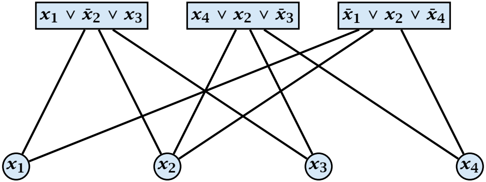
$$R = \{((F,F,F),F), ((F,T,F),F), ((F,F,T),T), ((F,T,T),T), ((T,T,T),T), ((T,T,F),F), ((T,F,F),F)\}$$

# MAX E3SAT via Label Cover

instance:

$\Phi(x) = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_4 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4)$

corresponding graph:




label sets: $L_1 = \{T, F\}^3, L_2 = \{T, F\}$ ($T$=true, $F$=false)

relation: $R_{C,x_i} = \{((u_i, u_j, u_k), u_i)\}$, where the clause $C$ is over variables $x_i, x_j, x_k$ and assignment $(u_i, u_j, u_k)$ satisfies $C$

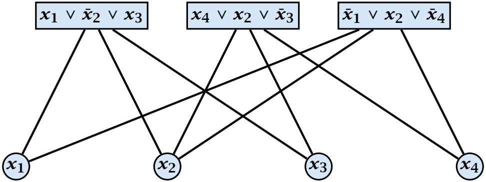
$$R = \{((F,F,F),F), ((F,T,F),F), ((F,F,T),T), ((F,T,T),T), ((T,T,T),T), ((T,T,F),F), ((T,F,F),F)\}$$

# MAX E3SAT via Label Cover

instance:

$\Phi(x) = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_4 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4)$

corresponding graph:




label sets: $L_1 = \{T, F\}^3, L_2 = \{T, F\}$ ($T$=true, $F$=false)

relation: $R_{C,x_i} = \{((u_i, u_j, u_k), u_i)\}$, where the clause $C$ is over variables $x_i, x_j, x_k$ and assignment $(u_i, u_j, u_k)$ satisfies $C$

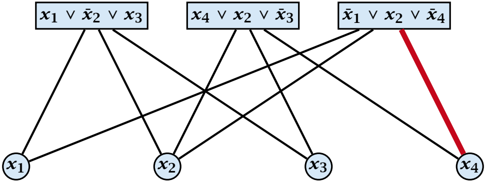
$$R = \{((F,F,F),F),((F,T,F),F),((F,F,T),T),((F,T,T),T),$$
$$((T,T,T),T),((T,T,F),F),((T,F,F),F)\}$$

# MAX E3SAT via Label Cover

instance:

$\Phi(x) = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_4 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_4)$

corresponding graph:




label sets: $L_1 = \{T, F\}^3, L_2 = \{T, F\}$ ($T$=true, $F$=false)

relation: $R_{C,x_i} = \{((u_i, u_j, u_k), u_i)\}$, where the clause $C$ is over variables $x_i, x_j, x_k$ and assignment $(u_i, u_j, u_k)$ satisfies $C$

$$R = \{((F,F,F),F), ((F,T,F),F), ((F,F,T),T), ((F,T,T),T), ((T,T,T),T), ((T,T,F),F), ((T,F,F),F)\}$$

# MAX E3SAT via Label Cover

**Lemma 55**
*If we can satisfy $k$ out of $m$ clauses in $\phi$ we can make at least $3k + 2(m - k)$ edges happy.*

**Proof:**

- for $V_2$ use the setting of the assignment that satisfies $k$ clauses
- for satisfied clauses in $V_1$ use the corresponding assignment to the clause-variables (gives $3k$ happy edges)
- for unsatisfied clauses flip assignment of one of the variables; this makes one incident edge unhappy (gives $2(m - k)$ happy edges)

# MAX E3SAT via Label Cover

**Lemma 55**

*If we can satisfy $k$ out of $m$ clauses in $\phi$ we can make at least $3k + 2(m - k)$ edges happy.*

**Proof:**

- for $V_2$ use the setting of the assignment that satisfies $k$ clauses
- for satisfied clauses in $V_1$ use the corresponding assignment to the clause-variables (gives $3k$ happy edges)
- for unsatisfied clauses flip assignment of one of the variables; this makes one incident edge unhappy (gives $2(m - k)$ happy edges)

# MAX E3SAT via Label Cover

**Lemma 55**

*If we can satisfy $k$ out of $m$ clauses in $\phi$ we can make at least $3k + 2(m - k)$ edges happy.*

**Proof:**

- for $V_2$ use the setting of the assignment that satisfies $k$ clauses
- for satisfied clauses in $V_1$ use the corresponding assignment to the clause-variables (gives $3k$ happy edges)
- for unsatisfied clauses flip assignment of one of the variables; this makes one incident edge unhappy (gives $2(m - k)$ happy edges)

# MAX E3SAT via Label Cover

### Lemma 55

*If we can satisfy* $k$ *out of* $m$ *clauses in* $\phi$ *we can make at least* $3k + 2(m - k)$ *edges happy.*

**Proof:**

- for $V_2$ use the setting of the assignment that satisfies $k$ clauses
- for satisfied clauses in $V_1$ use the corresponding assignment to the clause-variables (gives $3k$ happy edges)
- for unsatisfied clauses flip assignment of one of the variables; this makes one incident edge unhappy (gives $2(m - k)$ happy edges)

# MAX E3SAT via Label Cover

**Lemma 56**

*If we can satisfy at most $k$ clauses in $\Phi$ we can make at most $3k + 2(m - k) = 2m + k$ edges happy.*

**Proof:**

- the labeling of nodes in $V_2$ gives an assignment
- every unsatisfied clause in this assignment cannot be assigned a label that satisfies all 3 incident edges
- hence at most $3m - (m - k) = 2m + k$ edges are happy

# MAX E3SAT via Label Cover

**Lemma 56**

*If we can satisfy at most* $k$ *clauses in* $\Phi$ *we can make at most* $3k + 2(m - k) = 2m + k$ *edges happy.*

**Proof:**

- the labeling of nodes in $V_2$ gives an assignment
- every unsatisfied clause in this assignment cannot be assigned a label that satisfies all 3 incident edges
- hence at most $3m - (m - k) = 2m + k$ edges are happy

# MAX E3SAT via Label Cover

**Lemma 56**

*If we can satisfy at most $k$ clauses in $\Phi$ we can make at most $3k + 2(m - k) = 2m + k$ edges happy.*

**Proof:**

- the labeling of nodes in $V_2$ gives an assignment
- every unsatisfied clause in this assignment cannot be assigned a label that satisfies all 3 incident edges
- hence at most $3m - (m - k) = 2m + k$ edges are happy

# MAX E3SAT via Label Cover

**Lemma 56**
*If we can satisfy at most $k$ clauses in $\Phi$ we can make at most $3k + 2(m - k) = 2m + k$ edges happy.*

**Proof:**

- the labeling of nodes in $V_2$ gives an assignment
- every unsatisfied clause in this assignment cannot be assigned a label that satisfies all 3 incident edges
- hence at most $3m - (m - k) = 2m + k$ edges are happy

# Hardness for Label Cover

We cannot distinguish between the following two cases

- all $3m$ edges can be made happy
- at most $2m + (7/8 + \epsilon)m \approx (\frac{23}{8} + \epsilon)m$ out of the $3m$ edges can be made happy

Hence, we cannot obtain an approximation constant $\alpha > \frac{23}{24}$.

Here $\alpha$ is a constant!!! Maybe a guarantee of the form $\frac{23}{8} + \frac{1}{m}$ is possible.

# Hardness for Label Cover

We cannot distinguish between the following two cases

- all $3m$ edges can be made happy
- at most $2m + (7/8 + \epsilon)m \approx (\frac{23}{8} + \epsilon)m$ out of the $3m$ edges can be made happy

Hence, we cannot obtain an approximation constant $\alpha > \frac{23}{24}$.

Here $\alpha$ is a constant!!! Maybe a guarantee of the form $\frac{23}{8} + \frac{1}{m}$ is possible.

# Hardness for Label Cover

We cannot distinguish between the following two cases

- all $3m$ edges can be made happy
- at most $2m + (7/8 + \epsilon)m \approx (\frac{23}{8} + \epsilon)m$ out of the $3m$ edges can be made happy

Hence, we cannot obtain an approximation constant $\alpha > \frac{23}{24}$.

Here $\alpha$ is a constant!!! Maybe a guarantee of the form $\frac{23}{8} + \frac{1}{m}$ is possible.

# $(3, 5)$-regular instances

**Theorem 57**

*There is a constant $\rho$ s.t. MAXE3SAT is hard to approximate with a factor of $\rho$ even if restricted to instances where a variable appears in exactly 5 clauses.*

Then our reduction has the following properties:

- the resulting Label Cover instance is $(3, 5)$-regular
- it is hard to approximate for a constant $\alpha < 1$
- given a label $\ell_1$ for $x$ there is at most one label $\ell_2$ for $y$ that makes edge $(x, y)$ happy (uniqueness property)

# $(3,5)$-regular instances

**Theorem 57**
*There is a constant $\rho$ s.t. MAXE3SAT is hard to approximate with a factor of $\rho$ even if restricted to instances where a variable appears in exactly 5 clauses.*

Then our reduction has the following properties:

- the resulting Label Cover instance is $(3,5)$-regular
- it is hard to approximate for a constant $\alpha < 1$
- given a label $\ell_1$ for $x$ there is at most one label $\ell_2$ for $y$ that makes edge $(x,y)$ happy (uniqueness property)

# Regular instances

**Theorem 58**
*If for a particular constant $\alpha < 1$ there is an $\alpha$-approximation algorithm for Label Cover on 15-regular instances than P=NP.*

Given a label $\ell_1$ for $x \in V_1$ there is at most one label $\ell_2$ for $y$ that makes $(x, y)$ happy. (uniqueness property)

# Regular instances

proof...

# Boosting

Given Label Cover instance $I$ with $G = (V_1, V_2, E)$, label sets $L_1$ and $L_2$ we construct a new instance $I'$:

- $V_1' = V_1^k = V_1 \times \cdots \times V_1$
- $V_2' = V_2^k = V_2 \times \cdots \times V_2$
- $L_1' = L_1^k = L_1 \times \cdots \times L_1$
- $L_2' = L_2^k = L_2 \times \cdots \times L_2$
- $E' = E^k = E \times \cdots \times E$

An edge $((x_1, \dots, x_k), (y_1, \dots, y_k))$ whose end-points are labelled by $(\ell_1^x, \dots, \ell_k^x)$ and $(\ell_1^y, \dots, \ell_k^y)$ is happy if $(\ell_i^x, \ell_i^y) \in R_{x_i, y_i}$ for all $i$.

# Boosting

If $I$ is regular than also $I'$.

If $I$ has the uniqueness property than also $I'$.

**Theorem 59**
*There is a constant $c > 0$ such if $\mathrm{OPT}(I) = |E|(1-\delta)$ then $\mathrm{OPT}(I') \le |E'|(1-\delta)^{\frac{ck}{\log L}}$, where $L = |L_1| + |L_2|$ denotes total number of labels in $I$.*

proof is highly non-trivial

# Boosting

If $I$ is regular than also $I'$.

If $I$ has the uniqueness property than also $I'$.

**Theorem 59**
*There is a constant $c > 0$ such if $\mathrm{OPT}(I) = |E|(1-\delta)$ then $\mathrm{OPT}(I') \leq |E'|(1-\delta)^{\frac{ck}{\log L}}$, where $L = |L_1| + |L_2|$ denotes total number of labels in $I$.*

proof is highly non-trivial

# Boosting

If $I$ is regular than also $I'$.

If $I$ has the uniqueness property than also $I'$.

**Theorem 59**

*There is a constant $c > 0$ such if $\mathrm{OPT}(I) = |E|(1-\delta)$ then $\mathrm{OPT}(I') \leq |E'|(1-\delta)^{\frac{ck}{\log L}}$, where $L = |L_1| + |L_2|$ denotes total number of labels in $I$.*

proof is highly non-trivial

# Boosting

If $I$ is regular than also $I'$.

If $I$ has the uniqueness property than also $I'$.

**Theorem 59**

*There is a constant* $c > 0$ *such if* $\mathrm{OPT}(I) = |E|(1-\delta)$ *then* $\mathrm{OPT}(I') \leq |E'|(1-\delta)^{\frac{ck}{\log L}}$*, where* $L = |L_1| + |L_2|$ *denotes total number of labels in* $I$*.*

proof is highly non-trivial

# Boosting

If $I$ is regular than also $I'$.

If $I$ has the uniqueness property than also $I'$.

**Theorem 59**

*There is a constant* $c > 0$ *such if* $\mathrm{OPT}(I) = |E|(1-\delta)$ *then* $\mathrm{OPT}(I') \leq |E'|(1-\delta)^{\frac{ck}{\log L}}$*, where* $L = |L_1| + |L_2|$ *denotes total number of labels in* $I$*.*

proof is highly non-trivial

## Theorem 60

*There are constants $c > 0$, $\delta < 1$ s.t. for any $k$ we cannot distinguish regular instances for Label Cover in which either*

- $\mathrm{OPT}(I) = |E|$, *or*
- $\mathrm{OPT}(I) = |E|(1-\delta)^{\frac{ck}{\log 10}}$

*unless each problem in NP has an algorithm running in time $\mathcal{O}(n^{\mathcal{O}(k)})$.*

## Corollary 61

*There is no $\alpha$-approximation for Label Cover for any constant $\alpha$.*

# Set Cover

**Theorem 62**

*There exist regular Label Cover instances s.t. we cannot distinguish whether*

- *all edges are satisfiable, or*
- *at most a* $1/\log^2(|L_2||E|)$*-fraction is satisfiable*

*unless NP-problems have algorithms with running time* $\mathcal{O}(n^{\mathcal{O}(\log\log n)})$.

choose $k = \frac{2\log 10}{c}\log_{1/(1-\delta)}(\log(|L_2||E|)) = \mathcal{O}(\log\log n)$.

# Set Cover

**Partition System $(s, t, h)$**

- universe $U$ of size $s$
- $t$ pairs of sets $(A_1, \bar{A}_1), \dots, (A_t, \bar{A}_t)$;
  $A_i \subseteq U, \bar{A}_i = U \setminus A_i$
- choosing from any $h$ pairs only one of $A_i$, $\bar{A}_i$ we do not cover the whole set $U$

For any $h$, $t$ with $h \leq t$ there exist systems with $s = |U| \leq 2^{2h+2}t^2$.

# Set Cover

Given a Label Cover instance we construct a Set Cover instance;

The universe is $E \times U$, where $U$ is the universe of some partition system; ($t = |L_2|$, $h = (\log |E||L_2|)$)

for all $v \in V_2, j \in L_2$

$$S_{v,j} = \{((u,v),a) \mid (u,v) \in E, a \in A_j\}$$

for all $u \in V_1, i \in L_1$

$$S_{u,i} = \{((u,v),a) \mid (u,v) \in E, a \in \bar{A}_j, \text{ where } (i,j) \in R_{(u,v)}\}$$

note that $S_{u,i}$ is well-defined because of the uniqueness property

# Set Cover

Given a Label Cover instance we construct a Set Cover instance;

The universe is $E \times U$, where $U$ is the universe of some partition system; ($t = |L_2|$, $h = (\log |E||L_2|)$)

for all $v \in V_2, j \in L_2$

$$S_{v,j} = \{((u,v),a) \mid (u,v) \in E, a \in A_j\}$$

for all $u \in V_1, i \in L_1$

$$S_{u,i} = \{((u,v),a) \mid (u,v) \in E, a \in \bar{A}_j, \text{ where } (i,j) \in R_{(u,v)}\}$$

note that $S_{u,i}$ is well-defined because of the uniqueness property

# Set Cover

Given a Label Cover instance we construct a Set Cover instance;

The universe is $E \times U$, where $U$ is the universe of some partition system; ($t = |L_2|$, $h = (\log |E||L_2|)$)

for all $v \in V_2, j \in L_2$

$$S_{v,j} = \{((u,v),a) \mid (u,v) \in E, a \in A_j\}$$

for all $u \in V_1, i \in L_1$

$$S_{u,i} = \{((u,v),a) \mid (u,v) \in E, a \in \bar{A}_j, \text{ where } (i,j) \in R_{(u,v)}\}$$

note that $S_{u,i}$ is well-defined because of the uniqueness property

# Set Cover

Given a Label Cover instance we construct a Set Cover instance;

The universe is $E \times U$, where $U$ is the universe of some partition system; ($t = |L_2|$, $h = (\log |E||L_2|)$)

for all $v \in V_2, j \in L_2$

$$S_{v,j} = \{((u,v),a) \mid (u,v) \in E, a \in A_j\}$$

for all $u \in V_1, i \in L_1$

$$S_{u,i} = \{((u,v),a) \mid (u,v) \in E, a \in \bar{A}_j, \text{ where } (i,j) \in R_{(u,v)}\}$$

note that $S_{u,i}$ is well-defined because of the uniqueness property

# Set Cover

Given a Label Cover instance we construct a Set Cover instance;

The universe is $E \times U$, where $U$ is the universe of some partition system; ($t = |L_2|$, $h = (\log |E||L_2|)$)

for all $v \in V_2, j \in L_2$

$$S_{v,j} = \{((u,v),a) \mid (u,v) \in E, a \in A_j\}$$

for all $u \in V_1, i \in L_1$

$$S_{u,i} = \{((u,v),a) \mid (u,v) \in E, a \in \bar{A}_j, \text{ where } (i,j) \in R_{(u,v)}\}$$

note that $S_{u,i}$ is well-defined because of the uniqueness property

# Set Cover

Given a Label Cover instance we construct a Set Cover instance;

The universe is $E \times U$, where $U$ is the universe of some partition system; ($t = |L_2|$, $h = (\log |E||L_2|)$)

for all $v \in V_2, j \in L_2$

$$S_{v,j} = \{((u,v),a) \mid (u,v) \in E, a \in A_j\}$$

for all $u \in V_1, i \in L_1$

$$S_{u,i} = \{((u,v),a) \mid (u,v) \in E, a \in \bar{A}_j, \text{ where } (i,j) \in R_{(u,v)}\}$$

note that $S_{u,i}$ is well-defined because of the uniqueness property

Suppose that we can make all edges happy.

Choose sets $S_{u,i}$'s and $S_{v,j}$'s, where $i$ is the label we assigned to $u$, and $j$ the label for $v$. ($|V_1|+|V_2|$ sets)

For an edge $(u,v)$, $S_{v,j}$ contains $\{(u,v)\} \times A_j$. For a happy edge $S_{u,i}$ contains $\{(u,v)\} \times \bar{A}_j$.

Since all edges are happy we have covered the whole universe.

Suppose that we can make all edges happy.

Choose sets $S_{u,i}$'s and $S_{v,j}$'s, where $i$ is the label we assigned to $u$, and $j$ the label for $v$. ($|V_1|+|V_2|$ sets)

For an edge $(u,v)$, $S_{v,j}$ contains $\{(u,v)\} \times A_j$. For a happy edge $S_{u,i}$ contains $\{(u,v)\} \times \bar{A}_j$.

Since all edges are happy we have covered the whole universe.

Suppose that we can make all edges happy.

Choose sets $S_{u,i}$'s and $S_{v,j}$'s, where $i$ is the label we assigned to $u$, and $j$ the label for $v$. ($|V_1|+|V_2|$ sets)

For an edge $(u,v)$, $S_{v,j}$ contains $\{(u,v)\} \times A_j$. For a happy edge $S_{u,i}$ contains $\{(u,v)\} \times \bar{A}_j$.

Since all edges are happy we have covered the whole universe.

Suppose that we can make all edges happy.

Choose sets $S_{u,i}$'s and $S_{v,j}$'s, where $i$ is the label we assigned to $u$, and $j$ the label for $v$. ($|V_1|+|V_2|$ sets)

For an edge $(u,v)$, $S_{v,j}$ contains $\{(u,v)\} \times A_j$. For a happy edge $S_{u,i}$ contains $\{(u,v)\} \times \bar{A}_j$.

Since all edges are happy we have covered the whole universe.

Suppose that we can make all edges happy.

Choose sets $S_{u,i}$'s and $S_{v,j}$'s, where $i$ is the label we assigned to $u$, and $j$ the label for $v$. ($|V_1|+|V_2|$ sets)

For an edge $(u,v)$, $S_{v,j}$ contains $\{(u,v)\}\times A_j$. For a happy edge $S_{u,i}$ contains $\{(u,v)\}\times \bar{A}_j$.

Since all edges are happy we have covered the whole universe.

## Lemma 63

*Given a solution to the set cover instance using at most $\frac{h}{8}(|V_1| + |V_2|)$ sets we can find a solution to the Label Cover instance satisfying at least $\frac{2}{h^2}|E|$ edges.*

- $n_u$: number of $S_{u,i}$'s in cover
- $n_v$: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge $(u,v)$ we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i,j) \in R_{u,v}$ (making $(u,v)$ happy)
- we choose a random label for $u$ from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for $v$ from the (at most $h/2$) $S_{v,j}$-sets
- $(u,v)$ gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

- $n_u$: number of $S_{u,i}$'s in cover
- $n_v$: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge $(u, v)$ we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i, j) \in R_{u,v}$ (making $(u, v)$ happy)
- we choose a random label for $u$ from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for $v$ from the (at most $h/2$) $S_{v,j}$-sets
- $(u, v)$ gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

- $n_u$: number of $S_{u,i}$'s in cover
- $n_v$: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge $(u,v)$ we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i,j) \in R_{u,v}$ (making $(u,v)$ happy)
- we choose a random label for $u$ from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for $v$ from the (at most $h/2$) $S_{v,j}$-sets
- $(u,v)$ gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

- $n_u$: number of $S_{u,i}$'s in cover
- $n_v$: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge $(u, v)$ we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i, j) \in R_{u,v}$ (making $(u, v)$ happy)
- we choose a random label for $u$ from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for $v$ from the (at most $h/2$) $S_{v,j}$-sets
- $(u, v)$ gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

- $n_u$: number of $S_{u,i}$'s in cover
- $n_v$: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge $(u,v)$ we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i,j) \in R_{u,v}$ (making $(u,v)$ happy)
- we choose a random label for $u$ from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for $v$ from the (at most $h/2$) $S_{v,j}$-sets
- $(u,v)$ gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

- $n_u$: number of $S_{u,i}$'s in cover
- $n_v$: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge $(u,v)$ we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i,j) \in R_{u,v}$ (making $(u,v)$ happy)
- we choose a random label for $u$ from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for $v$ from the (at most $h/2$) $S_{v,j}$-sets
- $(u,v)$ gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

- $n_u$: number of $S_{u,i}$'s in cover
- $n_v$: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge $(u, v)$ we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i, j) \in R_{u,v}$ (making $(u, v)$ happy)
- we choose a random label for $u$ from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for $v$ from the (at most $h/2$) $S_{v,j}$-sets
- $(u, v)$ gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

- $n_u$: number of $S_{u,i}$'s in cover
- $n_v$: number of $S_{v,j}$'s in cover
- at most 1/4 of the vertices can have $n_u, n_v \geq h/2$; mark these vertices
- at least half of the edges have both end-points unmarked, as the graph is regular
- for such an edge $(u, v)$ we must have chosen $S_{u,i}$ and a corresponding $S_{v,j}$, s.t. $(i, j) \in R_{u,v}$ (making $(u, v)$ happy)
- we choose a random label for $u$ from the (at most $h/2$) chosen $S_{u,i}$-sets and a random label for $v$ from the (at most $h/2$) $S_{v,j}$-sets
- $(u, v)$ gets happy with probability at least $4/h^2$
- hence we make an $2/h^2$-fraction of edges happy

# Set Cover

**Theorem 64**
*There is no* $\frac{1}{32}\log N$*-approximation for the unweighted Set Cover problem unless problems in NP can be solved in time* $\mathcal{O}(n^{\mathcal{O}(\log\log n)})$.

Given label cover instance $(V_1, V_2, E)$, label sets $L_1$ and $L_2$;

Set $h = \log(|E||L_2|)$ and $t = |L_2|$; Size of partition system is

$$s = |U| = 2^{2h+2}t^2 = 4(|E||L_2|)^2|L_2|^2 = 4|E|^2|L_2|^4$$

The size of the ground set is then

$$N = |E||U| = 4|E|^3|L_2|^4 \le (|E||L_2|)^4$$

for sufficiently large $|E|$. Then $h \ge \frac{1}{4}\log N$.

If we get an instance where all edges are satisfiable there exists a cover of size only $|V_1| + |V_2|$.

If we find a cover of size at most $\frac{h}{8}(|V_1| + |V_2|)$ we can use this to satisfy at least a fraction of $2/h^2 \ge 1/\log^2(|E||L_2|)$ of the edges. this is not possible...

Given label cover instance $(V_1, V_2, E)$, label sets $L_1$ and $L_2$;

Set $h = \log(|E||L_2|)$ and $t = |L_2|$; Size of partition system is

$$s = |U| = 2^{2h+2}t^2 = 4(|E||L_2|)^2|L_2|^2 = 4|E|^2|L_2|^4$$

The size of the ground set is then

$$N = |E||U| = 4|E|^3|L_2|^4 \le (|E||L_2|)^4$$

for sufficiently large $|E|$. Then $h \ge \frac{1}{4}\log N$.

If we get an instance where all edges are satisfiable there exists a cover of size only $|V_1| + |V_2|$.

If we find a cover of size at most $\frac{h}{8}(|V_1| + |V_2|)$ we can use this to satisfy at least a fraction of $2/h^2 \ge 1/\log^2(|E||L_2|)$ of the edges. this is not possible...

Given label cover instance $(V_1, V_2, E)$, label sets $L_1$ and $L_2$;

Set $h = \log(|E||L_2|)$ and $t = |L_2|$; Size of partition system is

$$s = |U| = 2^{2h+2}t^2 = 4(|E||L_2|)^2|L_2|^2 = 4|E|^2|L_2|^4$$

The size of the ground set is then

$$N = |E||U| = 4|E|^3|L_2|^4 \leq (|E||L_2|)^4$$

for sufficiently large $|E|$. Then $h \geq \frac{1}{4}\log N$.

If we get an instance where all edges are satisfiable there exists a cover of size only $|V_1| + |V_2|$.

If we find a cover of size at most $\frac{h}{8}(|V_1| + |V_2|)$ we can use this to satisfy at least a fraction of $2/h^2 \geq 1/\log^2(|E||L_2|)$ of the edges. this is not possible...

Given label cover instance $(V_1, V_2, E)$, label sets $L_1$ and $L_2$;

Set $h = \log(|E||L_2|)$ and $t = |L_2|$; Size of partition system is

$$s = |U| = 2^{2h+2}t^2 = 4(|E||L_2|)^2|L_2|^2 = 4|E|^2|L_2|^4$$

The size of the ground set is then

$$N = |E||U| = 4|E|^3|L_2|^4 \leq (|E||L_2|)^4$$

for sufficiently large $|E|$. Then $h \geq \frac{1}{4}\log N$.

If we get an instance where all edges are satisfiable there exists a cover of size only $|V_1| + |V_2|$.

If we find a cover of size at most $\frac{h}{8}(|V_1| + |V_2|)$ we can use this to satisfy at least a fraction of $2/h^2 \geq 1/\log^2(|E||L_2|)$ of the edges. this is not possible...

Given label cover instance $(V_1, V_2, E)$, label sets $L_1$ and $L_2$;

Set $h = \log(|E||L_2|)$ and $t = |L_2|$; Size of partition system is

$$s = |U| = 2^{2h+2}t^2 = 4(|E||L_2|)^2|L_2|^2 = 4|E|^2|L_2|^4$$

The size of the ground set is then

$$N = |E||U| = 4|E|^3|L_2|^4 \le (|E||L_2|)^4$$

for sufficiently large $|E|$. Then $h \ge \frac{1}{4}\log N$.

If we get an instance where all edges are satisfiable there exists a cover of size only $|V_1| + |V_2|$.

If we find a cover of size at most $\frac{h}{8}(|V_1| + |V_2|)$ we can use this to satisfy at least a fraction of $2/h^2 \ge 1/\log^2(|E||L_2|)$ of the edges. this is not possible...

# Partition Systems

**Lemma 65**
*Given $h$ and $t$ there is a partition system of size $s = 2^h h \ln(4t) \le 2^{2h+2}t^2$.*

We pick $t$ sets at random from the possible $2^{|U|}$ subsets of $U$.

Fix a choice of $h$ of these sets, and a choice of $h$ bits (whether we choose $A_i$ or $\bar{A}_i$). There are $2^h \cdot \binom{t}{h}$ such choices.

# Partition Systems

**Lemma 65**

*Given $h$ and $t$ there is a partition system of size $s = 2^h h \ln(4t) \le 2^{2h+2} t^2$.*

We pick $t$ sets at random from the possible $2^{|U|}$ subsets of $U$.

Fix a choice of $h$ of these sets, and a choice of $h$ bits (whether we choose $A_i$ or $\bar{A}_i$). There are $2^h \cdot \binom{t}{h}$ such choices.

# Partition Systems

**Lemma 65**
*Given $h$ and $t$ there is a partition system of size* $s = 2^h h \ln(4t) \le 2^{2h+2} t^2$.

We pick $t$ sets at random from the possible $2^{|U|}$ subsets of $U$.

Fix a choice of $h$ of these sets, and a choice of $h$ bits (whether we choose $A_i$ or $\bar{A}_i$). There are $2^h \cdot \binom{t}{h}$ such choices.

What is the probability that a given choice covers $U$?

The probability that an element $u \in A_i$ is $1/2$ (same for $\bar{A}_i$).

The probability that $u$ is covered is $1 - \frac{1}{2^h}$.

The probability that all $u$ are covered is $(1 - \frac{1}{2^h})^s$

The probability that there exists a choice such that all $u$ are covered is at most

$$\binom{t}{h} 2^h \left(1 - \frac{1}{2^h}\right)^s \leq (2t)^h e^{-s/2^h} = (2t)^h \cdot e^{-h \ln(4t)} \leq \frac{1}{2^h}$$

What is the probability that a given choice covers $U$?

The probability that an element $u \in A_i$ is $1/2$ (same for $\bar{A}_i$).

The probability that $u$ is covered is $1 - \frac{1}{2^h}$.

The probability that all $u$ are covered is $(1 - \frac{1}{2^h})^s$

The probability that there exists a choice such that all $u$ are covered is at most

$$\binom{t}{h} 2^h \left(1 - \frac{1}{2^h}\right)^s \le (2t)^h e^{-s/2^h} = (2t)^h \cdot e^{-h \ln(4t)} \le \frac{1}{2^h}$$

What is the probability that a given choice covers $U$?

The probability that an element $u \in A_i$ is $1/2$ (same for $\bar{A}_i$).

The probability that $u$ is covered is $1 - \frac{1}{2^h}$.

The probability that all $u$ are covered is $(1 - \frac{1}{2^h})^s$

The probability that there exists a choice such that all $u$ are covered is at most

$$\binom{t}{h} 2^h \left(1 - \frac{1}{2^h}\right)^s \le (2t)^h e^{-s/2^h} = (2t)^h \cdot e^{-h \ln(4t)} \le \frac{1}{2^h}$$

What is the probability that a given choice covers $U$?

The probability that an element $u \in A_i$ is $1/2$ (same for $\bar{A}_i$).

The probability that $u$ is covered is $1 - \frac{1}{2^h}$.

The probability that all $u$ are covered is $(1 - \frac{1}{2^h})^s$

The probability that there exists a choice such that all $u$ are covered is at most

$$\binom{t}{h} 2^h \left(1 - \frac{1}{2^h}\right)^s \leq (2t)^h e^{-s/2^h} = (2t)^h \cdot e^{-h \ln(4t)} \leq \frac{1}{2^h}$$

What is the probability that a given choice covers $U$?

The probability that an element $u \in A_i$ is $1/2$ (same for $\bar{A}_i$).

The probability that $u$ is covered is $1 - \frac{1}{2^h}$.

The probability that all $u$ are covered is $(1 - \frac{1}{2^h})^s$

The probability that there exists a choice such that all $u$ are covered is at most

$$\binom{t}{h} 2^h \left(1 - \frac{1}{2^h}\right)^s \leq (2t)^h e^{-s/2^h} = (2t)^h \cdot e^{-h \ln(4t)} \leq \frac{1}{2^h}$$

What is the probability that a given choice covers $U$?

The probability that an element $u \in A_i$ is $1/2$ (same for $\bar{A}_i$).

The probability that $u$ is covered is $1 - \frac{1}{2^h}$.

The probability that all $u$ are covered is $(1 - \frac{1}{2^h})^s$

The probability that there exists a choice such that all $u$ are covered is at most

$$\binom{t}{h} 2^h \left(1 - \frac{1}{2^h}\right)^s \le (2t)^h e^{-s/2^h} = (2t)^h \cdot e^{-h \ln(4t)} \le \frac{1}{2^h}$$