Approximation Algorithms

Rolf Niedermeier and Jiong Guo

Lehrstuhl Theoretische Informatik I / Komplexitätstheorie

Institut für Informatik

Friedrich-Schiller-Universität Jena

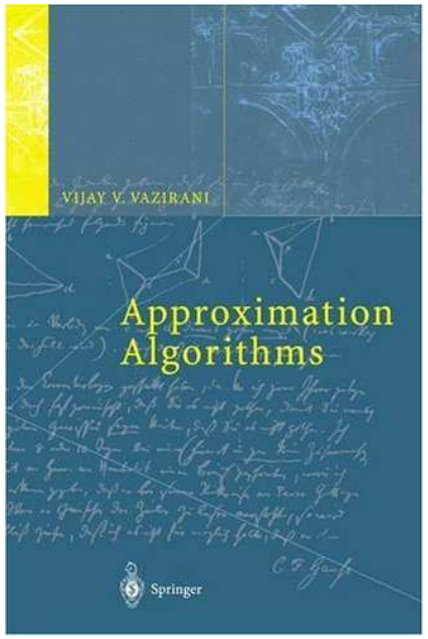
niedermr@minet.uni-jena.de guo@minet.uni-jena.de

http://theinf1.informatik.uni-jena.de/

Approximation Algorithms

Literature:

V. V. Vazirani, *Approximation Algorithms*, Springer-Verlag, 2003.



Lecture is based on this book.

Approximation Algorithms

Further literature:

G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation. Combinatorial Optimization Problems and their Approximability Properties*. Springer-Verlag. 1999.

D. S. Hochbaum (editor). *Approximation Algorithms for NP-Hard Problems*. PWS Publishing. 1997.

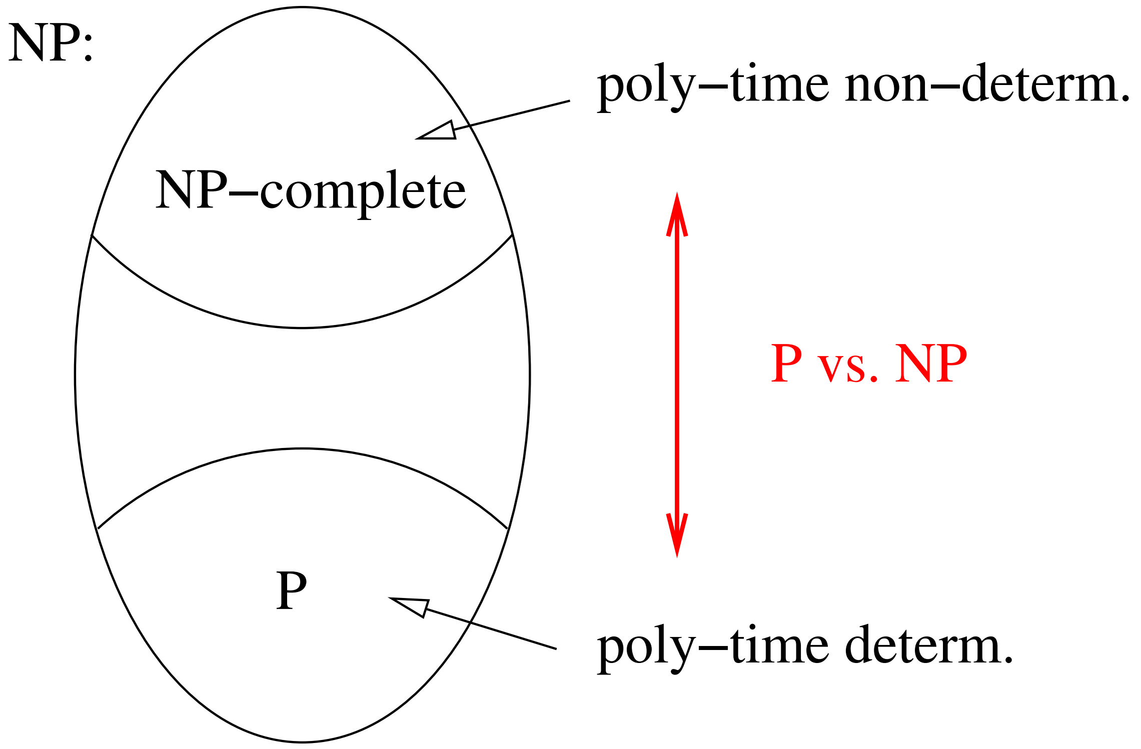
C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley. 1994.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co. 1979.

Approximation Algorithms

Introduction I

Starting position: The following picture illustrates the world of computational complexity (P vs. NP):



Introduction II

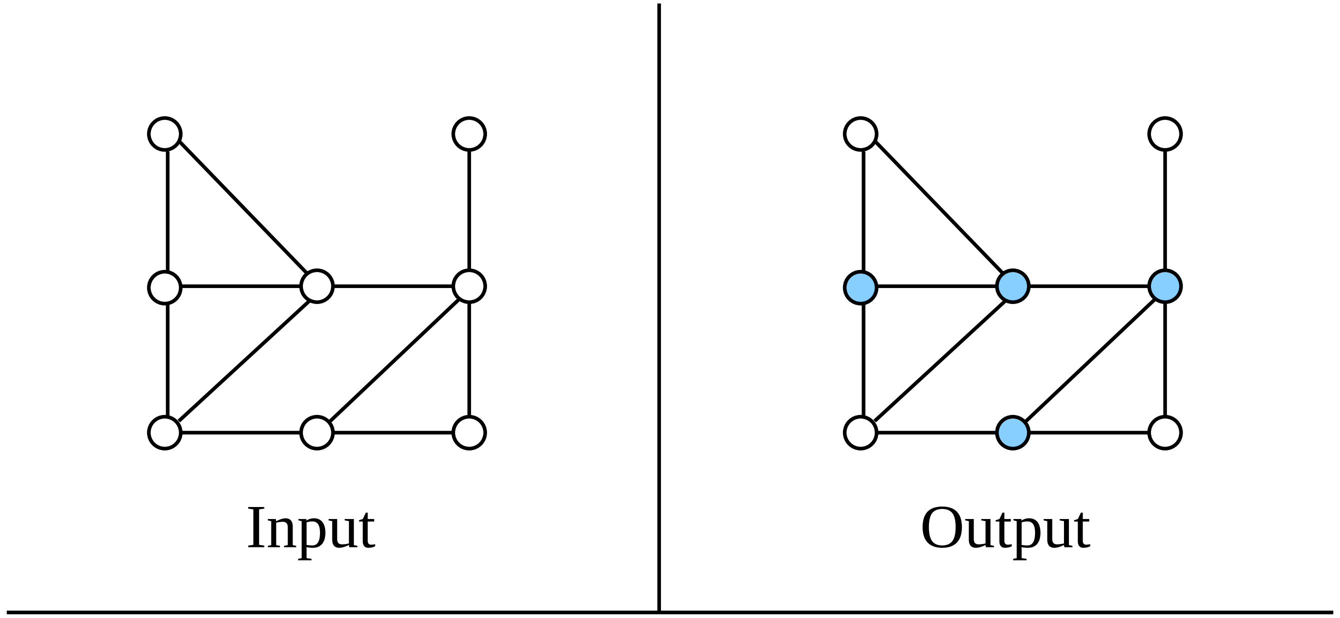
Most natural optimization problems, including those arising in important application areas, are NP-complete. Under the widely believed conjecture that P$\neq$NP their *exact* solution is not computable in polynomial time on a *deterministic* Turing machine (or, equivalently, a RAM).

Possible approaches:

- *Heuristic solving methods*
- *(Exact and) fixed-parameter algorithms*
- *Approximation algorithms*

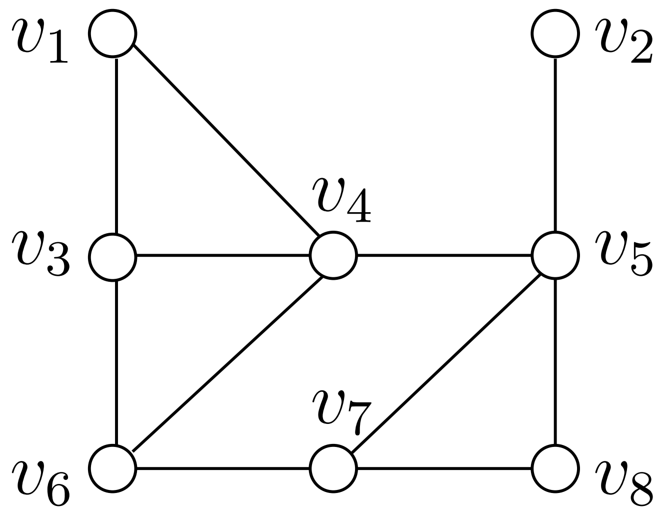
VERTEX COVER—an illustrative example

- Input: An undirected graph $G = (V, E)$.
- Task: Find a minimum cardinality subset of vertices $C \subseteq V$ such that each edge in E has at least one of its endpoints in C.



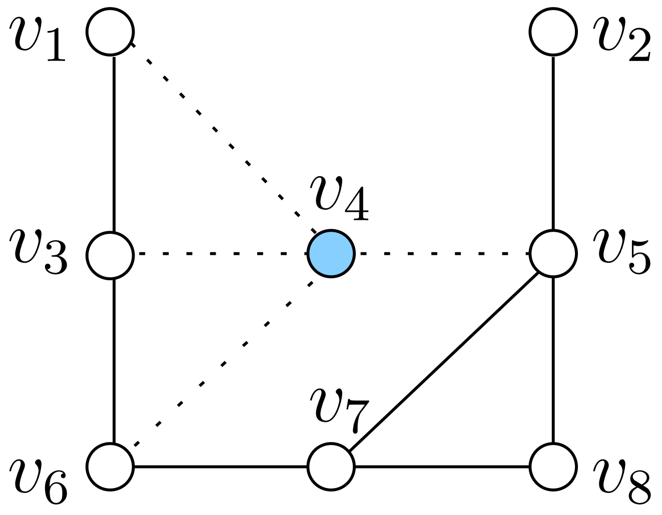
A first approximation algorithm for VERTEX COVER

Idea: Choose highest-degree vertices first ...



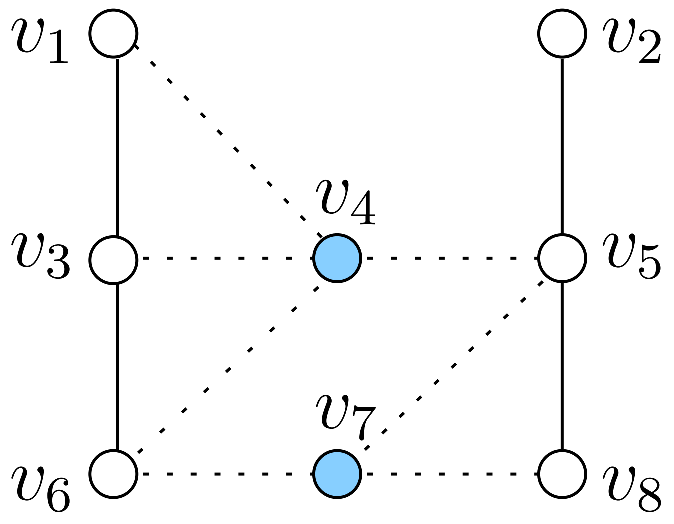
$C = \emptyset$

A first approximation algorithm for VERTEX COVER



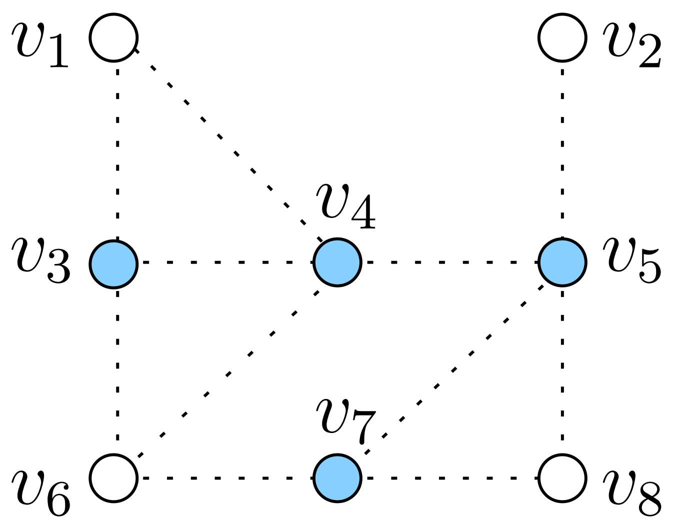
$$C = \{v_4\}$$

A first approximation algorithm for VERTEX COVER



$$C = \{v_4, v_7\}$$

A first approximation algorithm for Vertex Cover



$$C = \{v_4, v_7, v_3, v_5\}$$

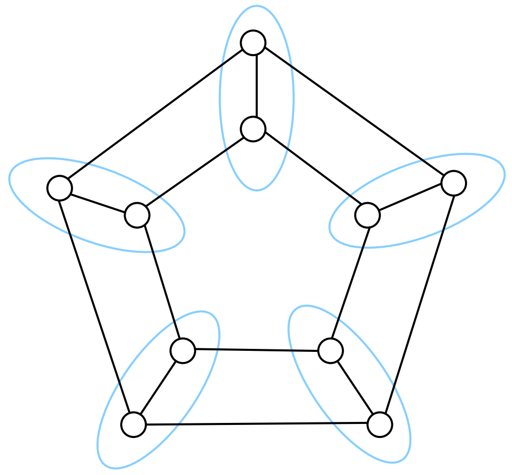
Remark: This algorithm provides an optimal vertex cover for this input graph. For general input graphs, it outputs a vertex cover that can be by a factor of $\ln |V|$ greater than an optimal vertex cover.

A second approximation algorithm for Vertex Cover I

An edge subset M of an undirected graph $G = (V, E)$ is called a *matching* if no two edges in M share an endpoint.

A matching M is *maximal* if there is no matching M' with $M \subsetneq M'$.

Example: (Maximal matching)

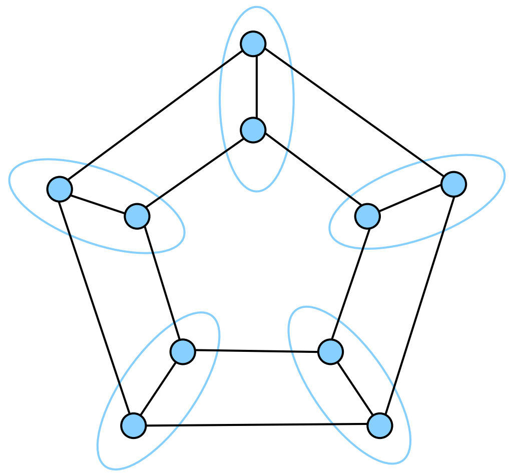


A second approximation algorithm for VERTEX COVER II

A second approximation algorithm for VERTEX COVER:

Idea: Find a maximal matching of G and output the set C of matched vertices.

Example:

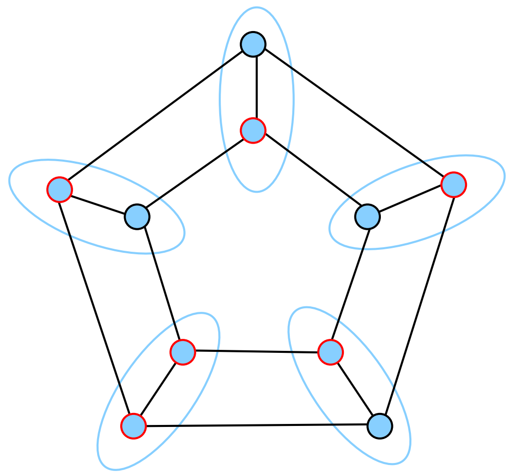


A second approximation algorithm for VERTEX COVER III

Theorem Let C_{opt} be an optimal vertex cover of G and let C be the output of the algorithm. Then, $|C| \leq 2 \cdot |C_{\text{opt}}|$.

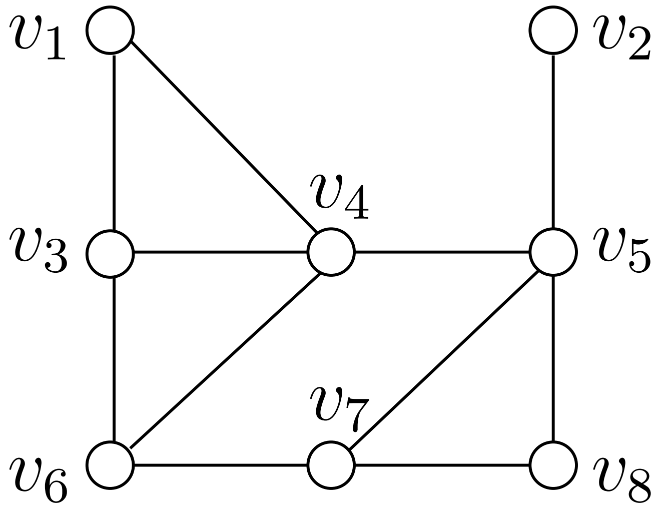
Proof: Blackboard.

Example:



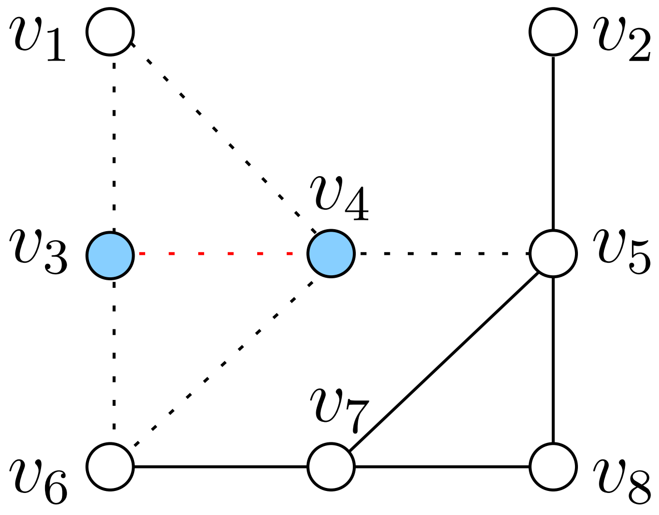
A third approximation algorithm for Vertex Cover

Idea: Iteratively choose both endpoints of arbitrarily chosen edges ...



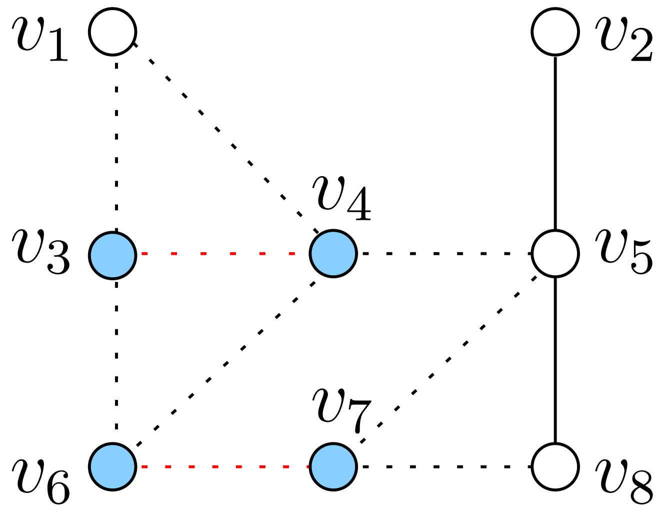
$C = \emptyset$

A third approximation algorithm for VERTEX COVER



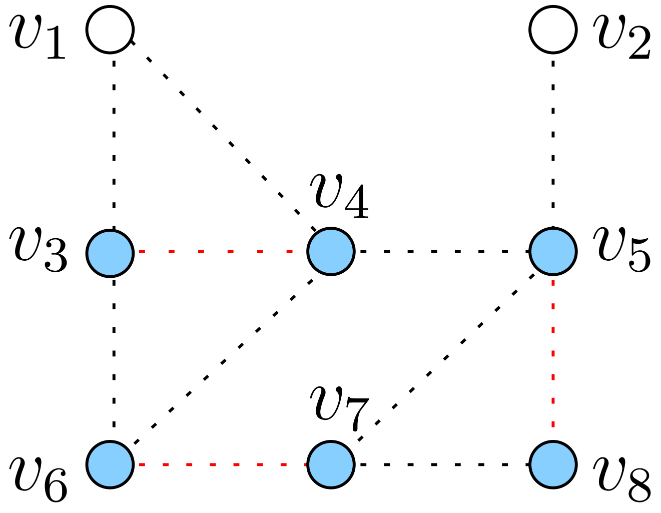
$$C = \{v_3, v_4\}$$

A third approximation algorithm for VERTEX COVER



$$C = \{v_3, v_4, v_6, v_7\}$$

A third approximation algorithm for VERTEX COVER



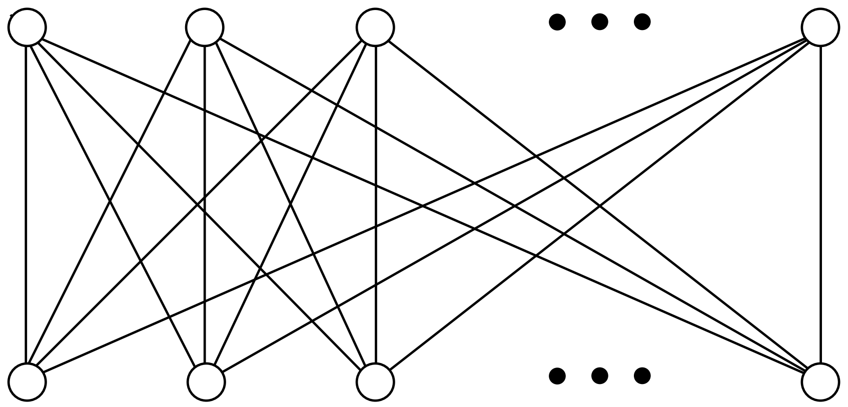
$$C = \{v_3, v_4, v_6, v_7, v_5, v_8\}$$

This algorithm outputs a vertex cover C with $|C| \leq 2 \cdot |C_{\text{opt}}|$ for an optimal vertex cover C_{opt}.

VERTEX COVER—an illustrative example

Remark: It is a currently central open problem in the field of approximation algorithms, whether VERTEX COVER is approximable in polynomial time within a factor $2 - \epsilon$ for a constant $\epsilon > 0$.

A *tight example* for the second and third approximation algorithms: Complete bipartite graph $K_{r,r}$.



MAX2SAT—a second illustrative example

MAXIMUM 2-SATISFIABILITY (MAX2SAT)

- Input: A boolean formula F in conjunctive normal form with at most two literals in each clause.
- Task: Find a truth assignment satisfying a maximum number of clauses.

Input:

$$\begin{aligned} F \quad = \quad & (x) \wedge (y) \wedge (z) \wedge (w) \wedge (\overline{x} \vee \overline{y}) \wedge (\overline{y} \vee \overline{z}) \\ & \wedge (\overline{z} \vee \overline{x}) \wedge (x \vee \overline{w}) \wedge (y \vee \overline{w}) \wedge (z \vee \overline{w}) \end{aligned}$$

Output: $x = y = z = w =$ TRUE.

An approximation algorithm for MAX2SAT

Observation

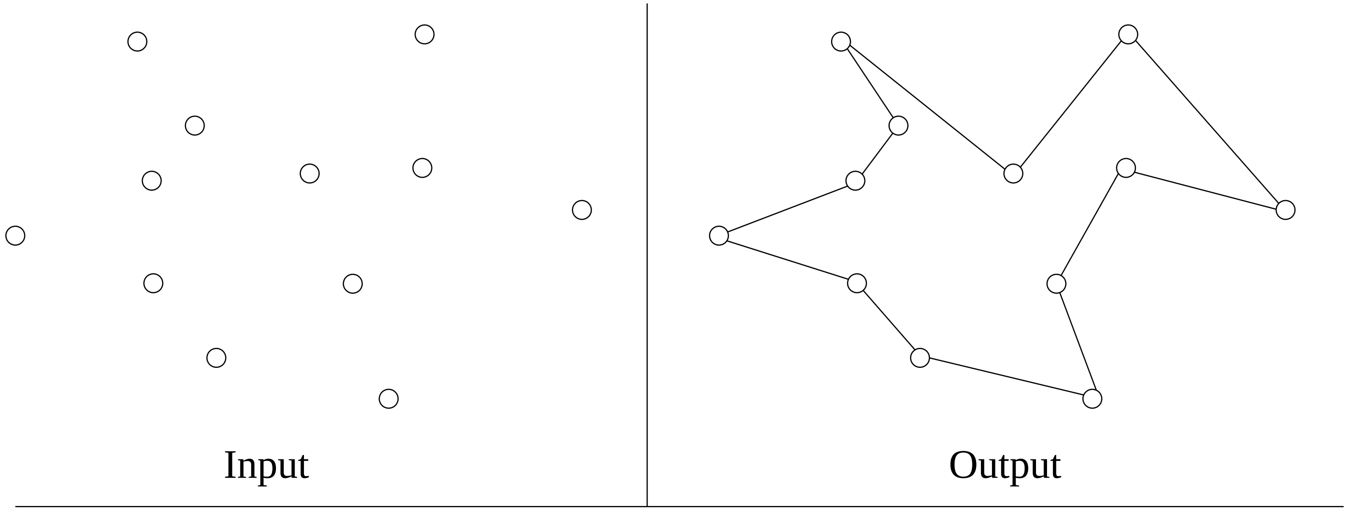
Let F be a boolean formula with m clauses. At least $\lceil m/2 \rceil$ clauses can always be satisfied.

Idea for improved approximation for the case that each clause contains *exactly* two literals: Randomization.

Blackboard.

TRAVELING SALESPERSON—a third illustrative example

- Input: A set $\{1, 2, \ldots, n\}$ of “cities” with pairwise nonnegative distances $d(i, j)$, $1 \leq i, j \leq n$.
- Task: Find an order of the cities such that following this order each city is visited exactly once and the total distance traveled is minimized.



Approximating TRAVELING SALESPERSON

Theorem

Unless P$=$NP, there is no polynomial-time algorithm which gives a "non-trivial approximation" for TRAVELING SALESPERSON.

Proof: Blackboard. (Idea: Reduction from HAMILTON CYCLE ...)

Remark: For special cases there exist "good" approximation algorithms. For example, if the triangle inequality holds, that is, $d(i,k) \leq d(i,j) + d(j,k)$ for all i, j, and k, this problem can be approximated within a factor $3/2$.

Preliminaries: Class NP I

Definition [Classes NP and co-NP]

A language $L \subset \{0,1\}^*$ is in NP if there is a polynomial p and a polynomial-time bounded Turing machine M, called *verifier*, such that for each string $x \in \{0,1\}^*$:

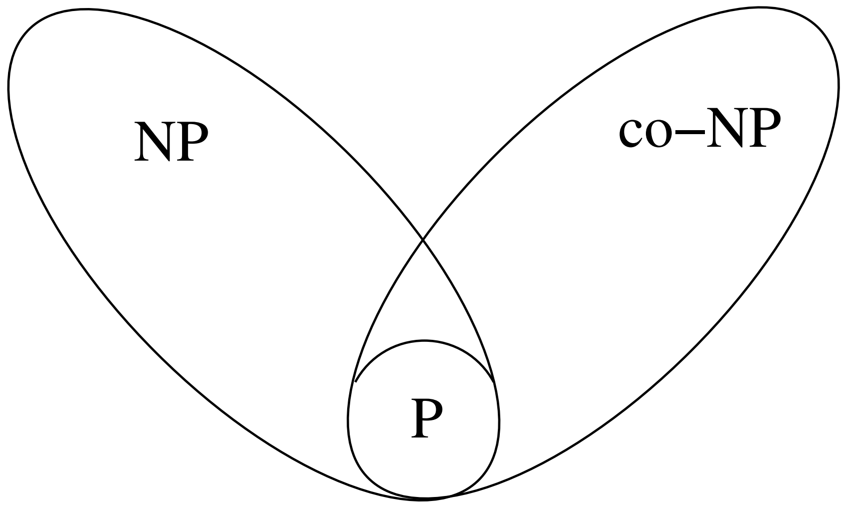
- if $x \in L$, then there is a string y (*Yes certificate*) of polynomially bounded length, that is, $|y| \leq p(|x|)$, such that $M(x,y)$ accepts, and
- if $x \notin L$, then for any string y such that $|y| \leq p(|x|)$, $M(x,y)$ rejects.

A language L belongs to the class co-NP iff $\overline{L} \in$ NP. Thus, co-NP is the class of languages that have *No certificates*.

The class NP defined above is exactly the class of languages that are decidable by non-deterministic polynomial-time Turing machine.

Preliminaries: Class NP II

Any problem in *the class P* trivially has Yes as well as No certificates and, thus, P $\subseteq$ NP $\cap$ co-NP.



Remark: An approximation algorithm provides for a problem in NP an *approximate No certificate*.

Preliminaries: Reductions and NP-completeness

Definition

Let L_1 and L_2 be two languages in NP. We say that L_1 in polynomial time *reduces* to L_2, and write $L_1 \preceq L_2$, if there is a polynomial-time Turing machine which given a string $x \in \{0,1\}^*$, outputs string y such that $x \in L_1$ iff $y \in L_2$.

A language L is *NP-hard* if for every $L' \in$ NP, $L' \preceq L$.

A language L is *NP-complete* if $L \in$ NP and L is NP-hard.

Preliminaries: NP-optimization problems I

Definition

An NP-optimization problem, Π, consists of:

- A set of *valid instances*, D_Π, recognizable in det. polynomial time. The *size* of an instance $I \in D_\Pi$, denoted by $|I|$, is defined as the number of bits needed to write I under the assumption that all numbers occurring in the instance are written in binary.
- Each instance $I \in D_\Pi$ has a set of *feasible solutions*, $S_\Pi(I)$. Every solution $s \in S_\Pi(I)$ is of length polynomially bounded in $|I|$. There is a polynomial-time algorithm that, given a pair (I, s), decides whether $s \in S_\Pi(I)$.
- There is a polynomial-time computable *objective function*, f_Π, that assigns a non-negative rational number to each pair (I, s).
- Finally, Π is specified to be either a *minimization* or a *maximization* problem.

Preliminaries: NP-optimization problems II

An *optimal solution* for an instance of a minimization (maximization) problem is a feasible solution that achieves the smallest (largest) objective function value. We use OPT(I) to denote the objective function value of an optimal solution to instance I.

With every NP-optimization problem, one can naturally associate a decision problem by giving a bound on the optimal solution. Thus, the decision version of an NP-optimization problem Π consists of pairs (I, B) with I being an instance of Π and B is a rational number.

Indeed, hardness for an NP-optimization problem is established by showing that its decision version is NP-hard. In this sense, we also speak of NP-hard optimization problems.

Preliminaries: Approximation algorithms

Definition

Let Π be an optimization problem, an algorithm $\mathcal{A}$ is called an *approximation algorithm* if, for any given instance I of Π, $\mathcal{A}$ returns a feasible solution.

Clearly, this definition is unsatisfactory. In this lecture, we consider only approximation algorithms which provide “near-optimal” solutions, that is, approximation algorithms with guaranteed performance.

Preliminaries: Absolute approximation

Definition

Let Π be a minimization (or maximization) problem and let ϵ be a function, $\epsilon : \mathbb{N}^+ \to \mathbb{Q}^+$. An algorithm $\mathcal{A}$ is said to be an *approximation algorithm with an additive term ϵ* for Π if, on each instance I, $\mathcal{A}$ produces a feasible solution s for I such that $f_\Pi(I, s) - \mathrm{OPT}(I) \leq \epsilon(|I|)$ (or $\mathrm{OPT}(I) - f_\Pi(I, s) \leq \epsilon(|I|)$), and the running time of $\mathcal{A}$ is bounded by a fixed polynomial in $|I|$.

Preliminaries: Relative approximation

Definition

Let Π be a minimization (or maximization) problem and let δ be a function, $\delta : \mathbb{N}^+ \rightarrow \mathbb{Q}^+$ with $\delta \geq 1$ (or $\delta \leq 1$). An algorithm $\mathcal{A}$ is said to be a *factor-δ approximation algorithm* for Π if, on each instance I, $\mathcal{A}$ produces a feasible solution s for I such that $f_\Pi(I, s) \leq \delta(|I|) \cdot \text{OPT}(I)$ (or $f_\Pi(I, s) \geq \delta(|I|) \cdot \text{OPT}(I)$), and the running time of $\mathcal{A}$ is bounded by a fixed polynomial in $|I|$.

Approximation factor δ is also known as *performance ratio*, *approximation guarantee*, etc.

An NP-hard optimization problem Π is called *δ-approximable* if there exists a factor-δ approximation algorithm for Π.

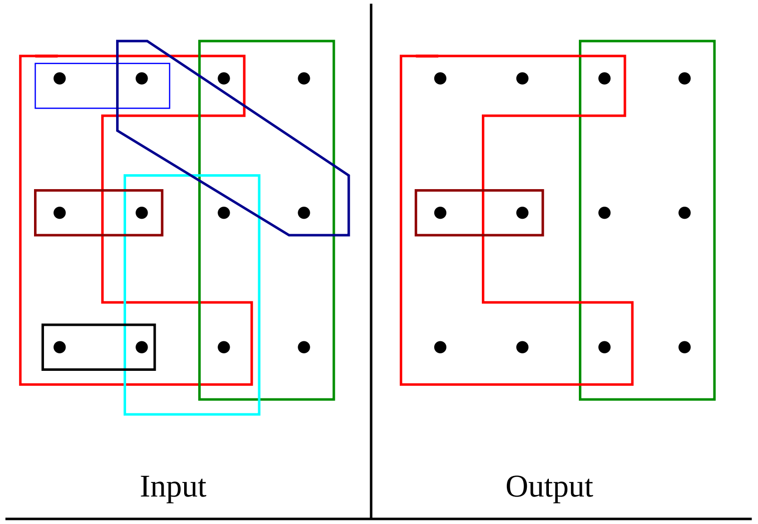
Note: Most approximation algorithms in this lecture provide relative approximations.

Approximation Algorithms

SET COVER I

- Input: A universe U of n elements, a collection of subsets of U, $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$, and a cost function $c : \mathcal{S} \to \mathbb{Q}^+$.
- Task: Find a minimum cost subcollection of $\mathcal{S}$ that covers all elements of U.

Example: $(\forall S \in \mathcal{S}, c(S) = 1)$



Set Cover II

Application: *Airline crew scheduling*.

An airline has m flights which can be combined in “flight legs” such that the same crew can serve all flights in one flight leg. Find a minimum number of crews required to serve all flights.

	Leg 1	Leg 2	Leg 3	Leg 4
Frankfurt – Berlin	1	1	0	0
Frankfurt – Stuttgart	0	0	1	0
Frankfurt – Hamburg	0	0	0	1
Berlin – München	1	0	0	0
Berlin – Stuttgart	0	1	0	0
Stuttgart – Hamburg	0	1	1	0

Two techniques: *Greedy* and *layering*.

SET COVER—Greedy algorithm I

Idea: Iteratively choose the most cost-effective subset.

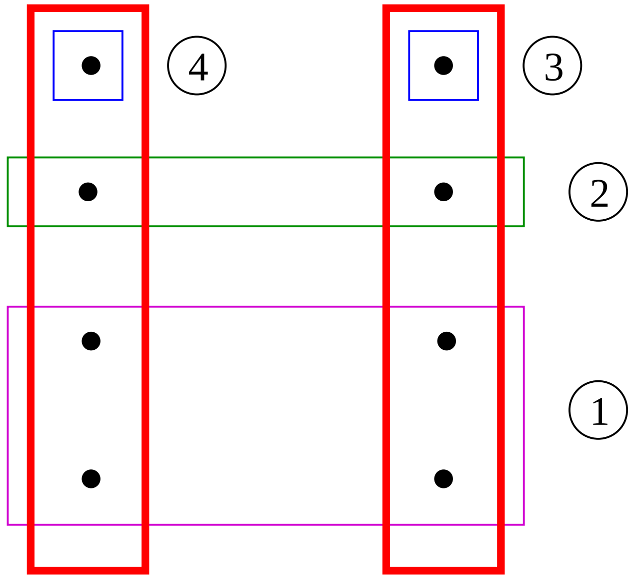
Define the *cost-effectiveness* of a set $S \in \mathcal{S}$ as $c(S)/|S \setminus C|$, where C is the set of already covered elements.

```
Greedy SET COVER Algorithm:

  1. $C \leftarrow \emptyset$.
  2. While $C \neq U$ do
       Pick the set whose cost-effectiveness is smallest, say $S$.
       $C \leftarrow C \cup S$.
  3. Output the picked sets.
```

Set Cover—Greedy algorithm II

Example: The application of the greedy algorithm.



$c(\square) = c(\square) = c(\square) = 1$ and
$c(\square) = 1 + \epsilon$ for a $\epsilon > 0$.

The algorithm outputs four weight-1 subsets, while the optimal solution has weight of $2(1 + \epsilon)$.

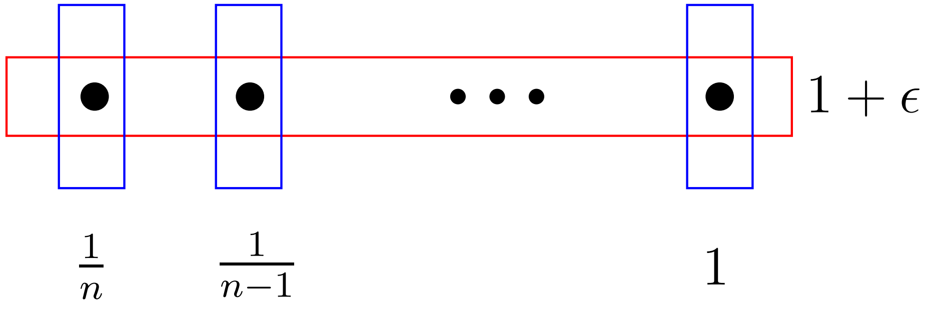
SET COVER—Greedy algorithm III

Theorem

The greedy algorithm is an H_n-factor approximation algorithm for SET COVER, where $H_n := 1 + \frac{1}{2} + \cdots + \frac{1}{n}$.

Proof: Blackboard.

Tight example:



SET COVER—Layering I

Example application to the special case of SET COVER: WEIGHTED VERTEX COVER.

- ☞ Input: An undirected graph $G = (V, E)$ and a weight function $\omega : V \to \mathbb{Q}^+$.
- ☞ Task: Find a minimum-weight vertex cover C.

The layering technique leads to a factor-2 approximation algorithm for WEIGHTED VERTEX COVER which can be generalized to a factor-f approximation algorithm for SET COVER where
$f := \max_{s \in U} |\{S \in \mathcal{S} : s \in S\}|$.

Set Cover—Layering II

Definition

Let $\omega : V \to \mathbb{Q}^+$ be the function assigning weights to vertices of a given graph $G = (V, E)$ and let $\deg_G(v)$ denote the degree of vertex v in G.

We say ω is *degree-weighted* if there is a constant $c > 0$ such that $\omega(v) = c \cdot \deg_G(v)$ for all $v \in V$

For an arbitrary function $\omega : V \to \mathbb{Q}^+$, the *largest degree-weighted function* $t(v)$ is defined as $t(v) = c \cdot \deg_G(v)$ where $c = \min_{v \in V}\{\omega(v)/\deg_G(v)\}$. Define $\omega'(v) = \omega(v) - t(v)$ to be the *residual weight function*.

SET COVER—Layering III

Lemma

Let $\omega : V \to \mathbb{Q}^+$ be a degree-weighted function. Then $\omega(V) \leq 2 \cdot \text{OPT}$.

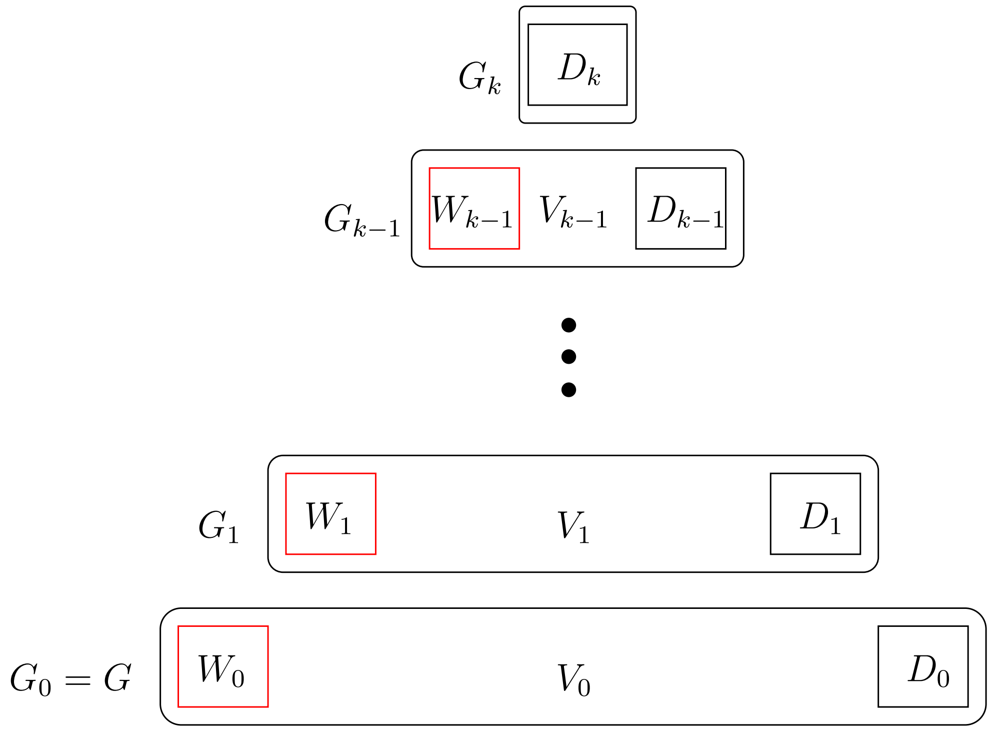
Proof: Let c be the constant such that $\omega(v) = c \cdot \deg_G(v)$, and let U be an optimal vertex cover of G. Since U covers all edges,

$$\sum_{v \in U} \deg_G(v) \geq |E|.$$

Therefore, $\omega(U) = \sum_{v \in U} \omega(v) \geq c \cdot |E|$. Since $\sum_{v \in V} \deg_G(v) = 2 \cdot |E|$, $\omega(V) = 2c \cdot |E|$. The lemma follows.

SET COVER—Layering IV

Idea: Decompose the given weight function into *degree-weighted* functions on a nested sequence of subgraphs of G.



$V_0 = V$

$W_i = \{v \in V_i \mid \omega_i(v) = 0\}$

$D_i = \{v \in V_i \mid \text{deg}_{G_i}(v) = 0\}$

$G_i = G[V_{i-1} \setminus (D_{i-1} \cup W_{i-1})],\ i > 0$

$c_i = \min_{v \in (V_i \setminus D_i)} \{\omega_i(v)/\text{deg}_{G_i}(v)\}$

$t_i(v) = c_i \cdot \text{deg}_{G_i}(v)$

$\omega_i(v) = \omega_{i-1}(v) - t_i(v)$

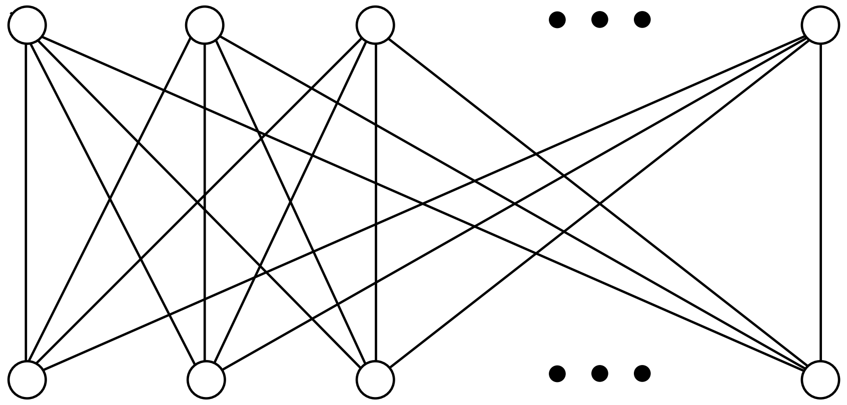
SET COVER—Layering V

Theorem

The layer algorithm achieves an approximation guarantee of factor 2 for WEIGHTED VERTEX COVER, assuming arbitrary vertex weights.

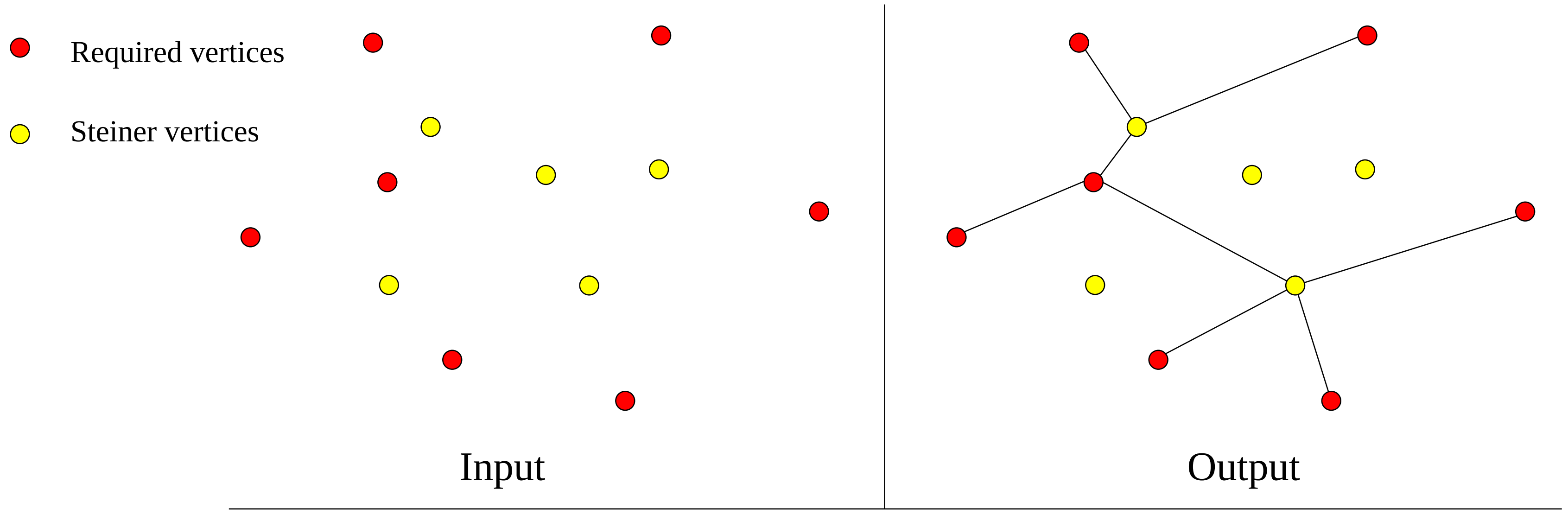
Proof: Blackboard.

Tight example: Complete bipartite graph $K_{n,n}$ with unit vertex weight.



Metric Steiner Tree I

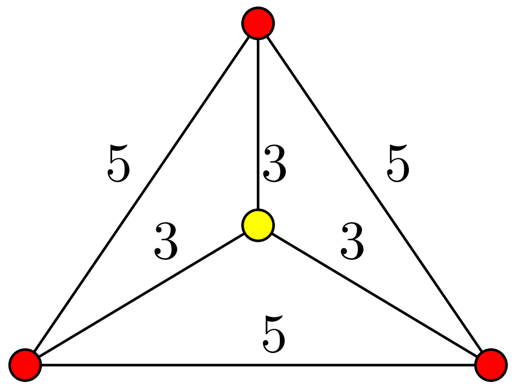
- Input: An undirected, complete graph $G = (V, E)$ whose vertices are partitioned into two sets, *required* and *Steiner*, and an edge cost function $c : E \longrightarrow \mathbb{Q}^+$ which satisfies the *triangle inequality*, that is, $c(\{u,v\}) \leq c(\{u,w\}) + c(\{v,w\})$.
- Task: Find a minimum-cost tree in G that contains all required vertices.



METRIC STEINER TREE II

Clearly, a minimum spanning tree (MST) on the set of required vertices is a feasible solution. Since METRIC STEINER TREE is NP-complete, the MST on the required vertices cannot always be an optimal Steiner tree...

Example:

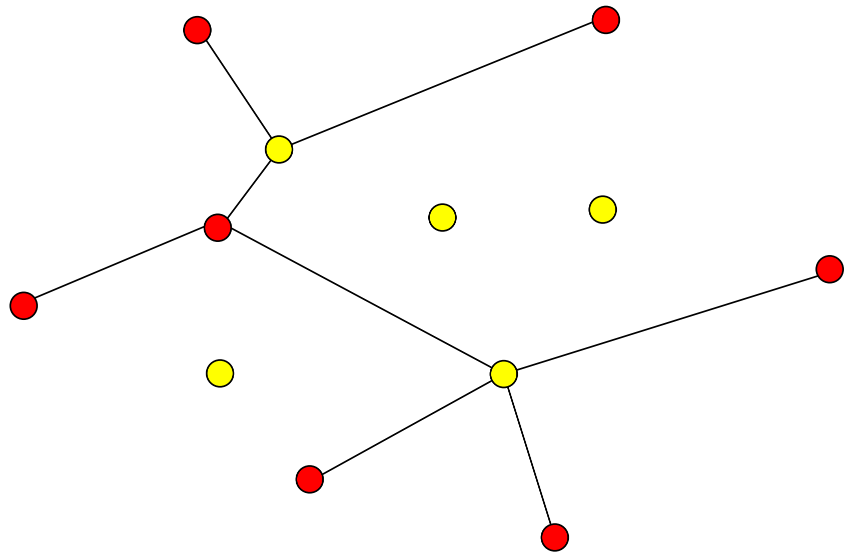


Metric Steiner Tree III

Theorem

The cost of an MST on the required vertices is within $2 \cdot$ OPT.

Idea of proof: Assume that we have an optimal Steiner tree T.

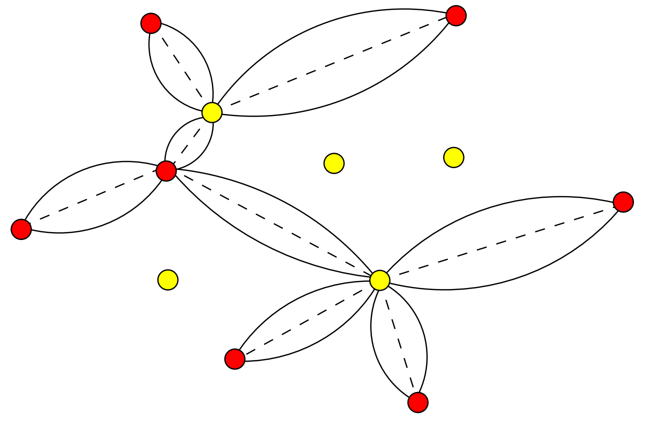


Metric Steiner Tree III

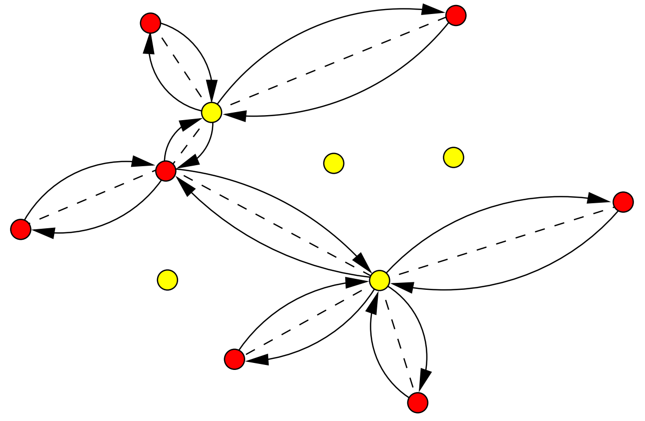
Theorem

The cost of an MST on the required vertices is within $2 \cdot$ OPT.

Idea of proof: Construct an Eulerian graph connecting all vertices in T and find an Euler tour of this graph.



Eulerian graph



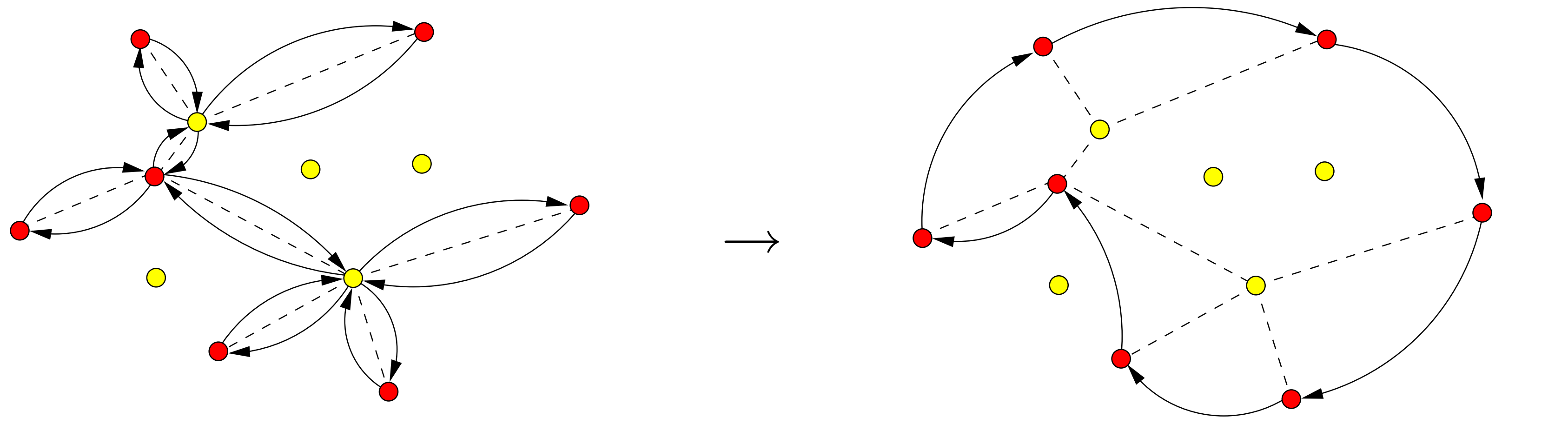
Euler tour

METRIC STEINER TREE III

Theorem

The cost of an MST on the required vertices is within $2 \cdot$ OPT.

Idea of proof: Obtain a Hamiltonian cycle on the required vertices by traversing the Euler tour and "short-cutting" Steiner vertices.

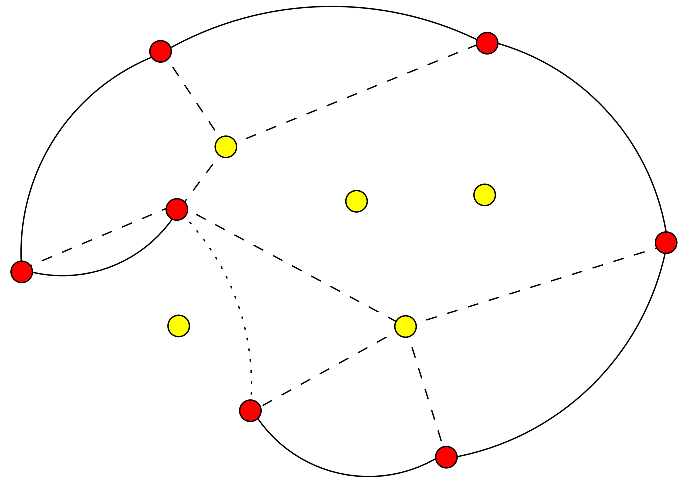


Metric Steiner Tree III

Theorem

The cost of an MST on the required vertices is within $2 \cdot$ OPT.

Proof: Remove an arbitrary edge of the Hamiltonian cycle.



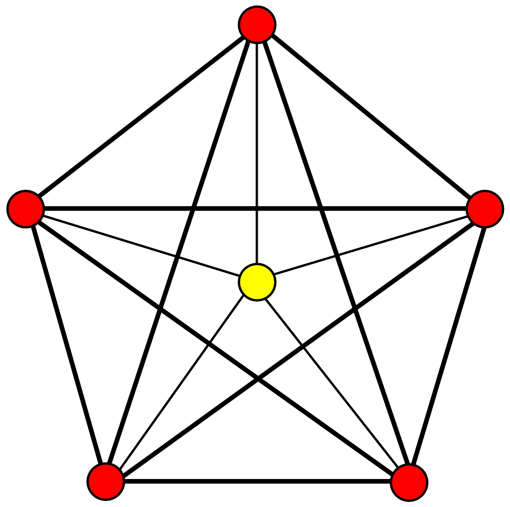
Now we have a spanning tree on all required vertices whose cost is at most $2 \cdot$ OPT.

Metric Steiner Tree IV

The above theorem suggests a factor-2 approximation algorithm for Metric Steiner Tree: Simply find a minimum spanning tree on the required vertices.

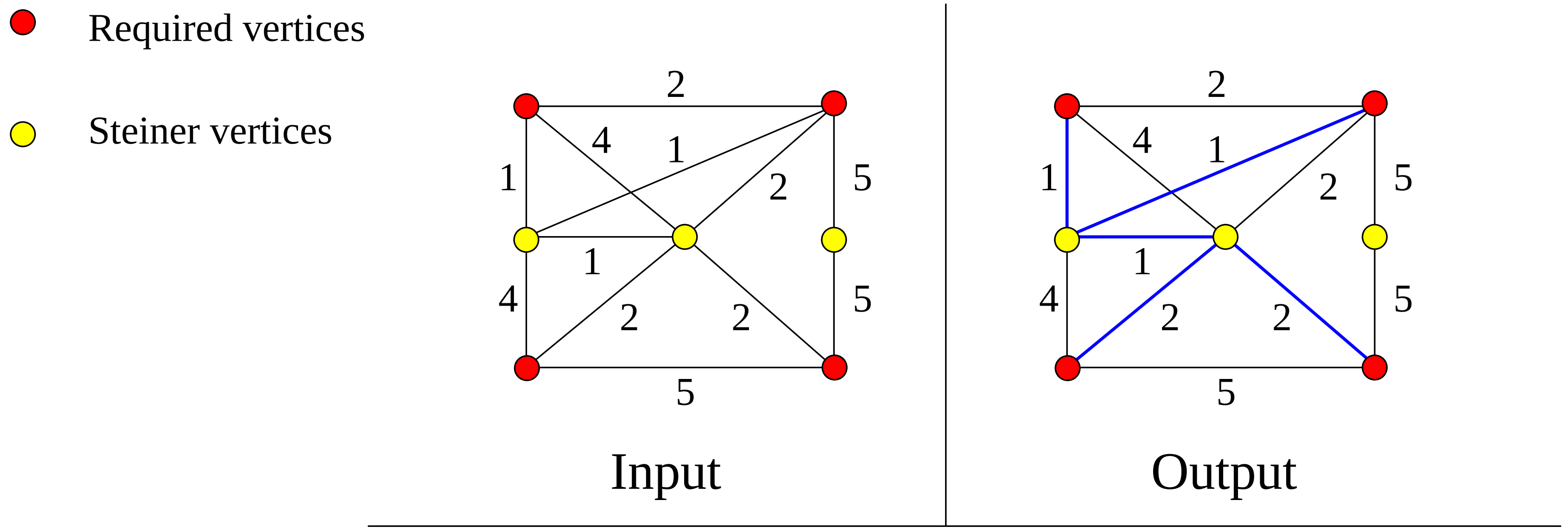
Tight example: A complete graph with n required vertices and one Steiner vertex.

The edges incident to the Steiner vertex have cost 1, while the rest has cost 2.



Steiner Tree I

- Input: An undirected graph $G = (V, E)$ whose vertices are partitioned into two sets, *required* and *Steiner*, and an edge cost function $c : E \to \mathbb{Q}^+$.
- Task: Find a minimum-cost tree in G that contains all the required vertices.



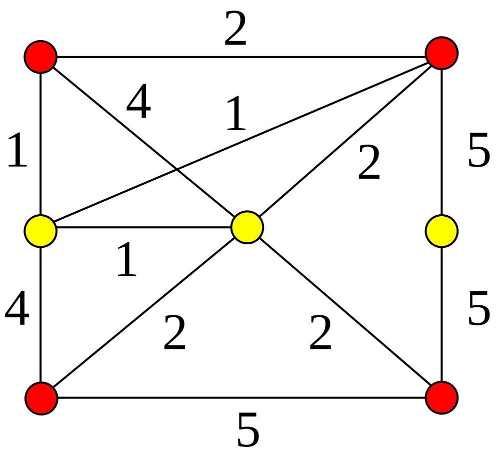
STEINER TREE II

Theorem

STEINER TREE can be approximated in polynomial time within a factor 2.

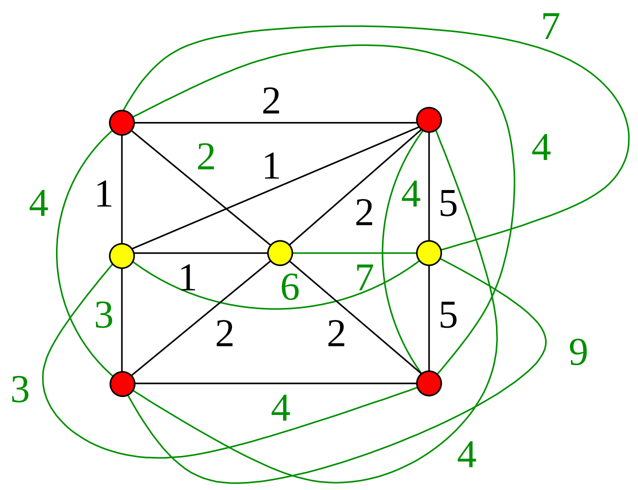
Idea of proof: (1) Transform in polynomial time a STEINER TREE instance into a METRIC STEINER TREE instance by computing the “metric closure” of the original instance: Cost of an edge becomes the cost of a shortest path between the endpoints in the original graph.

STEINER TREE:



$\longrightarrow$

METRIC STEINER TREE:

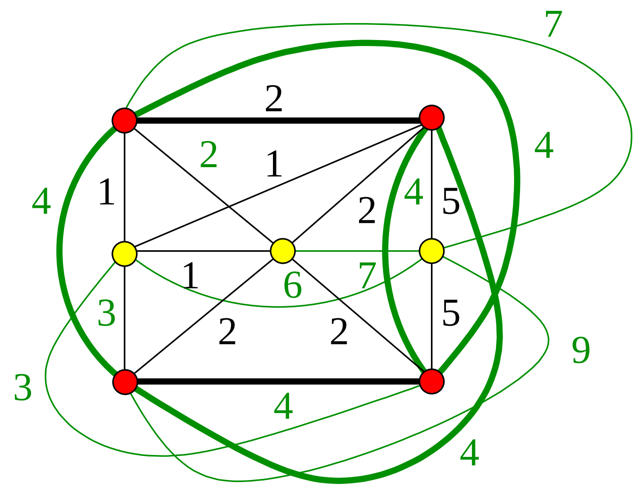


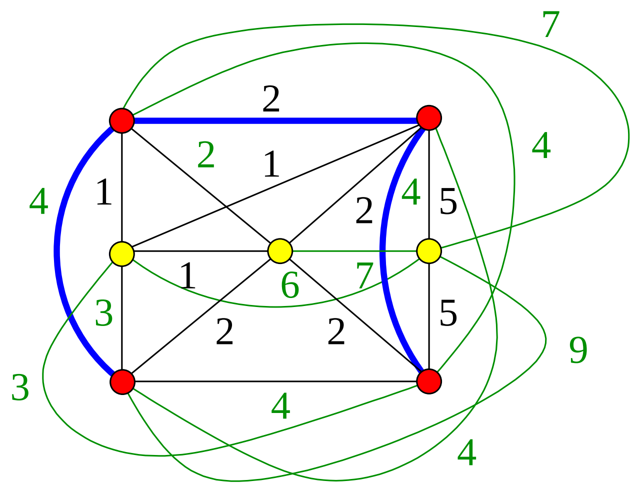
Steiner Tree III

Theorem

Steiner Tree can be approximated in polynomial time within a factor 2.

Idea of proof: (2) Apply the factor-2 approximation for Metric Steiner Tree by finding a minimum spanning tree of the graph induced by the required vertices.





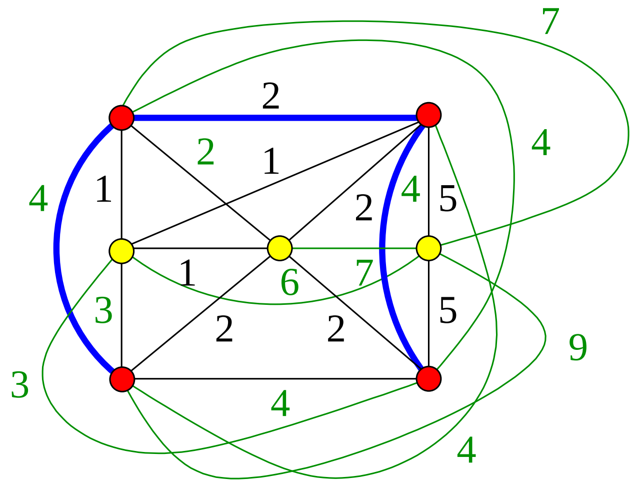
Steiner Tree IV

Theorem

Steiner Tree can be approximated in polynomial time within a factor 2.

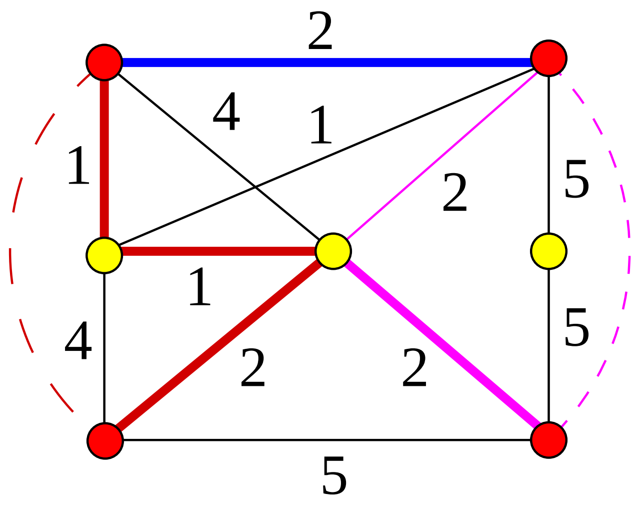
Idea of proof: (3) Finally, transform the resulting metric Steiner tree again into a Steiner tree for the original Steiner Tree instance.

Metric Steiner Tree:



$\longrightarrow$

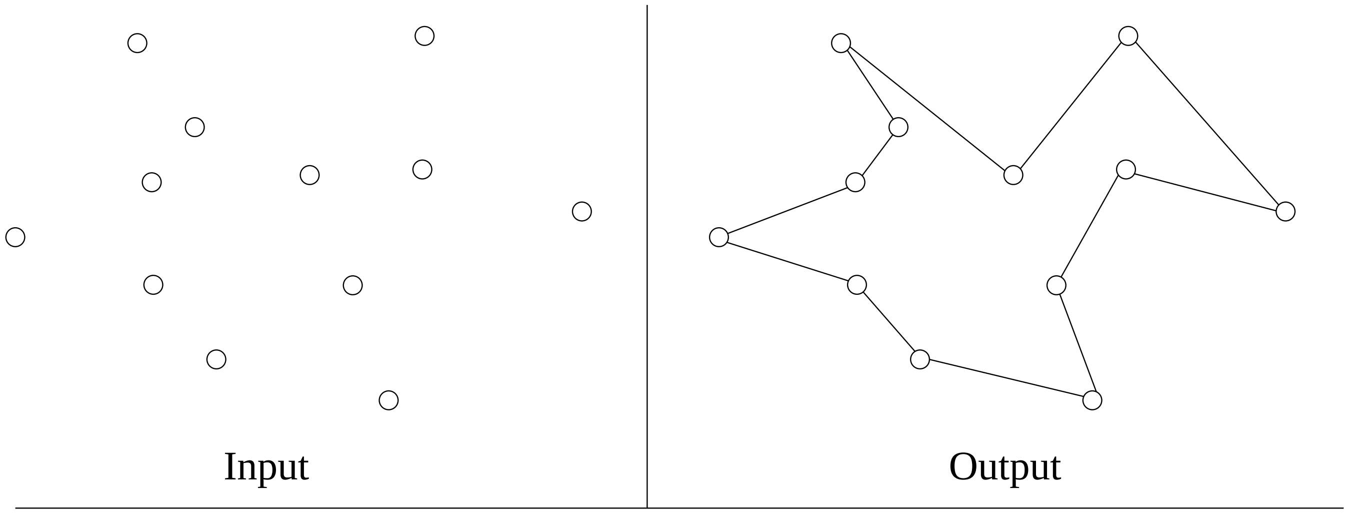
Steiner Tree:



Metric Traveling Salesperson I

☞ Input: An undirected, complete graph $G = (V, E)$ and an edge cost function $c : E \longrightarrow \mathbb{Q}^+$ which satisfies the *triangle inequality*, that is, $c(\{u,v\}) \leq c(\{u,w\}) + c(\{v,w\})$.

☞ Task: Find a minimum-cost cycle in G visiting all vertices exactly once.



METRIC TRAVELING SALESPERSON II

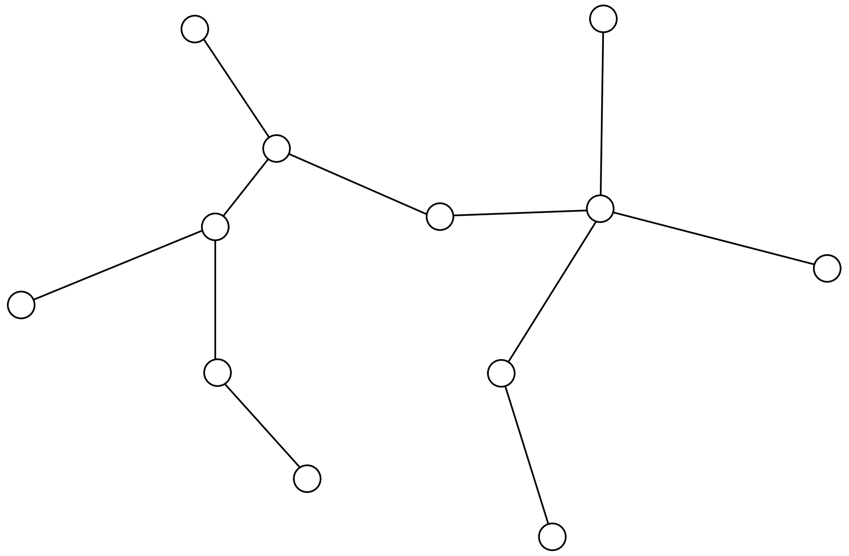
Recall: In its full generality, TRAVELING SALESPERSON cannot be approximated, assuming P$\neq$NP. However, the reduction from HAMILTONIAN CYCLE constructs a graph with an edge cost function which does not satisfy the triangle inequality.

In the *metric* case, there exist good approximation algorithms for TRAVELING SALESPERSON.

Metric TSP—Factor-2 approximation II

A simple factor-2 algorithm:

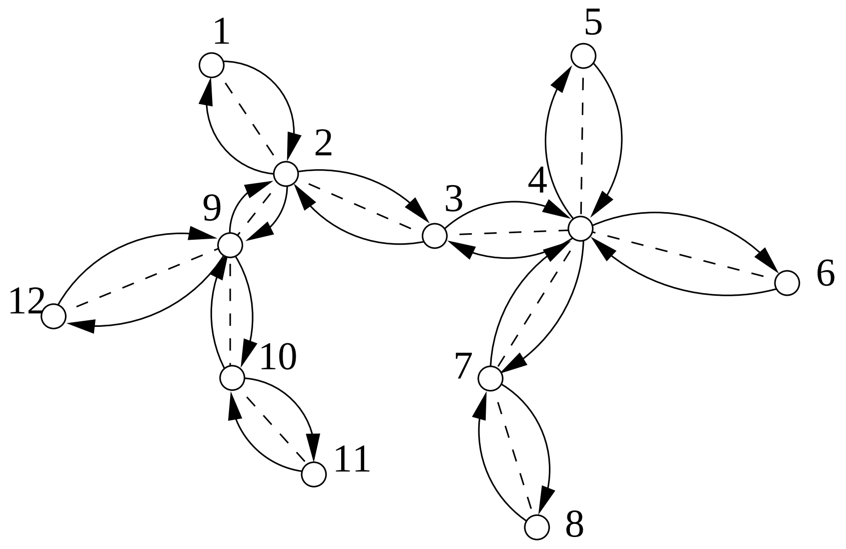
1. Find a minimum spanning tree T of G.



Metric TSP—Factor-2 approximation III

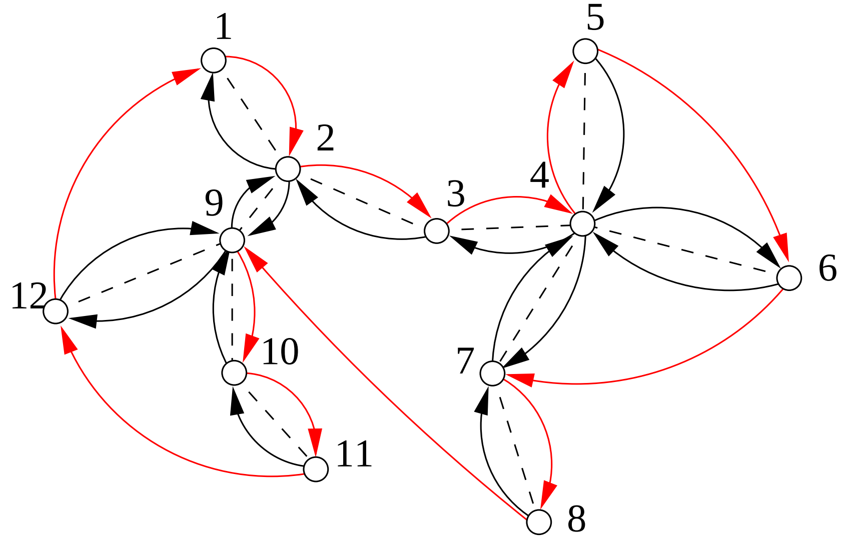
2. Double every edge of T to obtain an Eulerian graph.

3. Find an Euler tour $\mathcal{T}$ of the Eulerian graph.



Metric TSP—Factor-2 approximation IV

4. Output the tour $\mathcal{C}$ that visits vertices of G in the order of their first appearance in $\mathcal{T}$.



Metric TSP—Factor-2 approximation V

Theorem

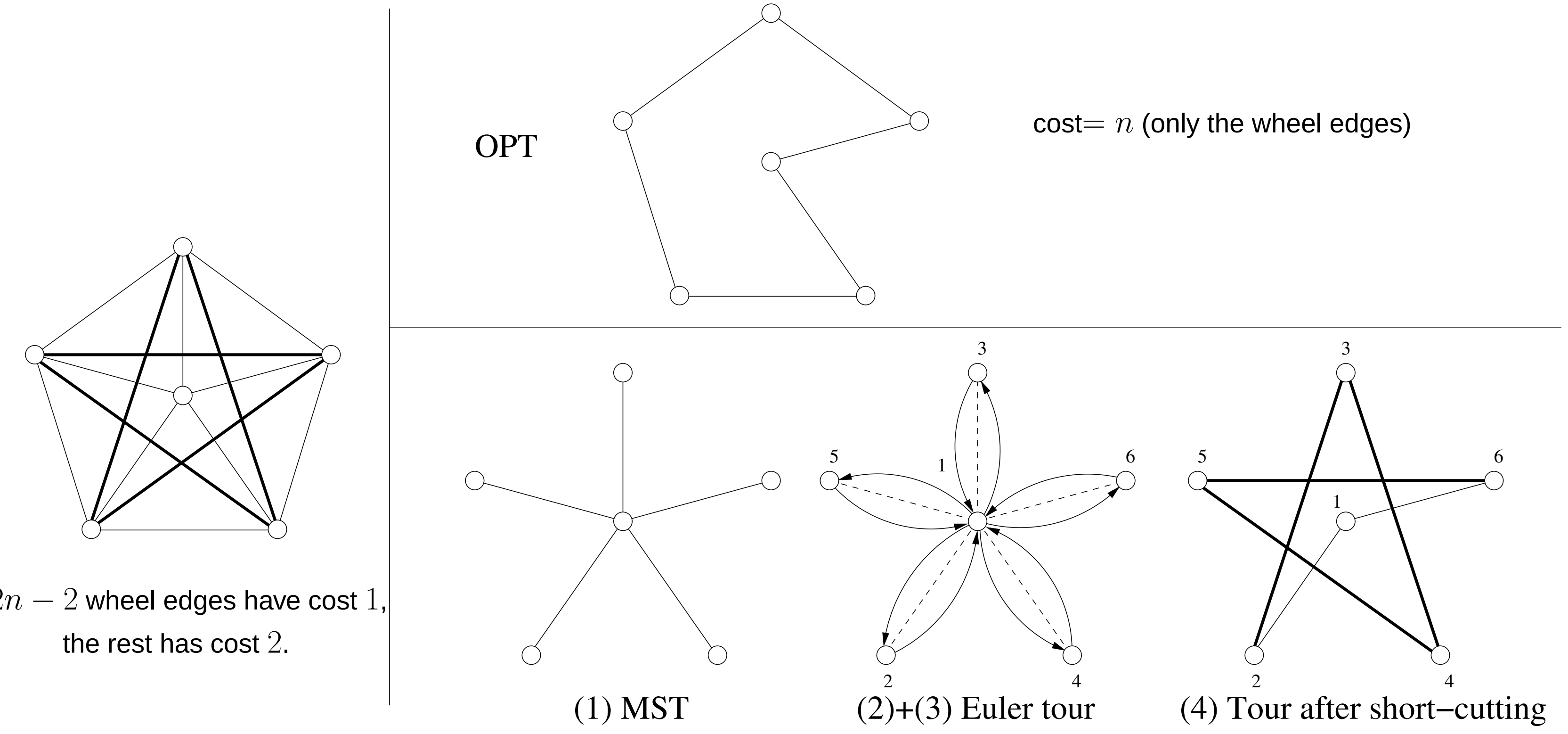
Metric Traveling Salesperson can be approximated with a factor 2.

Proof: Clearly, $c(T) \leq$ OPT. Since the Euler tour $\mathcal{T}$ contains each edge of T twice, $c(\mathcal{T}) = 2 \cdot c(T)$. Because of triangle inequality, after the "short-cutting", $c(\mathcal{C}) \leq c(\mathcal{T})$. Thus, $c(\mathcal{C}) \leq 2 \cdot$ OPT.

METRIC TSP—Factor-2 approximation VI

Tight example: A complete graph on n vertices, K_n, with edge costs 1 or 2.

OPT $= n$ whereas the algorithm delivers tour with cost $2n - 2$.

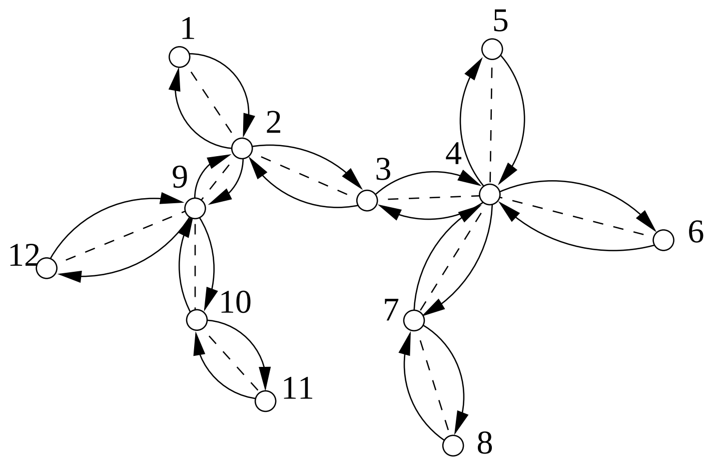


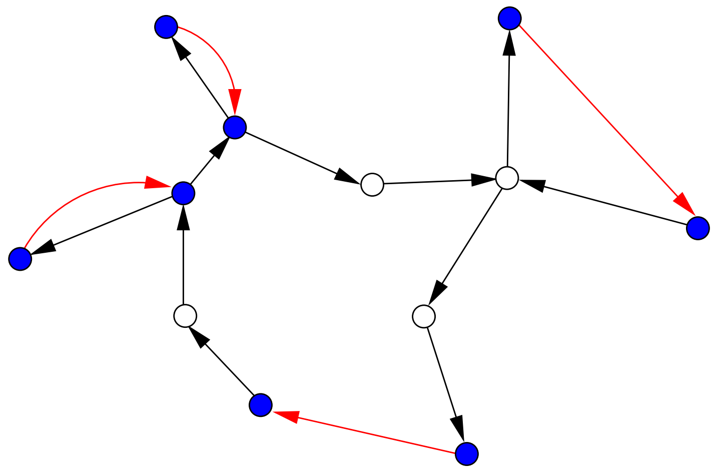
cost$= n$ (only the wheel edges)

$2n - 2$ wheel edges have cost 1, the rest has cost 2.

Metric TSP—Factor-$3/2$ approximation I

Idea: Find a cheaper Euler tour than that found by doubling a minimum spanning tree.





Minimum cost perfect matching on the vertices that have odd degree in the minimum spanning tree.

METRIC TSP—Factor-$3/2$ approximation II

The factor-$3/2$ algorithm for METRIC TSP:

1. Find a minimum spanning tree T of G.
2. Compute a minimum cost perfect matching M on the set of odd-degree vertices of T. Add M to T and obtain an Eulerian graph.
3. Find an Euler tour $\mathcal{T}$ of the Eulerian graph.
4. Output the tour that visits vertices of G in the order of their first appearance in $\mathcal{T}$.

METRIC TSP—Factor-$3/2$ approximation III

Lemma

$c(M) \leq$ OPT$/2$.

Proof: Let τ denote an optimal TSP tour of G and let τ' be the tour on V_o obtained by short-cutting τ. By the triangle inequality, $c(\tau') \leq c(\tau) =$ OPT. Now, τ' is the union of two perfect matchings on V_o, each consisting of alternate edges of τ'. Thus, one of these two matchings has a cost at most $c(\tau')/2 \leq$ OPT$/2$.

METRIC TSP—Factor-$3/2$ approximation III

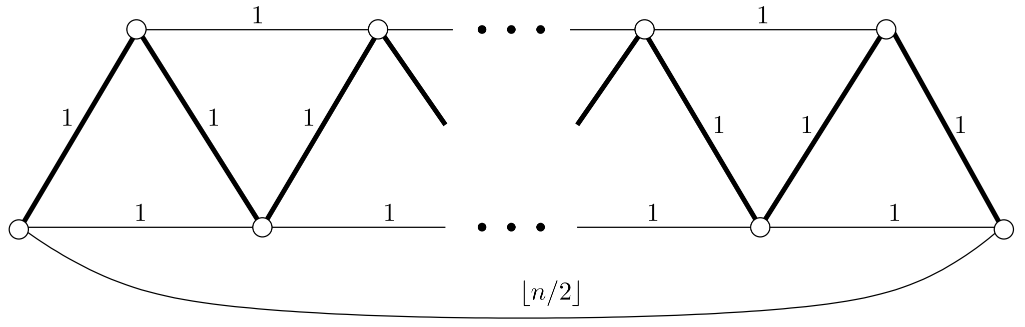
Theorem

The above algorithm achieves an approximation guarantee of $3/2$ for METRIC TRAVELING SALESPERSON.

Proof: The cost of the Euler tour is upper-bounded as follows:

$$c(\mathcal{T}) = c(T) + c(M) \leq \text{OPT} + \frac{1}{2}\text{OPT} = \frac{3}{2}\text{OPT}.$$

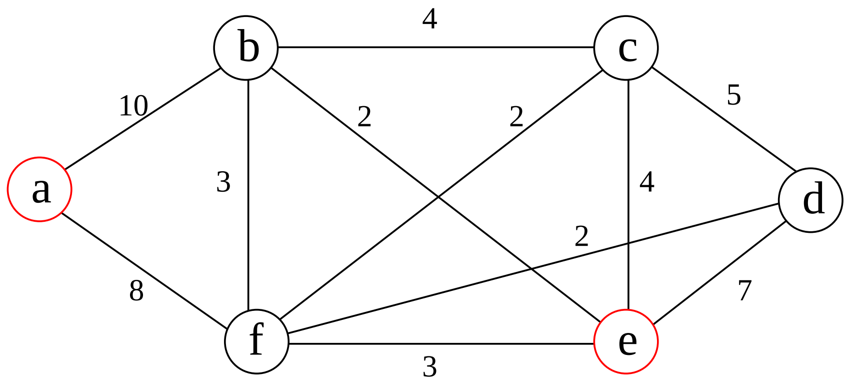
Tight example: The following graph has n vertices, with n odd. Thick edges represent the MST found in the first step.



Multiway Cut and k-Cut I

Definition

Given a connected, undirected graph $G = (V, E)$ with an assignment of weights to edges, $\omega : E \to \mathbb{R}^+$, a *cut* is defined by a partition of V into two sets, say V' and $V \setminus V'$, and consists of all edges that have one endpoint in each partition. Given two *terminals* $s, t \in V$, the cut defined by a partition that separates s and t is called an *s–t cut*.

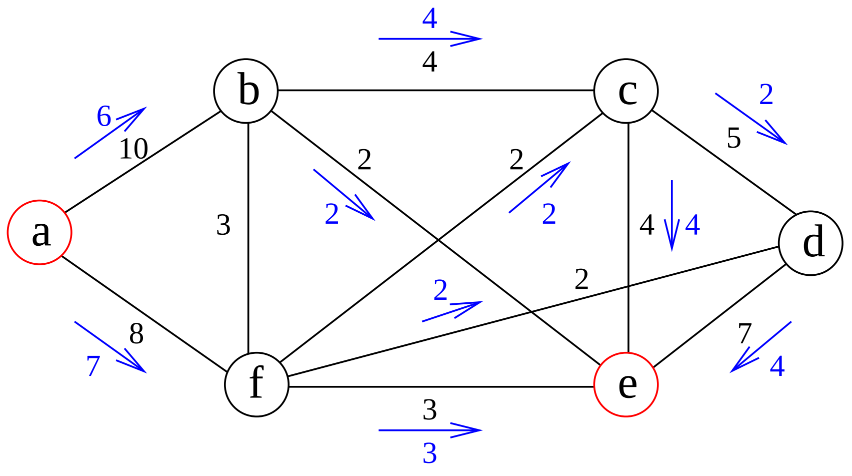


Quiz: The minimal cost of an a–e cut is: (a) 16? (b) 15? (c) 13? (d) 12?

Multiway Cut and k-Cut II

Excursion: Maximum Flow

- Input: A connected, undirected graph $G = (V, E)$ with an edge capacity function $c : E \to \mathbb{R}^+$ and two distinct vertices s (source) and t (sink or target).
- Task: Find the maximum flow that can be routed from s to t with respect to the given edge capacities.



The maximum amount of an a–e flow is 13.

MULTIWAY CUT and k-CUT III

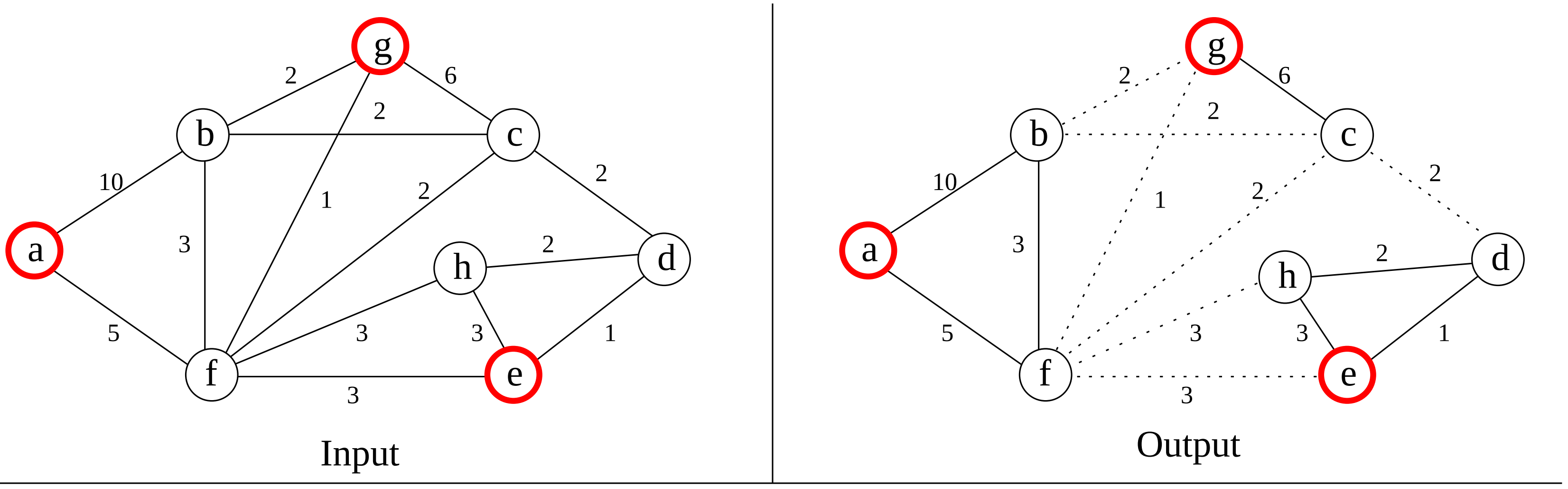
It can be shown that

- MAXIMUM FLOW can be solved in polynomial time, and
- the maximal amount of a flow between two vertices u and v is equal to the minimal cost of an u–v cut (MAX-FLOW MIN-CUT Theorem).

Multiway Cut I

- Input: A connected, undirected graph $G = (V, E)$ with an edge weight function $\omega : E \to \mathbb{R}^+$ and a set of terminals $S = \{s_1, s_2, \ldots, s_k\} \subseteq V$.
- Task: Find a minimum weight set of edges (*minimum multiway cut*) whose removal disconnects the terminals from each other.

Example: $S = \{$a, e, g$\}$.



Multiway Cut II

Remark: Multiway Cut with $k = 2$ is precisely the minimum s–t cut problem and, thus, is solvable in polynomial time using a maximum flow algorithm. For any fixed $k \geq 3$ Multiway Cut is NP-hard.

Applications: Multiprocessor scheduling, VLSI design...

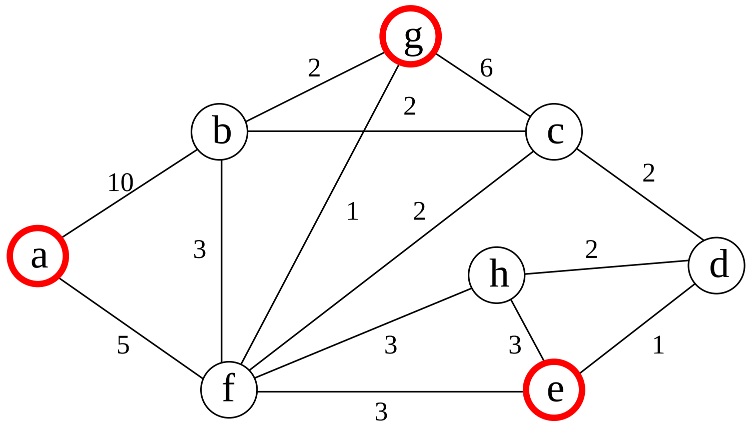
Definition

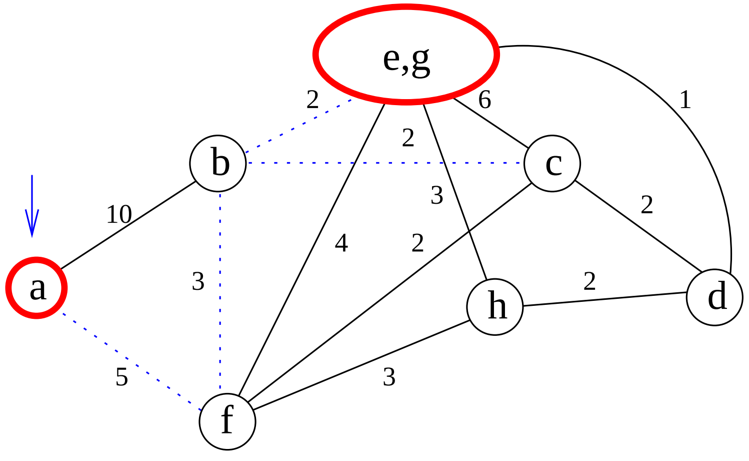
An *isolating cut* for a terminal s is defined as a set of edges whose removal disconnects s from the rest of the terminals.

MULTIWAY CUT III

MULTIWAY CUT heuristic:

1. Using MAXIMUM FLOW, for each $i = 1, \ldots, k$, compute a minimum weight isolating cut for s_i, say C_i.



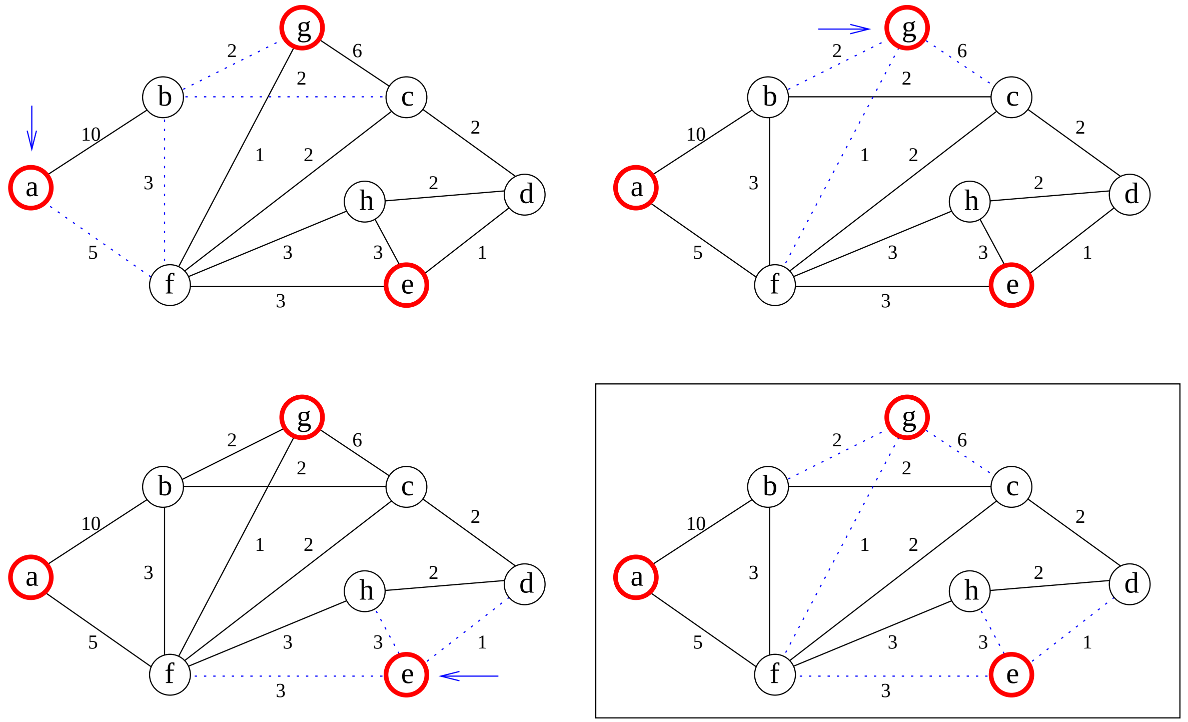


Multiway Cut III

Multiway Cut heuristic:

1. Using Maximum Flow, for each $i = 1, \ldots, k$, compute a minimum weight isolating cut for s_i, say C_i.

2. Discard the heaviest of these cuts and output the union of the rest.



MULTIWAY CUT IV

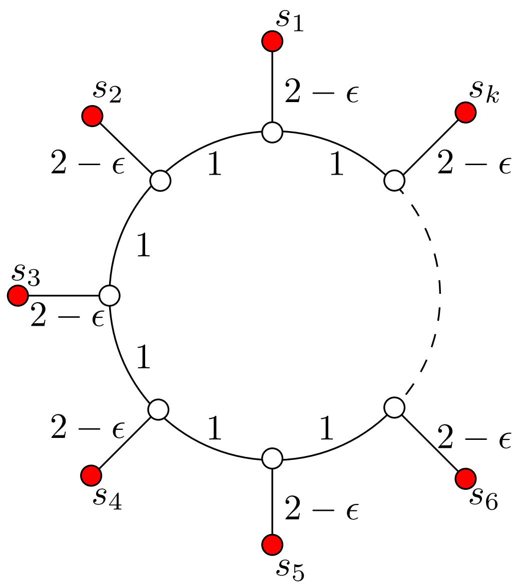
Theorem

The MULTIWAY CUT heuristic achieves an approximation factor of $2 - 2/k$.

Proof: Blackboard.

Tight example:

A graph on $2k$ vertices consisting of a k-cycle and a distinct terminal attached to each vertex of the cycle. The cycle edges have weight 1 and edges attaching terminals to the cycle have weight $2 - \epsilon$ for any $1 > \epsilon > 0$.

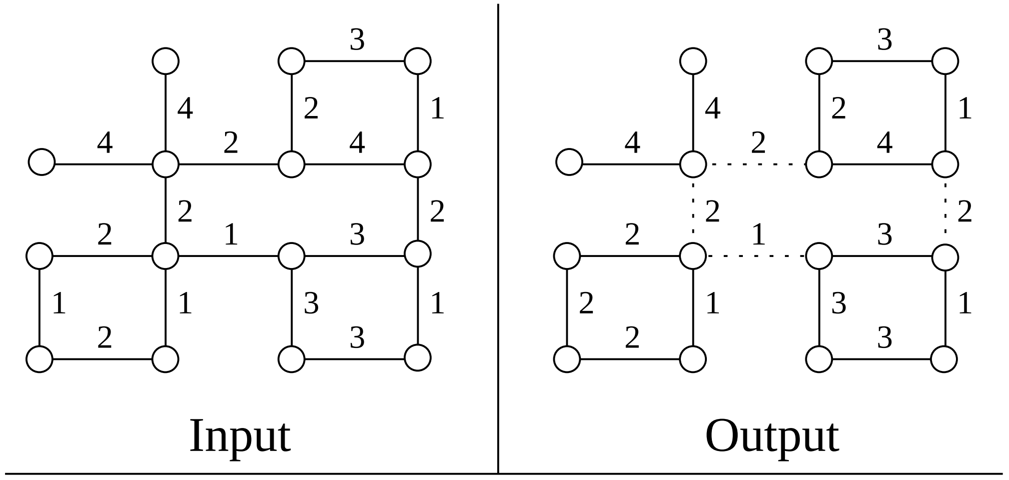


MINIMUM k-CUT I

☞ Input: A connected, undirected graph $G = (V, E)$ with an edge weight function $\omega : E \to \mathbb{R}^+$ and an integer $k > 1$.

☞ Task: Find a minimum weight set of edges (*minimum k-cut*) whose removal leaves k connected components.

Example: $k = 4$.



Minimum k-Cut II

Remark: Minimum k-Cut is in polynomial time solvable for *fixed* k; it is NP-hard if k is specified as part of the input.

Heuristic I:

```
    While $G$ has less than $k$ connected components do
        Compute a minimum cut in each connected component and generate
        an additional connected component by removing the edges of the lightest
        cut;
```

This heuristic is clearly correct and achieves a factor-$(2 - 2/k)$ approximation; however, the proof is quite involved.

We use the *Gomory-Hu tree representation of minimum cuts* to give a simpler algorithm achieving the same approximation factor.

Minimum k-Cut III

Definition

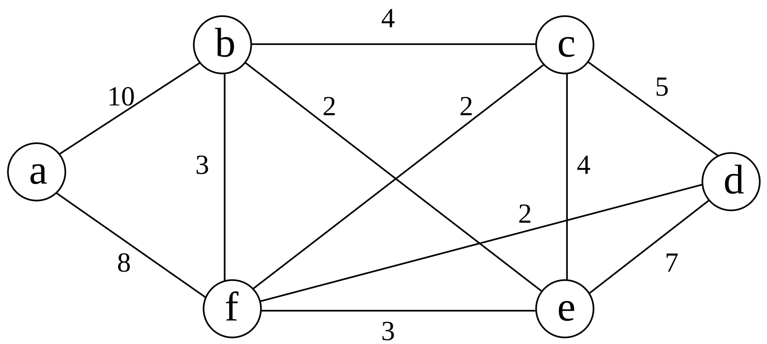
Let $G = (V, E)$ be an undirected and edge-weighted graph. A tree $T = (V, E')$ with an edge cost function $\omega' : E' \longrightarrow \mathbb{R}^+$ is said to be a *Gomory-Hu tree* of G if

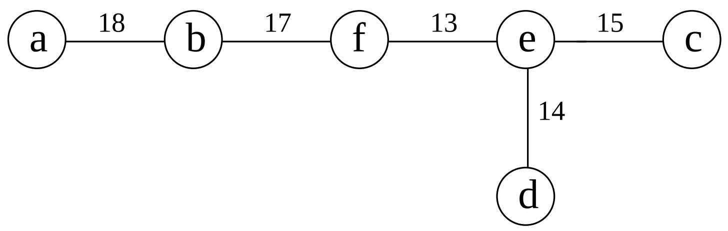
1. for each pair of vertices $u, v \in V$, the cost of a minimum u–v cut in G is the same as that in T, and
2. for each edge $e \in E'$, $\omega'(e)$ is the cost of the cut associated with e in G, that is, the cut in G defined by the partition $(S, \overline{S})$ where S and $\overline{S}$ are the vertex sets of the two trees resulting by removing e from T.

Informally, a Gomory-Hu tree encodes, in a compact manner, minimum u–v cuts in G for each pair of vertices $u, v \in V$: A minimum u–v cut in G is given by a minimum weight edge on the unique path between u and v in T.

Minimum k-Cut IV

An example of an edge-weighted graph and an associated Gomory-Hu tree:





Remark: For an arbitrary undirected and edge-weighted graph G, a Gomory-Hu tree of G can be constructed in polynomial time.

MINIMUM k-CUT V

A useful lemma:

Lemma

Let $S := \cup_{i=1}^{l} C_i$ where $C_{j_1} \neq C_{j_2}$ for $j_1 \neq j_2$ and C_i is a cut in G associated with an edge in T. Then, the removal of S from G leaves a graph with at least $l + 1$ connected components.

Proof: Removing the l edges from T that correspond to S leaves exactly $l + 1$ connected components in T, say with vertex sets $V_1, V_2, \ldots, V_{l+1}$. Clearly, removing S from G will disconnect each pair V_i and V_j. Hence we must get at least $l + 1$ connected components.

Minimum k-Cut VI

Heuristic II:

1. Compute a Gomory-Hu tree T for G.
2. Output the union C of the $k-1$ cuts in G which are associated with the lightest $k-1$ edges of T.

Remark: If the removal of C from G leaves more than k connected components, then reinsert some of the removed edges until there are exactly k components.

Minimum k-Cut VII

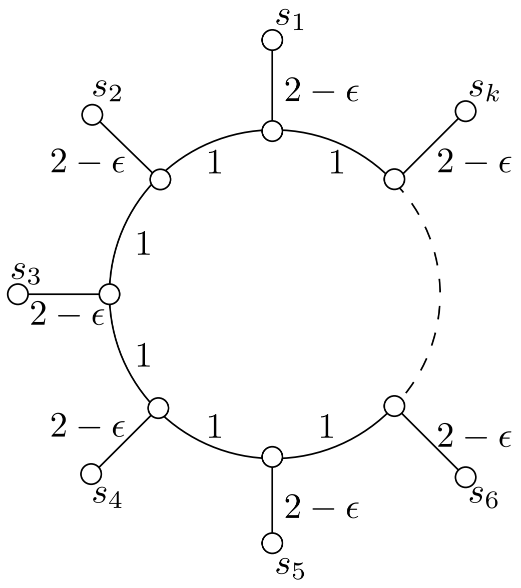
Theorem

Heuristic II achieves an approximation factor of $2 - 2/k$.

Proof: Blackboard.

Tight example:

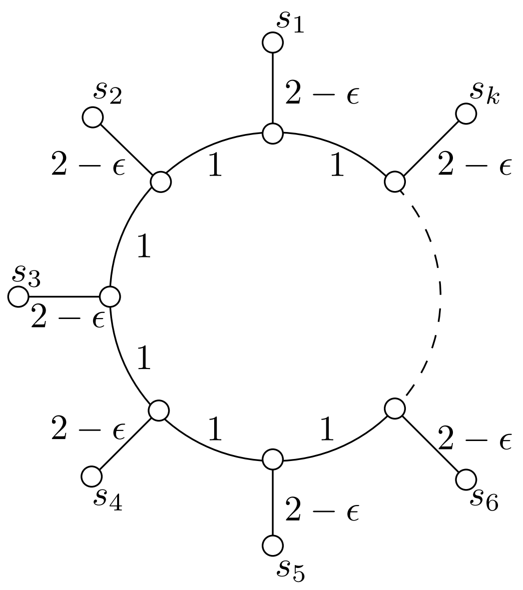
A graph G on $2k$ vertices consisting of a k-cycle and a distinct vertex attached to each vertex of the cycle. The cycle edges have weight 1 and edges attaching vertices to the cycle have weight $2 - \epsilon$ for any $1 > \epsilon > 0$.



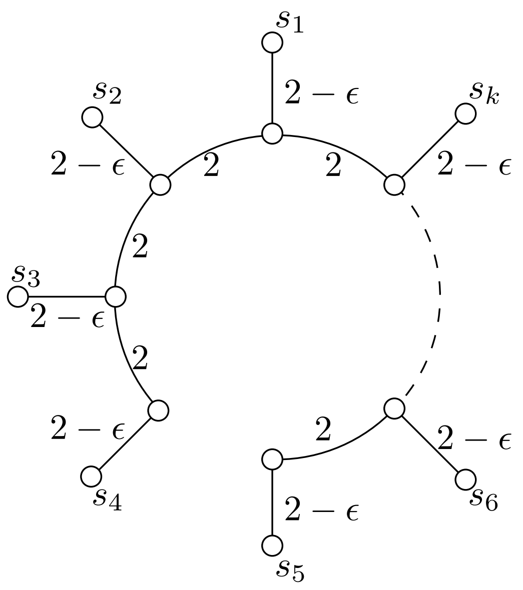
MINIMUM k-CUT VIII

Tight example: (continued)

Graph G



Gomory-Hu tree of G

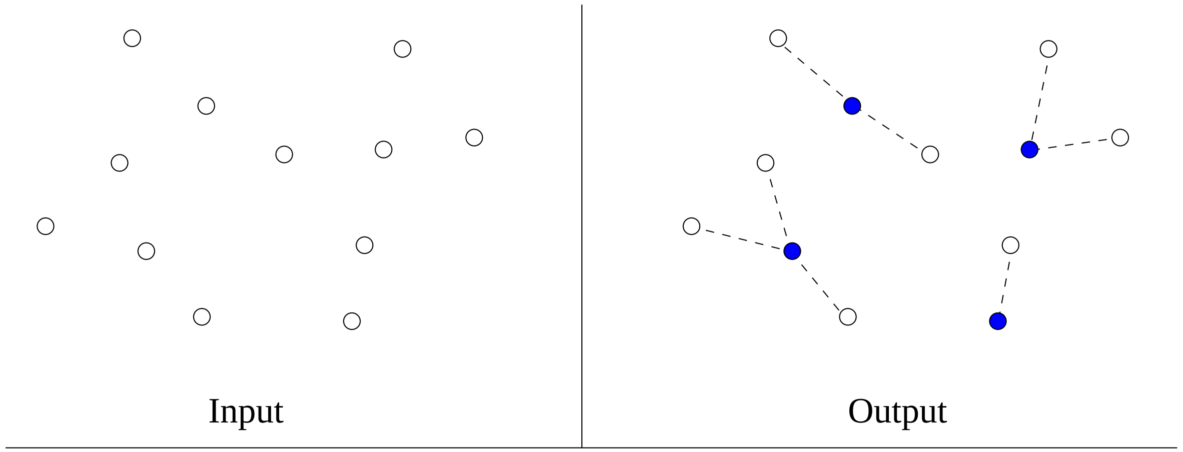


Metric k-Center I

☞ Input: A complete, undirected graph $G = (V, E)$ with edge weights $\omega : E \longrightarrow \mathbb{R}^+$ satisfying the triangle inequality and an integer $k \in \mathbb{N}^+$.

☞ Task: Find a set of vertices $S \subseteq V$ with $|S| \leq k$ such that $\max_{v \in V}\{\text{connect}(v, S)\}$ is minimized, where connect$(v, S) = \omega(e)$ for the cheapest edge e from v to a vertex in S.

Example: $k = 4$



METRIC k-CENTER II

Application: Improve Internet service quality—*content distribution network (CDN)*.

CDN replicates the content from the place of origin to the replica servers scattered over the Internet and serves a request from a replica server close to where the request originates. The replica servers placement problem can be modeled as a k-CENTER problem.

Remarks:

- (METRIC) k-CENTER is NP-hard if k is specified as part of the input.
- Without the triangle inequality, k-CENTER cannot be approximated within factor $f(n)$ for any computable function $f(n)$, assuming P$\neq$NP.

METRIC k-CENTER III

Idea of the *parametric pruning* technique:

A parameter t is chosen, as an estimation (lower bound) on the weight of an optimal solution. For each value of t, the given input instance I is pruned by removing parts which will not be used in any solution of weight at most t.

A reformulation of k-CENTER facilitates the application of parametric pruning:

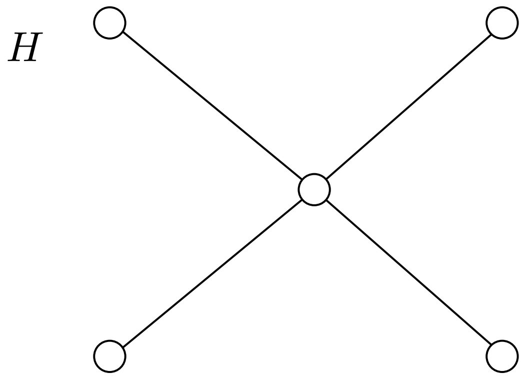
1. Sort the edges of $G = (V, E)$ in nondecreasing order of weight, i.e., $\omega(e_1) \leq \omega(e_2) \leq \ldots \leq \omega(e_m)$, where $m := |E|$;
2. Consider subgraphs $G_i = (V, E_i)$ with $E_i := \{e_1, \ldots, e_i\}$.
3. k-CENTER is equivalent to finding the smallest index i so that G_i has a *dominating set* of size at most k, that is, a vertex subset $D \subseteq V$ such that each vertex in V is either in D or adjacent in G_i to a vertex in D. Then, an optimal solution to k-CENTER has weight $\omega(e_i)$.

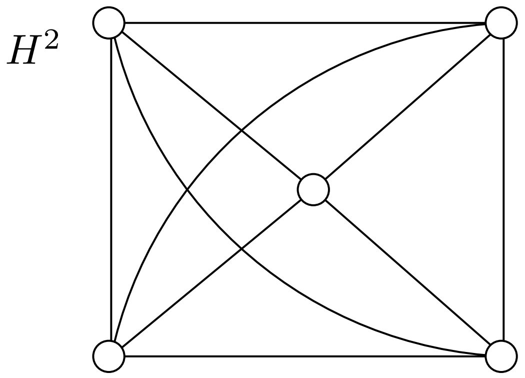
METRIC k-CENTER IV

Definition

The *square* H^2 of a graph H is the graph containing an edge $\{u, v\}$ whenever H has a path consisting of at most two edges between u and v, $u \neq v$.

Example:



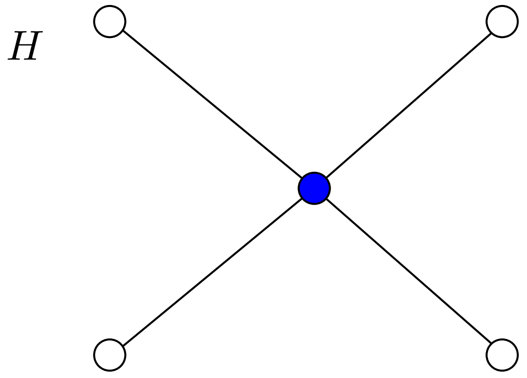


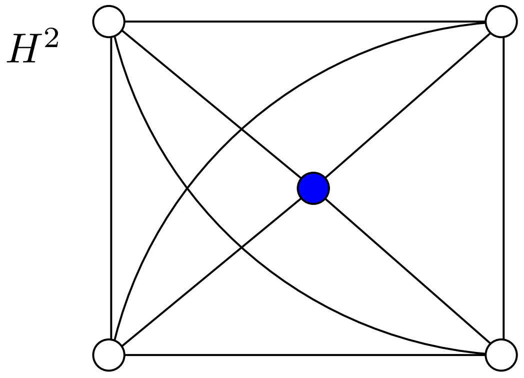
Metric k-Center V

Lemma

Let D be a minimum dominating set of a graph H and let I be an independent set (that is, the vertices from I are not directly connected with each other) in H^2. Then, $|I| \leq |D|$.

Proof: Since D is a dominating set of H, H contains $|D|$ stars spanning all vertices. Since each of these stars will be a clique in H^2, H^2 contains $|D|$ cliques spanning all vertices. Clearly, I can pick at most one vertex from each clique.





METRIC k-CENTER VI

METRIC k-CENTER heuristic:

1. Construct $G_1^2, G_2^2, \ldots, G_m^2$.
2. Compute a *maximal* independent set, M_i, in each G_i^2.
3. $j := \min\{i \ : \ |M_i| \leq k\}$.
4. Return M_j.

Clearly, this algorithm runs in polynomial time.

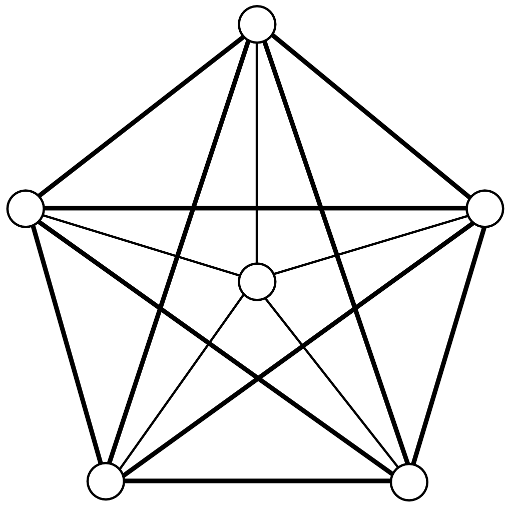
METRIC k-CENTER VII

Theorem

The METRIC k-CENTER heuristic achieves an approximation factor of 2.

Proof: Blackboard.

Tight example: A complete graph on n vertices, where all edges incident to the center vertex have weight 1, and the rest have weight 2. Parameter $k := 1$.



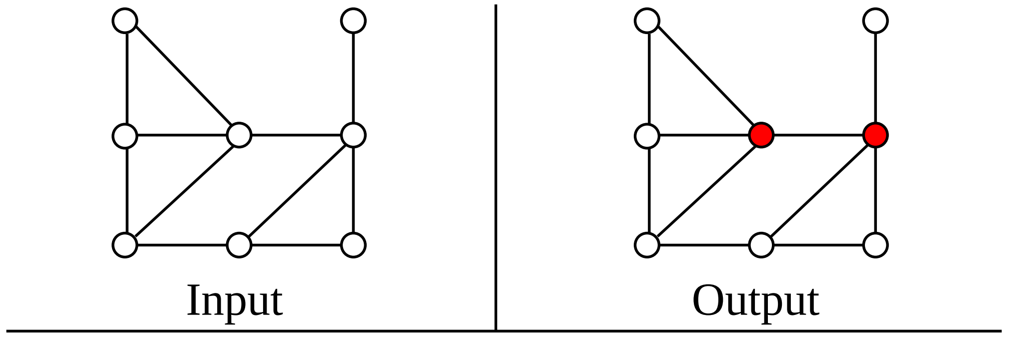
METRIC k-CENTER VIII

Theorem

Assuming P$\neq$NP, there is no polynomial-time algorithm achieving an approximation factor of $2-\epsilon$, $\epsilon > 0$, for METRIC k-CENTER.

Idea of the proof: Show that such an algorithm can solve the NP-complete DOMINATING SET problem in polynomial time.

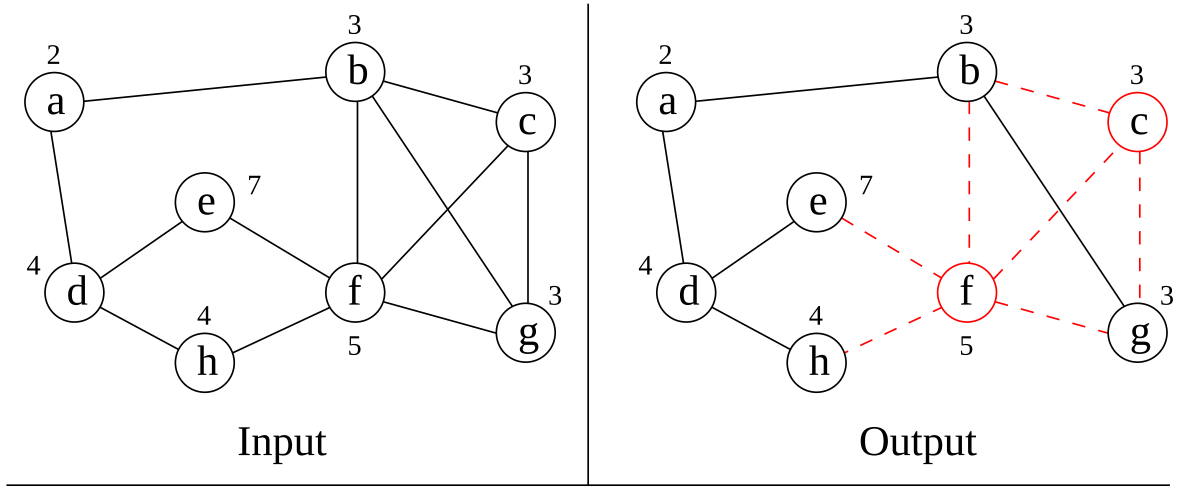
DOMINATING SET: Given an undirected graph $G=(V,E)$, find a minimum-cardinality set of vertices $D \subseteq V$ such that each vertex in V is either in D or adjacent to a vertex in D.



Feedback Vertex Set I

- Input: An undirected graph $G = (V, E)$ and a function $\omega : V \longrightarrow \mathbb{R}^+$ assigning non-negative weights to its vertices.
- Task: Find a minimum weight subset $F \subseteq V$ whose removal leaves an acyclic graph.

Example:



Feedback Vertex Set II

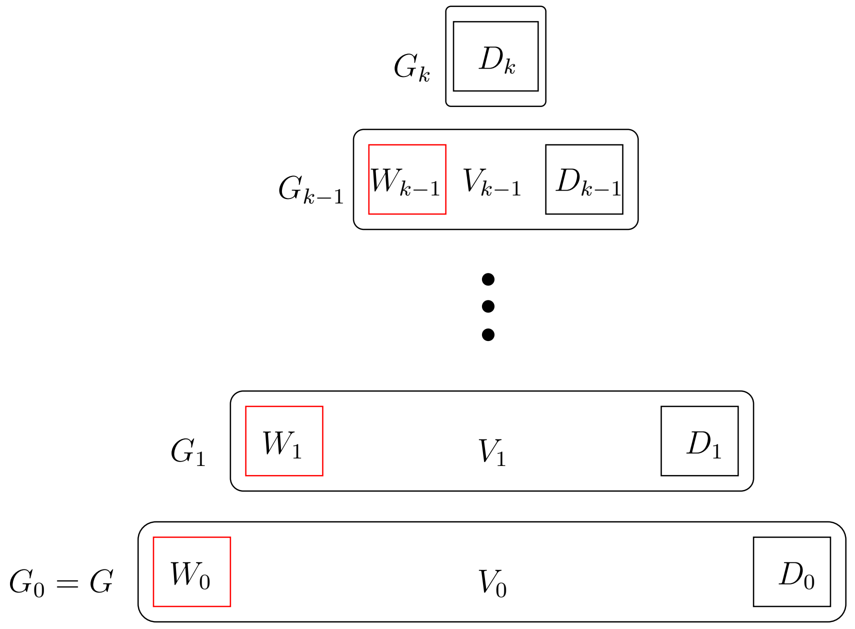
Remarks:

- Feedback Vertex Set is NP-hard.
- Feedback Edge Set (removing edges instead of vertices) is equivalent to Minimum Spanning Tree and, thus, is solvable in linear time.
- In directed graphs,
  - both Feedback Vertex Set and Feedback Edge Set are NP-hard
  - seemingly more difficult to solve than in undirected graphs.

Feedback Vertex Set III

In the following, we use the *layering* technique to obtain a factor-2 approximation algorithm for Feedback Vertex Set.

Recall: Layering for Vertex Cover.



$V_0 = V$

$W_i = \{v \in V_i \mid \omega_i(v) = 0\}$

$D_i = \{v \in V_i \mid \text{deg}_{G_i}(v) = 0\}$

$G_i = G[V_{i-1} \setminus (D_{i-1} \cup W_{i-1})],\ i > 0$

$c_i = \min_{v \in (V_i \setminus D_i)}\{\omega_i(v)/\text{deg}_{G_i}(v)\}$

$t_i(v) = c_i \cdot \text{deg}_{G_i}(v)$

$\omega_i(v) = \omega_{i-1}(v) - t_i(v)$

Feedback Vertex Set IV

Cyclomatic weighted graphs:

Definition

Order the edges of G in arbitrarily, $E = \{e_1, e_2, \ldots, e_m\}$. The *edge space* is the vector space over GF$[2]^m = \{0,1\}^m$ formed by the set of all functions from E to GF$[2] = \{0,1\}$. Vector addition is defined as the symmetric difference of the sets corresponding to the vectors.
The *cycle space* of G is the subspace of the edge space of G that is spanned by the vectors of all cycles in G. The dimension (the minimum number of vectors whose linear combinations can represent every vector in the space) of this space is called *cyclomatic number* cyc(G) of G.

It can be shown that cyc$(G) = |E| - |V| + $ comp(G) where comp(G) denotes the number of connected components of G.

FEEDBACK VERTEX SET V

Use $\delta_H(v)$ to denote the decrease in the cyclomatic number of the graph H on removing vertex v. Let $F = \{v_1, v_2, \ldots, v_f\}$ be a feedback vertex set of G. Clearly,

$$\text{cyc}(G) = \sum_{i=1}^{f} \delta_{G_{i-1}}(v_i),$$

where $G_0 = G$ and, for $i > 0$, $G_i = G[V \setminus \{v_1, \ldots, v_i\}]$.

Definition

A weight function ω is called *cyclomatic* if there is a constant $c > 0$ such that $\omega(v) = c \cdot \delta_G(v)$.

Feedback Vertex Set VI

Theorem

If F is a minimal feedback vertex set of a graph G with a cyclomatic weight function ω, then

$$\omega(F) = \sum_{v \in F} \omega(v) \leq 2 \cdot \text{OPT}.$$

Steps of proof:

(1) Show $\delta_H(v) \leq \delta_G(v)$ for each subgraph H of G.

(2) Show $c \cdot \text{cyc}(G) \leq \text{OPT}$.

(3) Show $\sum_{v \in F} \delta_G(v) \leq 2 \cdot \text{cyc}(G)$.

Then,

$$\omega(F) \overset{\text{Def.}}{=} \sum_{v \in F} c \cdot \delta_G(v) \overset{(3)}{\leq} 2c \cdot \text{cyc}(G) \overset{(2)}{\leq} 2 \cdot \text{OPT}.$$

FEEDBACK VERTEX SET VII

Application of layering to FEEDBACK VERTEX SET: Decompose G into a nested sequence of induced subgraphs, until an acyclic graph is obtained.

Let

$$\begin{aligned} c &:= \min_{v \in V}\{\frac{\omega(v)}{\delta_G(v)}\} \\ t(v) &:= c \cdot \delta_G(v) \\ \omega'(v) &:= \omega(v) - t(v) \\ V' &:= \{v \in V : \omega'(v) > 0\} \\ G' &:= G[V']. \end{aligned}$$

Sequence

$$\begin{aligned} G_0 &:= G; \quad V_0 = V \\ G_i &:= G[V_i]; \quad V_i \supsetneq V_j, \text{ if } j > i \\ t_i &:= \text{cyclomatic weight function of } G_i \\ V_i &:= V_{i-1} \setminus \{v \in V_i : [\omega - \sum_{j=0}^{i-1} t_j](v) = 0\} \end{aligned}$$

Then, $\sum_{i : v \in V_i} t_i(v) = \omega(v)$.

Feedback Vertex Set VIII

Approximation algorithm for Feedback Vertex Set:

1. Decomposition phase

```
$H \leftarrow G;\ \ \omega' \leftarrow \omega;\ \ i \leftarrow 0.$
While $H$ is not acyclic
    $c \leftarrow \min_{u \in H}\{\omega'(u)/\delta_H(u)\};\ \ G_i \leftarrow H;\ \ t_i \leftarrow c \cdot \delta_{G_i}.$
    $\omega' \leftarrow \omega' - t_i;\ \ H \leftarrow G_i[\{u : \omega'(u) > 0\}];\ \ i \leftarrow i + 1.$
$k \leftarrow i;\ \ G_k \leftarrow H.$
```

2. Extension phase

```
$F_k \leftarrow \emptyset.$
For $i = k, \ldots, 1$, extend $F_i$ to a feedback vertex set $F_{i-1}$ of $G_{i-1}$ by
adding a minimal set of vertices from $V_{i-1} \setminus V_i$.
Output $F_0$.
```

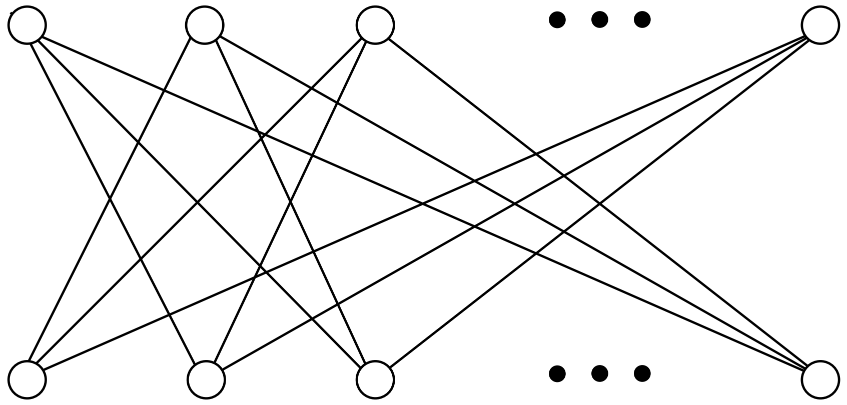
Feedback Vertex Set IX

Theorem

The above algorithm achieves an approximation factor of 2 for Feedback Vertex Set.

Proof: Blackboard.

Tight example: A graph obtained by removing a perfect matching from a complete bipartite graph and duplicating every edge. Each vertex receives the same weight.



Shortest Superstring I

- Input: A finite alphabet Σ and a set of n strings, $S = \{s_1, s_2, \ldots, s_n\}$, $s_i \in \Sigma^+$.
- Task: Find a shortest string s that contains each $s_i \in S$ as a substring.

W.l.o.g., we may assume that no string $s_i \in S$ is a substring of another string $s_j \in S$ with $j \neq i$.

Example:

Input:

$$\begin{aligned} s_1 &= \text{A B R A C} \\ s_2 &= \text{A C A D A} \\ s_3 &= \text{A D A B R} \\ s_4 &= \text{D A B R A} \\ s_5 &= \text{R A C A D} \end{aligned}$$

Output:

$$s = \text{A B R A C A D A B R A}$$

Shortest Superstring II

Shortest Superstring is NP-complete.

Motivations:

- Assembly of DNA-sequences
- Compression of sparse matrices

Definition

Define the *overlap* of two strings s_i and s_j, denoted by overlap(s_i, s_j), as the maximum length of a suffix of s_i that is also a prefix of s_j.
Define prefix(s_i, s_j) as the prefix of s_i obtained by removing overlap(s_i, s_j) from the end of s_i.

Shortest Superstring III

A greedy algorithm:

Select from S two strings that have maximum overlap and replace them with the string obtained by overlapping them as much as possible, until there is only one string in S.

Conjecture: This greedy algorithm achieves an approximation factor of 2.

Tight example: $s_1 = ab^k$, $s_2 = b^k c$, and $s_3 = b^{k+1}$.

Define *compression* achieved by a superstring s as the difference between the sum of the lengths of the strings in S and the length of s. It can be shown that the greedy algorithm achieves at least half the optimal compression.

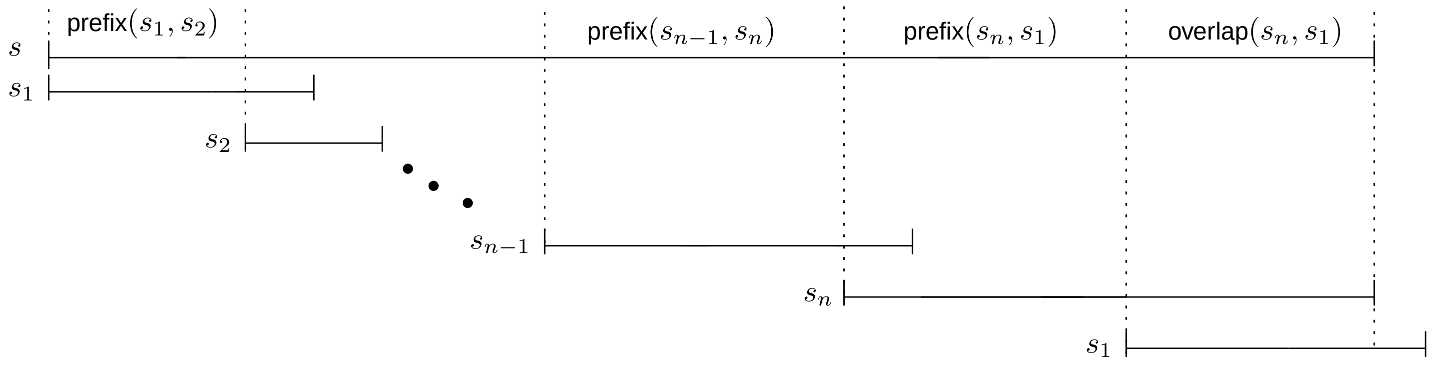
Shortest Superstring IV

Observation

Let s be a shortest superstring of the input strings $s_1, s_2, \ldots, s_n$, $|s| = \text{OPT}$. Assume that the input strings $s_1, s_2, \ldots, s_n$ are numbered in order of leftmost occurrence in s. Then,

$$\begin{aligned}\text{OPT} = \quad & |\text{prefix}(s_1, s_2)| + |\text{prefix}(s_2, s_3)| + \ldots + |\text{prefix}(s_n, s_1)| \\ & + |\text{overlap}(s_n, s_1)|.\end{aligned}$$

Proof:

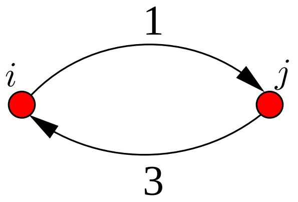


Shortest Superstring V

A close relation between the shortest superstring of S and the minimum traveling salesperson tour on the *prefix graph* of S:

The *prefix graph* of S is a complete, directed, arc-weighted graph on vertex set $\{1, 2, \ldots, n\}$. The weight of arc (i, j) is equal to $|\text{prefix}(s_i, s_j)|$.

Example: $s_i =$abc, $s_j =$bcd.



Clearly, the weight of the tour $1 \to 2 \to \ldots \to n \to 1$ is equal to

$$|\text{prefix}(s_1, s_2)| + |\text{prefix}(s_2, s_3)| + \ldots + |\text{prefix}(s_n, s_1)|.$$

By the observation, the minimum weight of a traveling salesperson tour of the prefix graph of S gives a lower bound on the length of a shortest superstring of S.

Shortest Superstring VI

Key idea: Instead of the minimum traveling salesperson tour, we compute a minimum weight "*cycle cover*" of the prefix graph to lower-bound OPT.

A minimum weight *cycle cover* of an edge-weighted graph G is a minimum weight collection of disjoint cycles covering all vertices of G.

A minimum weight cycle cover of a complete graph can be computed in polynomial time:
Construct an undirected, bipartite, edge-weighted graph $H = (U \cup V, E)$ with $U = \{u_1, \ldots, u_n\}$ and $V = \{v_1, \ldots, v_n\}$. For each $i, j \in \{1, \ldots, n\}$ with $i \neq j$, add edge $\{u_i, v_j\}$ of weight $\big|\text{prefix}(s_i, s_j)\big|$.

It is easy to see that each cycle cover of the prefix graph corresponds to a perfect matching of the same weight in H. A minimum weight perfect matching in bipartite graphs can be computed in polynomial time.

Shortest Superstring VII

Definition

- Let $c = (i_1 \rightarrow i_2 \rightarrow \ldots \rightarrow i_l \rightarrow i_1)$ be a cycle in the prefix graph, let $\alpha(c) = \text{prefix}(s_{i_1}, s_{i_2}) \circ \ldots \circ \text{prefix}(s_{i_{l-1}}, s_{i_l}) \circ \text{prefix}(s_{i_l}, s_{i_1})$.
- Define *weight* of cycle c, wt(c), to be $|\alpha(c)|$.
- Let $\sigma(c) := \alpha(c) \circ s_{i_1}$.
- We will call s_{i_1} the *representative string* for c.

Shortest Superstring VIII

"Obviously", $(\alpha(c))^\infty$ is a superstring of $s_{i_1}, \ldots, s_{i_l}$.

Example: A cycle c on three vertices i_1, i_2, i_3:

s_{i_1}:	a	b	c	abcab
s_{i_2}:		b	c	abcabc
s_{i_3}:			c	abcabca
s_{i_1}:				abcabcab

We have $\alpha(c) =$abc.

$s_{i_1} = (\alpha(c))^2 \circ$ab.

Shortest Superstring IX

An approximation algorithm for Shortest Superstring:

1. Construct the prefix graph corresponding to strings in S.
2. Find a minimum weight cycle cover of the prefix graph, $\mathcal{C} = \{c_1, \ldots, c_k\}$.
3. Output $\sigma(c_1) \circ \sigma(c_2) \circ \ldots \circ \sigma(c_k)$.

Clearly, the output is a superstring of all strings in S.

Shortest Superstring X

Observation

If in each cycle $c \in \mathcal{C}$ we can find a representative string r with $|r| \leq \text{wt}(c)$, then the above algorithm achieves a factor-2 approximation.

Proof: The weight $\sum_{c \in \mathcal{C}} \text{wt}(c)$ of cycle cover $\mathcal{C}$ lower-bounds OPT. Therefore,

$$\begin{aligned}
|\sigma(c_1) \circ \ldots \circ \sigma(c_k)| &= \textstyle\sum_{i=1}^{k} |\alpha(c_i) \circ r_i| \\
&= \textstyle\sum_{i=1}^{k} |\alpha(c_i)| + \sum_{i=1}^{k} |r_i| \\
&\leq \textstyle\sum_{i=1}^{k} \text{wt}(c_i) + \sum_{i=1}^{k} \text{wt}(c_i) \\
&\leq 2 \cdot \text{OPT}.
\end{aligned}$$

Shortest Superstring XI

Thus, the hard case is when all strings of some cycle $c \in \mathcal{C}$ are *longer* than $\alpha(c)$.

But since they must all be substrings of $(\alpha(c))^\infty$, they must be *periodic*.

Goal: Upper-bound $\sum_{c \in \mathcal{C}} |\sigma(c)|$ by $4 \cdot \text{OPT}$.

A useful lemma:

Lemma

Let c and c' be two cycles in $\mathcal{C}$ for a cycle cover $\mathcal{C}$, and let r, r' be the representative strings from c, c'. Then

$$|\text{overlap}(r, r')| < \text{wt}(c) + \text{wt}(c').$$

Proof: Blackboard or omitted.

Shortest Superstring XII

Theorem

The above algorithm achieves an approximation factor of 4 for Shortest Superstring.

Proof: Blackboard.

Remark: The following algorithm achieves an approximation factor of 3 for Shortest Superstring.

1. Construct the prefix graph.
2. Compute a minimum weight cycle cover $\mathcal{C} = \{c_1, \ldots, c_k\}$.
3. Apply the greedy algorithm to $\{\sigma(c_1), \ldots, \sigma(c_k)\}$ and output the resulting string.

KNAPSACK I

Definition

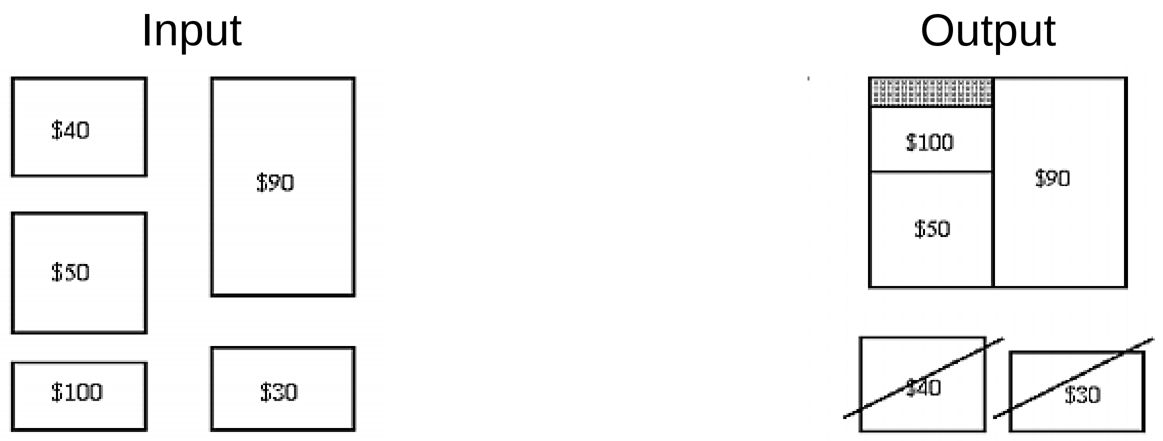
1. Let Π be an NP-hard optimization problem with objective function f_Π. An algorithm $\mathcal{A}$ is an *approximation scheme* for Π if on input (I, ϵ), where I is an instance of Π and ϵ is an error parameter, it outputs a solution s such that
   - $f_\Pi(I, s) \leq (1 + \epsilon) \cdot$ OPT if Π is a minimization problem.
   - $f_\Pi(I, s) \geq (1 - \epsilon) \cdot$ OPT if Π is a maximization problem.
2. $\mathcal{A}$ is called a *polynomial time approximation scheme* (PTAS) if for each *fixed* $\epsilon > 0$ its running time is bounded by a polynomial in the size of instance I.
3. $\mathcal{A}$ is called a *fully polynomial time approximation scheme* (FPTAS) if the running time of $\mathcal{A}$ is bounded by a polynomial in the size of instance I and $1/\epsilon$.

KNAPSACK II

- Input: A set $S = \{a_1, \ldots, a_n\}$ of objects, with specified sizes and profits, size$(a_i) \in \mathbb{Z}^+$ and profit$(a_i) \in \mathbb{Z}^+$ for $1 \leq i \leq n$, and a "knapsack capacity" $B \in \mathbb{Z}^+$.
- Task: Find a subset of objects whose total size is bounded from above by B and the total profit is maximized.

W.l.o.g., we may assume that $\forall\, a \in S,$ size$(a) \leq B$.

Example:



KNAPSACK III

Remarks:

- KNAPSACK typically arises in resource allocation scenarios.
- KNAPSACK is NP-hard if each object must be put entirely in the knapsack or it is not included at all (“$0/1$-KNAPSACK”).
- If all objects have the same size (or profit), then KNAPSACK can be solved trivially by sorting.
- The restricted version where all objects have the same ratio of profit to size remains NP-hard.
- A further NP-hard special case is the so-called INTEGER PARTITION, where we are asked to partition S in two subsets such that both subsets have the same total size.

KNAPSACK IV

Dynamic programming for KNAPSACK:

Let $P := \max_{a \in S}\{\text{profit}(a)\}$. Then $n \cdot P$ is a trivial upper bound on the profit that can be achieved by any solution.

For each $i \in \{1, \ldots, n\}$ and $p \in \{1, \ldots, nP\}$, let $S_{i,p}$ denote a subset of $\{a_1, \ldots, a_i\}$ whose total profit is exactly p and whose total size is minimized.

Define table A as follows: For $1 \leq i \leq n$ and $1 \leq p \leq n \cdot P$,

$$A(i,p) := \begin{cases} \text{total size of } S_{i,p} & \text{if such set exists;} \\ \infty & \text{otherwise.} \end{cases}$$

KNAPSACK V

Dynamic programming for KNAPSACK (continued):

Clearly, $A(1, p)$ is known for every $1 \leq p \leq n \cdot P$. The value of $A(i + 1, p)$ is computed recursively:

$A(i + 1, p) :=$

$$\begin{cases} \min\{A(i, p), \text{size}(a_{i+1}) + A(i, p - \text{profit}(a_{i+1}))\}, & \text{if profit}(a_{i+1}) \leq p; \\ A(i, p) & \text{otherwise.} \end{cases}$$

The maximum achievable profit is then $\max\{p : A(n, p) \leq B\}$.

Running time: $O(n^2 \cdot P)$.

This is a *pseudo-polynomial time algorithm*: The running time is polynomial in n if P is bounded by a polynomial in n.

KNAPSACK VI

An FPTAS for KNAPSACK: “Scale” the profits depending on the error parameter ϵ...:

1. Given $\epsilon > 0$, let $K := \epsilon \cdot P/n$.
2. $\forall a \in S :$ $\text{profit}'(a) := \lfloor \text{profit}(a)/K \rfloor$.
3. With these as profits of objects, using the dynamic programming algorithm, find the most profitable set S'.
4. Output S'.

Lemma

$\text{profit}(S') \geq (1-\epsilon) \cdot \text{OPT}$.

Proof: Blackboard.

Knapsack VII

Theorem

The above algorithm is a fully polynomial time approximation scheme (FPTAS) for Knapsack.

Proof: The running time of the algorithm is

$$O(n^2 \cdot \lfloor P/K \rfloor) = O(n^2 \cdot \lfloor n/\epsilon \rfloor) = O(n^3/\epsilon).$$

Since this is polynomial in n and $1/\epsilon$, the theorem follows.

KNAPSACK VIII

Probably, very few of the known NP-hard problems admit an FPTAS...:

Definition

A problem is called *strongly NP-hard* if it remains NP-hard when restricted to the instances I for which all numbers occurring in I are bounded from above by $p(|I|)$ where $|I|$ is the length of the encoding of instance I and p is a polynomial.

Examples:

- VERTEX COVER is strongly NP-hard. The numbers appearing in any meaningful instance are bounded from above by $n := |V|$ for an input graph $G = (V, E)$.
- TRAVELING SALESPERSON is strongly NP-hard, since this problem is already NP-hard for distance values at most two.

KNAPSACK IX

Examples (continued):

- KNAPSACK is not strongly NP-hard. If the maximum profit P is bounded from above by a polynomial in n, the dynamic programming algorithm with a running time of $O(n^2 \cdot P)$ gives a polynomial-time solving algorithm for KNAPSACK. Thus, KNAPSACK is NP-hard only if P is exponential in n.

Remark: Most known NP-hard problems are strongly NP-hard.

KNAPSACK X

Theorem

Let p be a polynomial and Π be an NP-hard optimization problem such that, on any instance I, $\text{OPT}_\Pi < p(|I_u|)$ where I_u is the encoding of I with all numbers occurring in it written in unary. If Π admits an FPTAS, then it also admits a pseudo-polynomial time algorithm.

Proof (for minimization problems): Suppose that there is an FPTAS for Π whose running time on instances I and error parameter ϵ is $q(|I|, 1/\epsilon)$, where q is a polynomial. Set $\epsilon := 1/p(|I_u|)$ and run the FPTAS. The solution produced has objective function value at most

$$(1+\epsilon)\cdot \text{OPT}_\Pi < \text{OPT}_\Pi + \epsilon \cdot p(|I_u|) = \text{OPT}_\Pi + 1.$$

In other words, this solution is optimal. The running time is then $O(q(|I|, p(|I_u|)))$, that is, polynomial in $|I_u|$.

Knapsack XI

Corollary

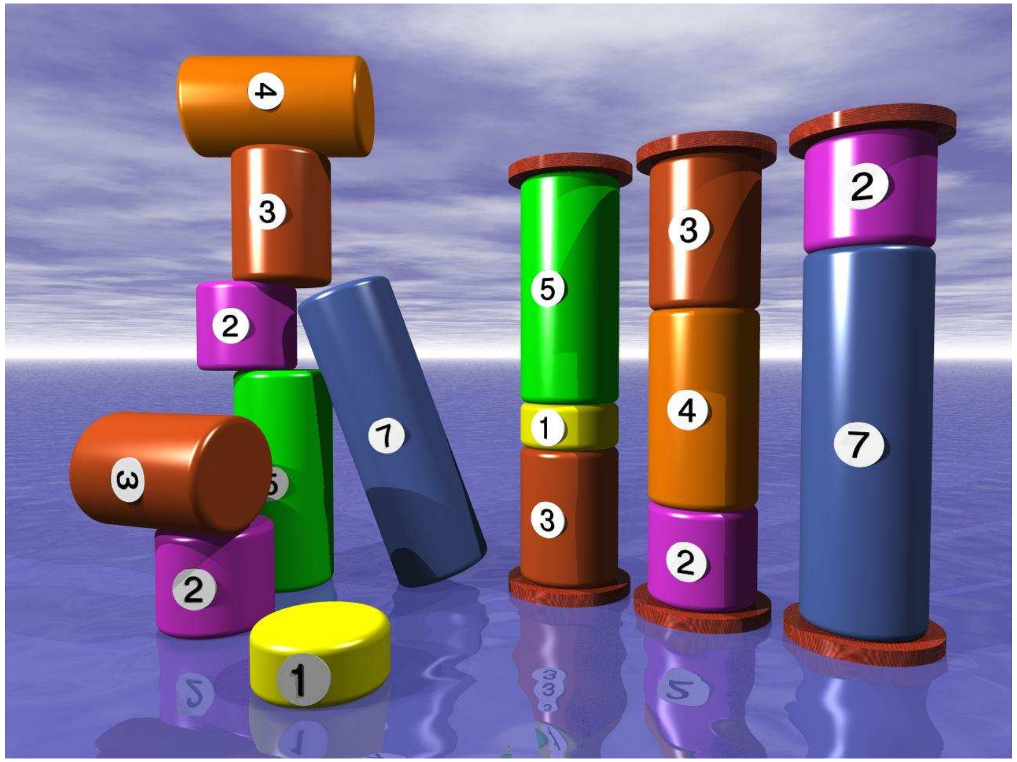
Let Π be an NP-hard optimization problem satisfying the restrictions of the above theorem. If Π is strongly NP-hard, then Π does not admit an FPTAS, unless P$=$NP.

Proof: If Π admits an FPTAS, then it admits a pseudo-polynomial time algorithm by the above theorem. But then it is not strongly NP-hard, assuming P$\neq$NP, leading to a contradiction.

Bin Packing I

- **Input:** A set of n objects with sizes $s_1, s_2, \ldots, s_n \in (0, 1]$.
- **Task:** Find a packing in bins of size 1 that minimizes the number of bins used.

Example: (Number $x \in \{1, 2, 3, 4, 5, 6, 7\}$ should be read as size $0.x$)



BIN PACKING II

- BIN PACKING arises in a variety of packing and manufacturing problems.
- Even the most elementary-sounding BIN PACKING problems usually are NP-complete.
- The choice of algorithms strongly depends on the shapes of the objects to be packed: one-dimensional vs. two-dimensional vs. three-dimensional ...
- Further, many applications have often peculiar, problem-specific constraints.
- The difference between off-line and on-line problems is important.

Bin Packing III

A simple factor-2 approximation algorithm, the so-called *First-Fit* algorithm:

Insert each object one by one into the first bin that has room for it. If no bin has room for it, open a new bin.

Analysis of the approximation factor:

If the First-Fit algorithm uses m bins, then at least $m-1$ bins are more than half full. Therefore,

$$\sum_{i=1}^{n} s_i > \frac{m-1}{2}.$$

Since the sum of the object sizes is a lower bound on OPT, $m-1 < 2 \cdot$ OPT, that is, $m \leq 2 \cdot$ OPT.

BIN PACKING IV

Lower bound for approximability of BIN PACKING

Useful remark: The problem of deciding whether there is a way to partition n non-negative numbers $s_1, s_2, \ldots, s_n$ into two sets, each adding up to $\frac{1}{2} \cdot \sum_{i=1}^{n} s_i$, the so-called PARTITION problem, is NP-hard.

Theorem

For any $\epsilon > 0$, there is no approximation algorithm achieving a factor of $3/2 - \epsilon$ for BIN PACKING, assuming P$\neq$NP.

Proof: Assume that there is such an algorithm. Then we show how to solve PARTITION in polynomial time. An instance of PARTITION is a “Yes”-instance iff the n objects can be packed in 2 bins of size $\frac{1}{2} \cdot \sum_{i=1}^{n} s_i$. If the answer is “yes” then the factor-$(3/2 - \epsilon)$ algorithm will have to give an optimal packing and thereby solve the PARTITION problem.

Bin Packing V

We will show that, for any ϵ with $0 < \epsilon \leq 1/2$, there is an algorithm $\mathcal{A}_\epsilon$ that runs in time polynomial in n and finds a packing using at most $(1 + 2\epsilon) \cdot \text{OPT} + 1$ bins.

Remark: The algorithm $\mathcal{A}_\epsilon$ is an *asymptotic polynomial time approximation scheme* for Bin Packing.

Bin Packing VI

Algorithm $\mathcal{A}_\epsilon$ for Bin Packing:

1. Remove objects of size $< \epsilon$.
2. Round the object sizes to obtain a constant number of object sizes.
3. Find an optimal packing for the “simplified” instance from Step 2.
4. Use this packing for packing the objects with their original sizes.
5. Pack remaining objects of size $< \epsilon$ using First-Fit.

Bin Packing VII

Lemma

Let $\epsilon > 0$ and let K be a constant. Consider the restriction of Bin Packing to instances in which each object is of size at least ϵ and the number of distinct object sizes is K. There is a polynomial-time algorithm that optimally solves this restricted problem.

Proof: The number of objects in a bin is bounded from above by $M := \lfloor 1/\epsilon \rfloor$.

Then, the number of different bin types is bounded from above by $R = \binom{M+K}{M}$,

which is a (large!) constant. Clearly, the total number of bins used is at most n.

Therefore, the number of possible feasible packings is bounded from above by

$P = \binom{n+R}{R}$, which is polynomial in n. Enumerating them and picking the best packing gives an optimal solution.

Bin Packing VIII

Lemma

Bin Packing, restricted to instances in which each object is of size at least ϵ for a fixed $\epsilon > 0$, is in polynomial time approximable within a factor of $1 + \epsilon$.

Proof: Blackboard.

Theorem

For any ϵ with $0 < \epsilon \leq 1/2$, the algorithm $\mathcal{A}_\epsilon$ runs in time polynomial in n and finds a packing using at most $(1 + 2\epsilon) \cdot \text{OPT} + 1$ bins.

Proof: Blackboard.

Approximation Algorithms

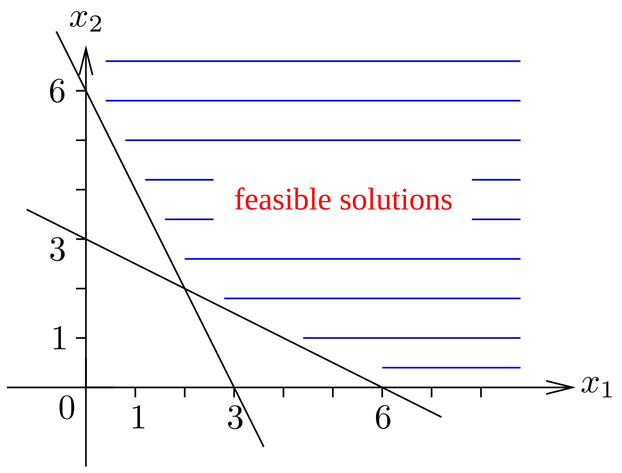
Introduction to Linear Programming I

Definition

Linear programming (LP) is the problem of optimizing a linear function subject to linear inequality *constraints*. The function being optimized is called the *objective function*.

Example for linear inequality constraints:

$$\begin{aligned} x_1 &\geq 0 \\ x_2 &\geq 0 \\ x_1 + 2x_2 &\geq 6 \\ 2x_1 + x_2 &\geq 6 \end{aligned}$$



Introduction to Linear Programming II

In general, the linear inequality constraints can be defined as

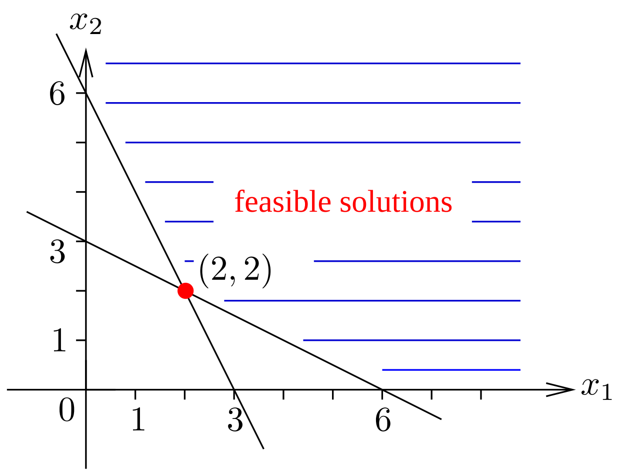
$$Ax \geq b,$$

where A is a matrix, x and b are vectors over $\mathbb{R}$. Linear programming optimizes a given objective function $c^t x$ for a fixed vector c, where x can be chosen freely under all x obeying $Ax \geq b$.

Example (continued):
Objective function:

$$\text{minimize} \qquad \frac{3}{2}x_1 + x_2,$$

that is $c = (3/2, 1)$. A solution $x^t = (2, 2)$ achieves an objective function value of $c^t x = 5$.



Introduction to Linear Programming III

Standard formulation of the linear programming problem (minimization problem):

- ☞ Input: An $m \times n$-matrix A over $\mathbb{R}$ and two vectors $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$.
- ☞ Task: Find a vector $x \in \mathbb{R}^n$ with $x \geq 0$ and $Ax \geq b$ such that $c^t x$ is minimized.

Remark: Linear programming with integer coefficient values for (A, b, c) is polynomial-time solvable.

ILP and WEIGHTED VERTEX COVER I

WEIGHTED VERTEX COVER

- Input: An undirected graph $G = (V, E)$ and a positive rational weight ω_i for each vertex $i \in V$.
- Task: Find a minimum-weight vertex cover C.

Objective function

$$\text{minimize} \sum_{i \in V} \omega_i x_i.$$

Variable x_i represents the vertex i.

$$x_i = 1 \quad \Leftrightarrow \quad i \in C$$

$$x_i = 0 \quad \Leftrightarrow \quad i \notin C$$

Constraints

$$x_i + x_j \geq 1, \quad \forall \{i, j\} \in E$$

$$x_i \in \{0, 1\}, \quad \forall i \in V$$

Observe that with the constraint $x_i \in \{0, 1\}$, $\forall i \in V$ we have now an *integer linear program (ILP)* instead of a linear program.

ILP and WEIGHTED VERTEX COVER II

Remark: Solving integer linear programs is NP-hard.

Approximation algorithm for WEIGHTED VERTEX COVER by using *LP-relaxation* and *rounding*...:

LP-relaxation: Replace the constraint $x_i \in \{0,1\}, \forall i \in V$ by $0 \leq x_i \leq 1, \quad \forall i \in V$ and solve the resulting linear program.

Lemma

Let x^* with $0 \leq x_i^* \leq 1$ be the solution vector of the linear program and let C be an optimal vertex cover. Then, $\omega_{\text{LP}} \leq \omega(C)$ where $\omega_{\text{LP}} := \sum_{i=1}^{n} \omega_i \cdot x_i^*$.

Proof: Since vertex covers "1:1-correspond" to the solutions of the ILP and the feasible solutions of the ILP form a subset of the feasible solutions of the LP, the lemma follows.

ILP and WEIGHTED VERTEX COVER III

LP-relaxation provides a natural lower bound for the optimum objective function value.

Example: Let G be a complete graph over n vertices and all vertices have weight 1.

Solve the following LP:

Objective function

$$\text{minimize} \quad \sum_{i \in V} \omega_i x_i .$$

Constraints

$$\begin{aligned} x_i + x_j \geq 1 &\quad \forall \{i,j\} \in E \\ 0 \leq x_i \leq 1 &\quad \forall i \in V \end{aligned}$$

We have an LP-solution with $x_i^* = 1/2$ for all i and $\omega_{\text{LP}} = n/2$. Clearly, a minimum-weight vertex cover has weight OPT $= n - 1$.

ILP and Weighted Vertex Cover IV

Rounding: Convert the fractional solution obtained by the LP-relaxation into an integral solution, trying to ensure that in the process the cost does not increase much.

Clearly,

$x_i^* = 1 \quad \rightarrow x_i = 1$ vertex i is in vertex cover

$x_i^* = 0 \quad \rightarrow x_i = 0$ vertex i is not in vertex cover

Round variables x_i^* with $0 < x_i^* < 1$:

$x_i^* \geq 1/2 \quad \rightarrow x_i = 1$ vertex i is in vertex cover

$x_i^* < 1/2 \quad \rightarrow x_i = 0$ vertex i is not in vertex cover

ILP and WEIGHTED VERTEX COVER V

Lemma

Let $S := \{i \mid x_i^* \geq 1/2\}$. Then, S is a vertex cover and $\omega(S) \leq 2 \cdot \omega_{\text{LP}}$.

Proof: 1. S is a vertex cover: From the constraint $\forall \{i,j\} \in E,\ x_i + x_j \geq 1$, we know that at least one of x_i^* and x_j^* has a value $\geq 1/2$. Thus, S contains at least one of i and j.

2. $\omega(S) \leq 2 \cdot \omega_{\text{LP}}$: Since $\forall i \in S,\ x_i^* \geq 1/2$, then

$$\omega_{\text{LP}} = \sum_{i=1}^{n} \omega_i x_i^* \geq \sum_{i \in S} \omega_i x_i^* \geq \Big(\sum_{i \in S} w_i\Big)/2 = \omega(S)/2.$$

Theorem

The above LP-based rounding algorithm achieves a factor-2 approximation for WEIGHTED VERTEX COVER.

LP-Duality I

Example: A minimization LP in standard form:

$$
\begin{array}{ll}
\text{Minimize} & 7x_1 + x_2 + 5x_3 \\
\text{Constraints} & x_1 - x_2 + 3x_3 \geq 10 \\
 & 5x_1 + 2x_2 - x_3 \geq 6 \\
 & x_1, x_2, x_3 \geq 0
\end{array}
$$

Let x^* denote the optimum value of this LP. A natural question:

Is x^* at most α for a rational number α?

For $\alpha = 30$, we get a feasible solution $x = (2, 1, 3)$, which is called a *Yes-certificate* for $\alpha = 30$.

Clearly, each Yes-certificate provides an upper bound on x^*.

LP-Duality II

Example (continued):

How do we provide a *No-certificate* for such a question, that is, how do we place a good lower bound on the optimal objective function value?

$$\begin{array}{ll} \text{Minimize} & 7x_1 + x_2 + 5x_3 \\ \text{Constraints} & x_1 - x_2 + 3x_3 \geq 10 \\ & 5x_1 + 2x_2 - x_3 \geq 6 \\ & x_1, x_2, x_3 \geq 0 \end{array}$$

Idea: Find suitable non-negative multipliers for the constraints so that when we take their sum, the coefficient of each x_i in the sum is dominated by the coefficient in the objective function. Now, the right-hand side of this sum is a lower bound on x^*.

LP-Duality III

Example (continued):

The problem of choosing the multipliers in such a way that the right-hand side of the sum is as large as possible again can be formulated as a linear program (y_1 and y_2 are the multipliers we search for):

Minimize $7x_1 + x_2 + 5x_3$

Constraints

$$x_1 - x_2 + 3x_3 \geq 10$$
$$5x_1 + 2x_2 - x_3 \geq 6$$
$$x_1, x_2, x_3 \geq 0$$

Primal LP

Maximize $10y_1 + 6y_2$

Constraints

$$y_1 + 5y_2 \leq 7$$
$$-y_1 + 2y_2 \leq 1$$
$$3y_1 - y_2 \leq 5$$
$$y_1, y_2 \geq 0$$

Dual LP

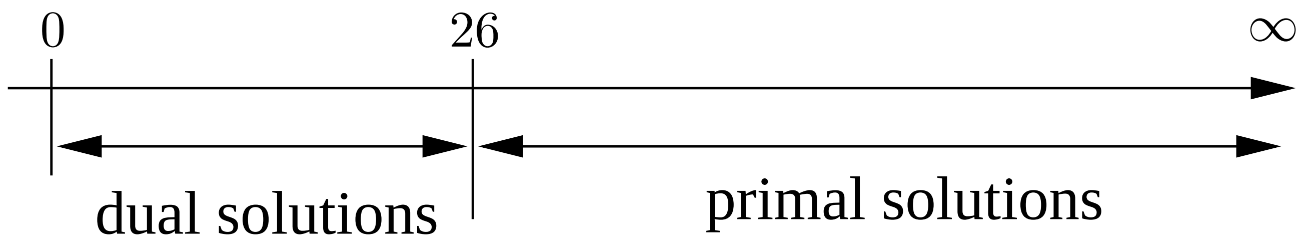
LP-Duality IV

Observation

Every feasible solution to the primal LP gives an upper bound on the optimum value of the dual LP and every feasible solution to the dual LP gives a lower bound on the optimum value of the primal LP (*MinMax-relation*).

If we can find feasible solutions for the dual and the primal LPs with matching objective function value, then both solutions must be optimal.

Example (continued): $x = (7/4, 0, 11/4)$ and $y = (2, 1)$ both achieve objective function values of 26 and thus both are optimal solutions.



LP-Duality V

Let a_{ij}, b_i, c_j be rational numbers.

Primal LP (minimization problem)		Dual LP (maximization problem)	
Minimize	$\sum_{j=1}^{n} c_j x_j$	Maximize	$\sum_{i=1}^{m} b_i y_i$
Constraints	$\sum_{j=1}^{n} a_{ij} x_j \geq b_i,$ $i = 1, \ldots, m$	Constraints	$\sum_{i=1}^{m} a_{ij} y_i \leq c_j,$ $j = 1, \ldots, n$
	$x_j \geq 0,$ $j = 1, \ldots, n$		$y_i \geq 0,$ $i = 1, \ldots, m$

LP-Duality VI

It can be shown:

Theorem (LP-Duality Theorem)

The primal LP has a finite optimum iff its dual LP has a finite optimum. Moreover, if $x^* = (x_1^*, \ldots, x_n^*)$ and $y^* = (y_1^*, \ldots, y_m^*)$ are optimal solutions for the primal and dual LPs, respectively, then

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*.$$

LP-Duality VII

Theorem (Weak Duality Theorem)

If $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_m)$ are feasible solutions for the primal and dual LPs, respectively, then

$$\sum_{j=1}^{n} c_j x_j \geq \sum_{i=1}^{m} b_i y_i.$$

Proof:

$$\sum_{j=1}^{n} c_j x_j \overset{(\star)}{\geq} \sum_{j=1}^{n} \Big(\sum_{i=1}^{m} a_{ij} y_i \Big) x_j = \sum_{i=1}^{m} \Big(\sum_{j=1}^{n} a_{ij} x_j \Big) y_i \overset{(\star\star)}{\geq} \sum_{i=1}^{m} b_i y_i.$$

$(\star)$: y is dual feasible and x_j's are non-negative.

$(\star\star)$: x is primal feasible and y_i's are non-negative,

LP-Duality VIII

Corollary (Complementary slackness conditions)

Let $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_m)$ be primal and dual feasible solutions, respectively. Then, x and y are both optimal iff all the following conditions are satisfied:

Primal complementary slackness conditions

For each $1 \leq j \leq n$: either $x_j = 0$ or $\sum_{i=1}^{m} a_{ij} y_i = c_j$; and

Dual complementary slackness conditions

For each $1 \leq i \leq m$: either $y_i = 0$ or $\sum_{j=1}^{n} a_{ij} x_j = b_i$.

The complementary slackness conditions play a vital role in the design of efficient algorithms, both exact and approximation.

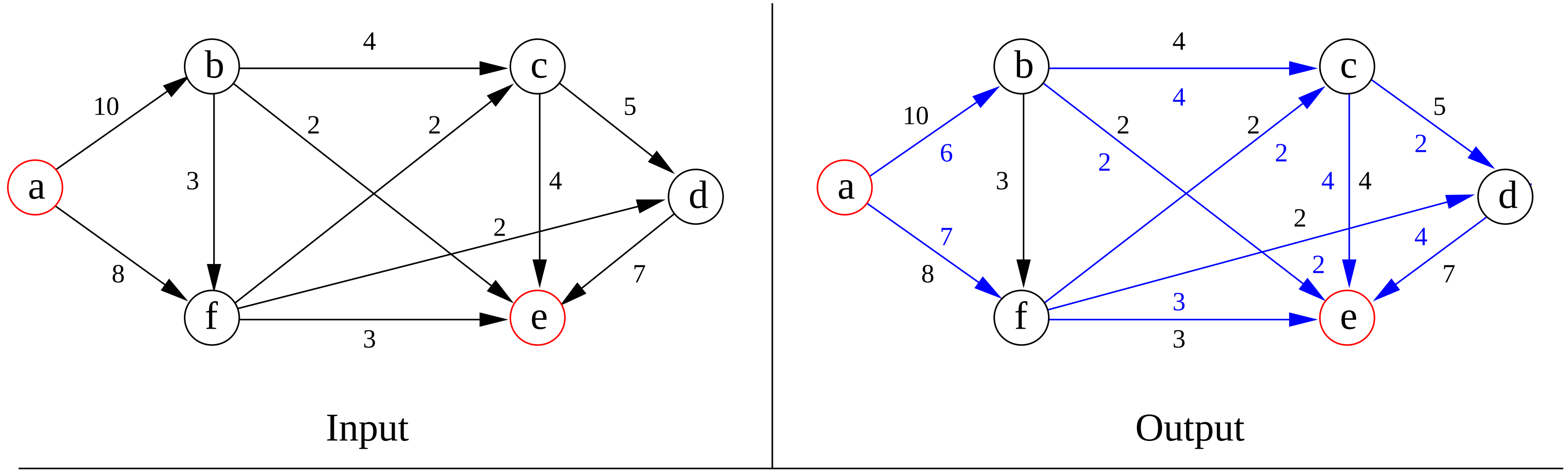
MinMax-Relations and LP-Duality I

MAXIMUM FLOW

- Input: A connected, directed graph $G = (V, E)$ with an arc capacity function $c : E \longrightarrow \mathbb{R}^+$ and two distinct vertices s (source) and t (sink or target).
- Task: Find the maximum amount of flow that can be sent from s to t subject to
  1. *capacity constraint*: for each arc e, the flow sent through e is bounded from above by e's capacity, and
  2. *flow conservation*: at each vertex v other than s and t, the total flow into v equals the total flow out of v.

MinMax-Relations and LP-Duality II

Example:



The maximum amount of an a–e flow is 13.

- An *s-t cut* is defined by a partition of V into X and $\overline{X}$ so that $s \in X$ and $t \in \overline{X}$: it consists of the set of arcs going from X to $\overline{X}$
- The *capacity* of an s-t cut, $c(X, \overline{X})$, is defined as the sum of capacities of the arcs in the cut.

MinMax-Relations and LP-Duality III

It can be shown that in each “flow graph” the maximum amount of an s-t flow is equal to the minimum capacity of an s-t cut (MaxFlow-MinCut Theorem).

Formulation of MAXIMUM FLOW as LP:

- Introduce an arc (t, s) of infinite capacity from t to s;
- The objective now is to maximize the flow on (t, s).
- Let f_{ij} denote the amount of flow sent through arc $(i, j) \in E$.

MinMax-Relations and LP-Duality IV

LP of MAXIMUM FLOW:

$$
\begin{array}{lll}
\text{Maximize} & f_{ts} & \\
\text{Constraints} & f_{ij} \leq c_{ij}, & \forall (i,j) \in E \\
& \sum_{\{j:(j,i)\in E\}} f_{ji} - \sum_{\{j:(i,j)\in E\}} f_{ij} \leq 0, & \forall i \in V \\
& f_{ij} \geq 0, & \forall (i,j) \in E
\end{array}
$$

Dual LP:

$$
\begin{array}{lll}
\text{Minimize} & \sum_{(i,j)\in E} c_{ij} d_{ij} & \\
\text{Constraints} & d_{ij} - p_i + p_j \geq 0, & \forall (i,j) \in E \\
& p_s - p_t \geq 1 & \\
& d_{ij} \geq 0, & \forall (i,j) \in E \\
& p_i \geq 0, & \forall i \in V
\end{array}
$$

MinMax-Relations and LP-Duality IV

For the sake of better understanding, transform the dual LP into an ILP:

$$\begin{array}{lll}
\text{Minimize} & \sum_{(i,j)\in E} c_{ij} d_{ij} & \\
\text{Constraints} & d_{ij} - p_i + p_j \geq 0, & \forall (i,j) \in E \\
 & p_s - p_t \geq 1 & \\
 & d_{ij} \in \{0,1\}, & \forall (i,j) \in E \\
 & p_i \in \{0,1\}, & \forall i \in V
\end{array}$$

This ILP is a formulation of the *minimum s-t cut* problem! The dual LP is the *LP-relaxation* of this ILP.

MinMax-Relations and LP-Duality V

A feasible solution of the dual LP can be interpreted as a *fractional s-t cut*: The sum of the d_{ij}-values for arcs (i,j) on a path from s to t is at least 1. The capacity of a fractional s-t cut is defined as the dual objective function value achieved by this cut.

By the LP-Duality Theorem it can be shown that the maximum flow in G must equal the capacity of a minimum fractional s-t cut. Since the latter equals the capacity of a minimum s-t cut, we get the MaxFlow-MinCut Theorem.

The MaxFlow-MinCut Theorem is a special case of the LP-Duality Theorem.

MinMax-Relations and LP-Duality V

The usefulness of the complementary slackness conditions:

Let f^* be an optimal solution of the primal LP (a maximum s-t flow). Let (d^*, p^*) be an integral optimum solution of the dual LP, and let $(X, \overline{X})$ be the cut defined by (d^*, p^*). Note that $s \in X$ and $t \in \overline{X}$.

We use the complementary slackness conditions to show that arcs going from X to $\overline{X}$ are saturated by f^* and the reverse arcs carry no flow:

- Consider arc (i, j) with $i \in X$ and $j \in \overline{X}$. By $d^*_{ij} = 1$ and the dual complementary slackness condition, $f^*_{ij} = c_{ij}$.
- Consider arc (k, l) with $l \in X$ and $k \in \overline{X}$. Since $p^*_k - p^*_l = -1$ and $d^*_{kl} \in \{0, 1\}$, constraint $d^*_{kl} - p^*_k + p^*_l \geq 0$ must be satisfied as a strict inequality. By the primal complementary slackness condition, $f^*_{kl} = 0$.

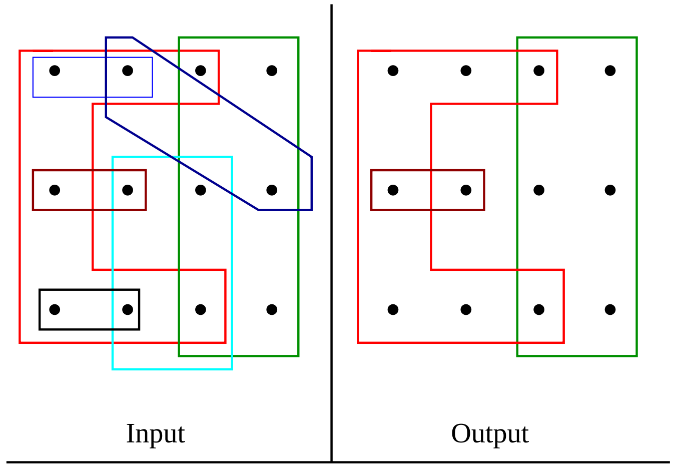
SET COVER via Dual Fitting I

Dual fitting helps analyze combinatorial algorithms using LP-duality theory.

SET COVER

- Input: A universe U of n elements, a collection of subsets of U, $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$, and a cost function $c : \mathcal{S} \to \mathbb{Q}^+$.
- Task: Find a minimum cost subcollection of $\mathcal{S}$ that covers all elements of U.

Example: $(\forall S \in \mathcal{S}, c(S) = 1)$



Set Cover via Dual Fitting II

Formulation of Set Cover as an ILP: Assign a variable x_S for each set $S \in \mathcal{S}$.

Set x_S to 1 iff S is picked in the set cover.

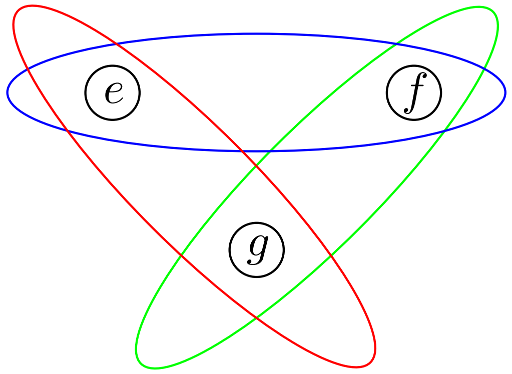
$$\begin{array}{lll} \text{Minimize} & \sum_{S\in\mathcal{S}} c(S)\cdot x_S & \\ \text{Constraints} & \sum_{\{S:e\in S\}} x_S \geq 1, & \forall e \in U \\ & x_S \in \{0,1\}, & \forall S \in \mathcal{S} \end{array}$$

LP-relaxation (Primal LP)

$$\begin{array}{lll} \text{Minimize} & \sum_{S\in\mathcal{S}} c(S) x_S & \\ \text{Constraints} & \sum_{\{S:e\in S\}} x_S \geq 1, & \forall e \in U \\ & x_S \geq 0, & \forall S \in \mathcal{S} \end{array}$$

SET COVER via Dual Fitting III

Example: $U = \{e, f, g\}$ and $\mathcal{S} = \{S_1 = \{e, f\}, S_2 = \{e, g\}, S_3 = \{f, g\}\}$. $c(S_1) = c(S_2) = c(S_3) = 1$.



A *fractional* solution $x^*_{S_1} = x^*_{S_2} = x^*_{S_3} = 1/2$ gives a fractional set cover of cost $3/2$. An optimal integral set cover must pick two of the sets for a total cost of 2.

Set Cover via Dual Fitting IV

Primal LP

$$\begin{array}{lll} \text{Minimize} & \sum_{S\in\mathcal{S}} c(S)x_S & \\ \text{Constraints} & \sum_{\{S:e\in S\}} x_S \geq 1, & \forall e \in U \\ & x_S \geq 0, & \forall S \in \mathcal{S} \end{array}$$

Dual LP: Introduce a variable y_e for each element $e \in U$.

$$\begin{array}{lll} \text{Maximize} & \sum_{e\in U} y_e & \\ \text{Constraints} & \sum_{\{e:e\in S\}} y_e \leq c(S), & \forall S \in \mathcal{S} \\ & y_e \geq 0, & \forall e \in U \end{array}$$

The minimization LP is called a *covering LP* (all coefficients non-negative) and the maximization LP is called a *packing LP*.

SET COVER via Dual Fitting V

Recall: Greedy SET COVER Algorithm:

```
1. $C \leftarrow \emptyset$.
2. While $C \neq U$ do
     Pick the set $S$ with a minimal cost-effectiveness $c(S)/|S \setminus C|$.
     $C \leftarrow C \cup S$ and for each $e \in S \setminus C$ set price$(e) := c(S)/|S \setminus C|$.
3. Output the picked sets.
```

Set Cover via Dual Fitting VI

Lemma

By setting $y_e = \text{price}(e)/H_n$ with $H_n = 1 + 1/2 + 1/3 + \ldots + 1/n$, we have a feasible solution to the dual LP.

Proof: Blackboard.

Theorem

The greedy algorithm achieves an approximation factor of H_n for Set Cover.

Proof: Blackboard.

SET COVER via Dual Fitting VII

Tight example: $U = \{e_1, e_2, \ldots, e_n\}$ with $n = 2^k - 1$ for a positive integer k. For $1 \leq i \leq n$, consider index i as a k-bit string and view this string as a k-dimensional vector $\overrightarrow{i}$ over GF$[2]$. Set $S_i := \{e_j \mid \overrightarrow{i} \cdot \overrightarrow{j} = 1\}$ and $\mathcal{S} := \{S_1, S_2, \ldots, S_n\}$. For all $S \in \mathcal{S}$, $c(S) := 1$.

Then, with some effort, we can observe:

- $x_{S_i} = 2/(n+1)$, $1 \leq i \leq n$, is a fractional set cover. Its cost is $2n/(n+1)$.
- Any optimal integral set cover must pick at least k sets and, thus, has a cost at least $k = \log_2(n+1)$.

Then, the lower bound on the quotient of both costs ("*integrality gap*") is

$$\left(\frac{n+1}{2n}\right) \cdot \log_2(n+1) > \frac{\log_2 n}{2}.$$

SET COVER via Dual Fitting VIII

Remark: The greedy algorithm and its analysis using dual fitting extend naturally to several generalizations of SET COVER, e.g., the so-called SET MULTICOVER:

- Input: A universe U of n elements, a collection of subsets of U, $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$, a cost function $c : \mathcal{S} \to \mathbb{Q}^+$, and a positive integer r_e for each $e \in U$.
- Task: Find a minimum cost set cover S' such that each elements e in U is contained in at least r_e sets in S'. Hereby, a set $S \in \mathcal{S}$ can be picked more than only once.

SET COVER and Rounding I

Deterministic rounding:

Recall: LP-relaxation of SET COVER:

$$\begin{array}{lll} \text{Minimize} & \sum_{S \in \mathcal{S}} c(S) \cdot x_S & \\ \text{Constraints} & \sum_{\{S : e \in S\}} x_S \geq 1, & \forall e \in U \\ & x_S \geq 0, & \forall S \in \mathcal{S} \end{array}$$

A simple rounding scheme: Round up all non-zero variables to 1 and pick all corresponding sets S with $x_S = 1$.

SET COVER and Rounding II

Slight modification of the simple rounding scheme: Let f be the frequency of the most frequent element, that is, $f := \max_{\{e \in U\}} |\{S : e \in S, S \in \mathcal{S}\}|$.

Deterministic rounding algorithm for SET COVER:

1. Find an optimal solution to the LP-relaxation.
2. Pick all sets S for which $x_S \geq 1/f$ in this solution.

SET COVER and Rounding III

Theorem

The above rounding algorithm achieves an approximation factor of f for SET COVER.

Proof: Let $\mathcal{C}$ denote the collection of the sets output by the rounding algorithm.

1. $\mathcal{C}$ is a set cover: Consider an arbitrary element $e \in U$. Since e is in at most f sets, one of the sets must be picked to the extent of at least $1/f$ in the fractional set cover. Thus, e is covered by $\mathcal{C}$ and hence $\mathcal{C}$ is a set cover.

2. The approximation factor is f: Since the rounding process increases x_S for each set $S \in \mathcal{C}$ by a factor of at most f, the cost of $\mathcal{C}$ is at most f times the cost of the fractional set cover, thereby proving the desired approximation factor.

Set Cover and Rounding IV

Tight example: Let $V := V_1 \cup V_2 \cup \ldots \cup V_k$ be the union of k disjoint sets of cardinality n each. Consider the hypergraph $H = (V, E)$ where $E := V_1 \times V_2 \times \ldots \times V_k = \{(v_{1i_1}, \ldots, v_{ki_k}) : 1 \leq i_1, \ldots, i_k \leq n\}$. We construct a Set Cover instance:

- $U := E$,
- $\mathcal{S} := \{S_{ij} : 1 \leq i \leq k, 1 \leq j \leq n\}$ where $S_{ij} = \{e \in E : e$ contains $v_{ij}\}$.
- $\forall S \in \mathcal{S}$, $c(S) := 1$.

“Clearly”, $x_S = 1/k = 1/f$ for all $S \in \mathcal{S}$ gives an optimal fractional set cover with cost $1/k \cdot nk = n$. Thus, the rounding algorithm will pick all $n \cdot k$ sets, while picking all sets S_{1j} for $1 \leq j \leq n$ gives a set cover of cost n.

SET COVER and Rounding V

Randomized rounding: View the fractions as probabilities, round the variables according to random experiments based on these probabilities...

Example: $x_i^* = 1/3 \Rightarrow$ Round x_i^* with probability of $1/3$ up to 1 and with probability of $2/3$ down to 0.

Theorem

Randomized rounding leads within expected polynomial time to a factor-$O(\log n)$ randomized approximation algorithm for SET COVER.

Proof: Blackboard.

SET COVER via the Primal-Dual Schema I

Primal-Dual Schema

- is one of the basic techniques to obtain approximation algorithms using linear programming.
- has the following advantages compared to LP-rounding:
  - It yields *combinatorial algorithms* with good approximation factors and good running time.
  - It leaves more room to exploit the special combinatorial structure of individual problems.
- has its origins in the design of exact algorithms.

SET COVER via the Primal-Dual Schema II

Overview of the schema — Relaxation of complementary slackness conditions.

Recall:

Primal LP (minimization problem)

Minimize	$\sum_{j=1}^{n} c_j x_j$
Constraints	$\sum_{j=1}^{n} a_{ij} x_j \geq b_i,$
	$i = 1, \ldots, m$
	$x_j \geq 0,$
	$j = 1, \ldots, n$

Primal complementary slackness conditions:

$\forall 1 \leq j \leq n$: either $x_j = 0$ or $\sum_{i=1}^{m} a_{ij} y_i = c_j$.

Dual LP (maximization problem)

Maximize	$\sum_{i=1}^{m} b_i y_i$
Constraints	$\sum_{i=1}^{m} a_{ij} y_i \leq c_j,$
	$j = 1, \ldots, n$
	$y_i \geq 0,$
	$i = 1, \ldots, m$

Dual complementary slackness conditions:

$\forall 1 \leq i \leq m$: either $y_i = 0$ or $\sum_{j=1}^{n} a_{ij} x_j = b_i$.

SET COVER via the Primal-Dual Schema III

Primal LP (minimization problem)

Minimize $\sum_{j=1}^{n} c_j x_j$

Constraints $\sum_{j=1}^{n} a_{ij} x_j \geq b_i,$

$i = 1, \ldots, m$

$x_j \geq 0,$

$j = 1, \ldots, n$

Primal complementary slackness conditions:

$\forall 1 \leq j \leq n$: either $x_j = 0$ or $c_j/\alpha \leq \sum_{i=1}^{m} a_{ij} y_i \leq c_j$ for an $\alpha \geq 1$.

Dual LP (maximization problem)

Maximize $\sum_{i=1}^{m} b_i y_i$

Constraints $\sum_{i=1}^{m} a_{ij} y_i \leq c_j,$

$j = 1, \ldots, n$

$y_i \geq 0,$

$i = 1, \ldots, m$

Dual complementary slackness conditions:

$\forall 1 \leq i \leq m$: either $y_i = 0$ or $b_i \leq \sum_{j=1}^{n} a_{ij} x_j \leq \beta \cdot b_i$ for a $\beta \geq 1$.

SET COVER via the Primal-Dual Schema IV

Theorem

If x and y are primal and dual feasible solutions satisfying the slackness conditions stated above, then

$$\sum_{j=1}^{n} c_j x_j \leq \alpha \cdot \beta \cdot \sum_{i=1}^{m} b_i y_i .$$

Proof: Blackboard.

Set Cover via the Primal-Dual Schema V

Overview of the schema — Algorithm:

1. Start with a primal infeasible solution and a dual feasible solution; usually, $x = \underbrace{(0, 0, \ldots, 0)}_{n}$ and $y = \underbrace{(0, 0, \ldots, 0)}_{m}$.
2. Iteratively improve the feasibility of the primal solution and the optimality of the dual solution, ensuring that in the end a primal feasible solution is obtained and all slackness conditions with a suitable choice of α and β are satisfied.

Note: The primal solution x is always extended integrally, ensuring that the final solution is integral.

Analysis of the approximation factor:

- The cost of the dual solution is used as a lower bound on OPT.
- By the above theorem, the approximation factor is then $\alpha \cdot \beta$.

SET COVER via the Primal-Dual Schema VI

We use the primal-dual schema to obtain a factor-f approximation algorithm for SET COVER, where f denotes the frequency of the most frequent element.

Primal LP:		Dual LP:	
Minimize	$\sum_{S \in \mathcal{S}} c(S) \cdot x_S$	Maximize	$\sum_{e \in U} y_e$
Constraints	$\sum_{\{S : e \in S\}} x_S \geq 1, \quad \forall e \in U$	Constraints	$\sum_{e \in S} y_e \leq c(S), \quad \forall S \in \mathcal{S}$
	$x_S \geq 0, \quad \forall S \in \mathcal{S}$		$y_e \geq 0, \quad \forall e \in U$

SET COVER via the Primal-Dual Schema VII

We choose $\alpha = 1$ and $\beta = f$.

Primal complementary slackness conditions:

$\forall S \in \mathcal{S}$:

$$x_S \neq 0 \Rightarrow \sum_{e \in S} y_e = c(S).$$

If $\sum_{e \in S} y_e = c(S)$ is true, then we call S *tight*.

Dual complementary slackness conditions:

$\forall e \in U$:

$$y_e \neq 0 \Rightarrow \sum_{\{S : e \in S\}} x_S \leq f.$$

SET COVER via the Primal-Dual Schema VIII

Algorithm for SET COVER:

1. Initialization: $x \leftarrow \vec{0}$, $y \leftarrow \vec{0}$.
2. Until all elements are covered do:
   - Pick an uncovered element e and raise y_e until some set S goes "tight", that is, $\sum_{\{e : e \in S\}} y_e = c(S)$.
   - Pick all tight sets in the cover and update x_S to 1.
   - Declare all the elements occurring in these sets as "covered."
3. Output the sets S with $x_S = 1$.

SET COVER via the Primal-Dual Schema IX

Theorem

The primal-dual algorithm achieves an approximation factor f for SET COVER.

Proof: Clearly, the sets output by the algorithm are a set cover. Since the primal and dual solutions satisfy the relaxed complementary slackness conditions with $\alpha = 1$ and $\beta = f$, by the above theorem the approximation factor is f.

Tight example: $U := \{e_1, \dots, e_{n+1}\}$,
$\mathcal{S} := \{\{e_1, e_n\}, \{e_2, e_n\}, \dots, \{e_{n-1}, e_n\}, U\}$. The first n subsets in $\mathcal{S}$ have cost 1. The last one has cost $1 + \epsilon$ for a small $\epsilon > 0$. Clearly, $f = n$.

Suppose that the algorithm raises y_{e_n} in the first iteration. Then, all sets $\{e_i, e_n\}$, $i = 1, \dots, n-1$, go tight and all of them are picked in the cover. In the second iteration, $y_{e_{n+1}}$ is raised to ϵ and the set U goes tight. Thus, the resulting set cover has a cost of $n + \epsilon$, whereas the optimal cover has cost $1 + \epsilon$.

Approximation Algorithms

Hardness of Approximation I

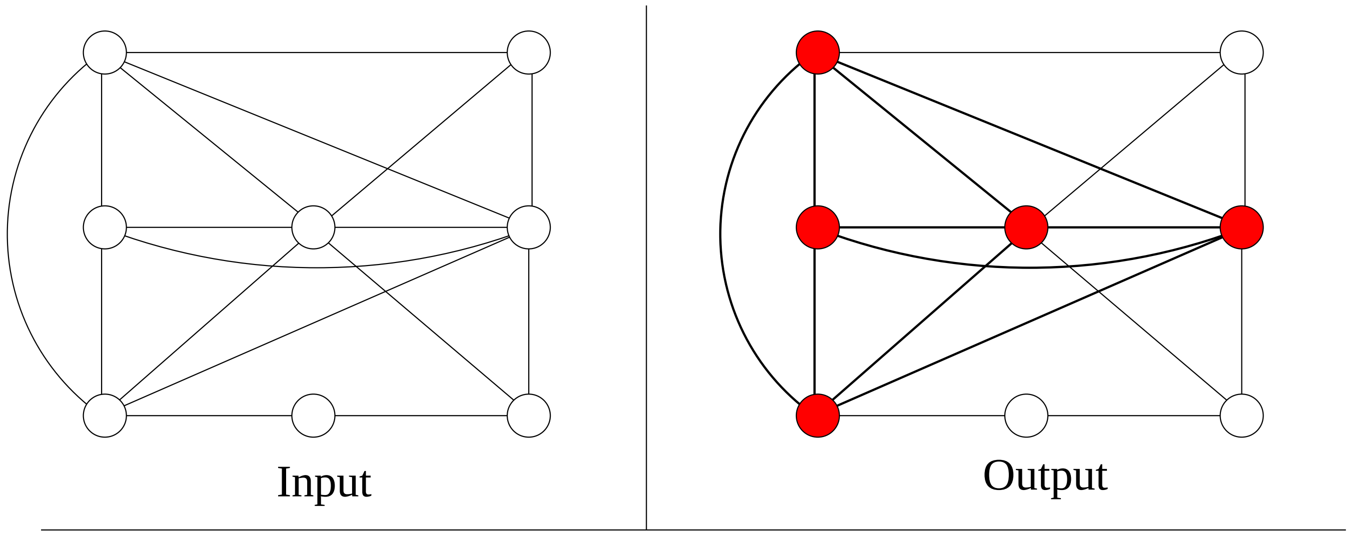
- Recall: Unless P$=$NP, there is no polynomial-time algorithm which gives a “non-trivial approximation” for TRAVELING SALESPERSON.
- Is VERTEX COVER polynomial-time approximable within a factor of $2-\epsilon$ for a $\epsilon > 0$?
- Is SET COVER polynomial-time approximable better than a factor of $\theta(\ln n)$?
- Which problems allow a PTAS and which do not?
- What is the best possible approximation factor for INDEPENDENT SET and CLIQUE?
- ...

Hardness of Approximation II

Example: CLIQUE

- Input: An undirected graph $G = (V, E)$.
- Task: Find a maximum-size complete subgraph of G.

Example:



Hardness of Approximation III

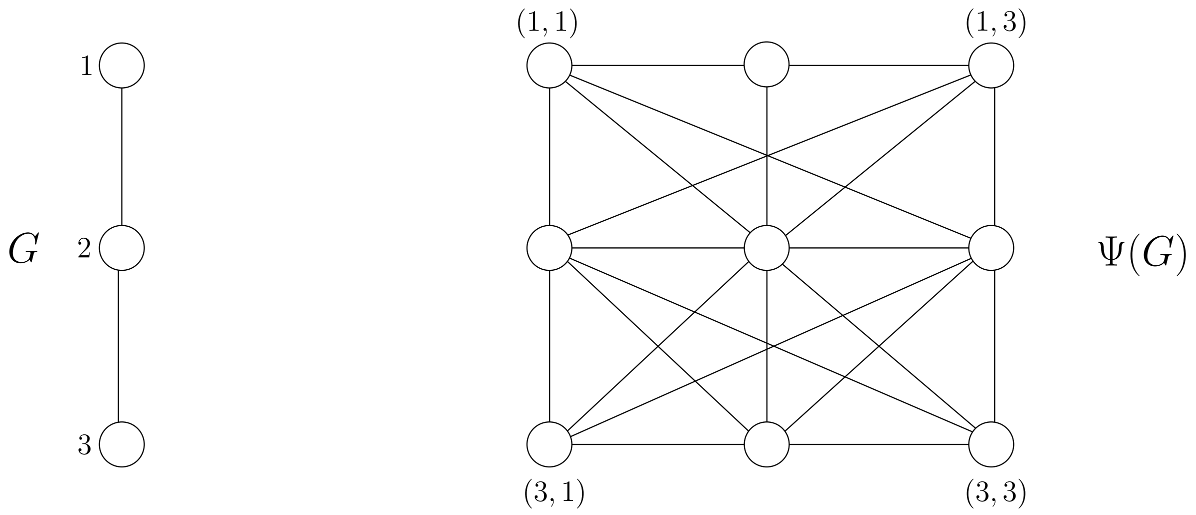
We show: "CLIQUE can be approximated either very well or very poorly."

Consider the following function Ψ for an undirected graph $G = (V, E)$:

$$\Psi(G) := (V \times V, E'),$$

where $E' := \{\{(u,v),(u'v')\} \mid (u = u' \text{ and } \{v,v'\} \in E) \text{ or } (\{u,u'\} \in E)\}$.

Example:



Hardness of Approximation IV

Lemma

G has a clique with at least k vertices iff $\Psi(G)$ has a clique with at least k^2 vertices.

Proof: Blackboard.

Theorem

If there exists a polynomial-time constant-factor approximation algorithm for CLIQUE, then CLIQUE admits a PTAS.

Proof: Blackboard.

Gap-Preserving Reductions I

Example: Assume that there is a reduction from SATISFIABILITY (SAT) to VERTEX COVER with the following properties:

Given a Boolean formula F in CNF, we can construct a graph $G = (V, E)$ such that

- if F is satisfiable, then G has a vertex cover C with $|C| \leq 2/3 \cdot |V|$, and
- if F is not satisfiable, then the smallest vertex cover of G is of size $> \alpha \cdot 2/3 \cdot |V|$ for a fixed constant $\alpha > 1$.

As a consequence of the reduction:

Claim

There is no polynomial-time algorithm for VERTEX COVER that achieves an approximation factor of α, unless P$=$NP.

Proof: Blackboard.

Gap-Preserving Reductions II

Definition

Let Π be a minimization problem. A *gap-introducing reduction* from SAT to Π comes with two parameters, functions f and α. Given an instance F of SAT, it outputs, in polynomial time, an instance x of Π, such that,

- if F is satisfiable, OPT$(x) \leq f(x)$, and
- if F is not satisfiable, OPT$(x) > \alpha(|x|) \cdot f(x)$ with $\alpha(|x|) > 1$.

The *gap*, $\alpha(|x|)$, is the “hardness factor” for Π.

In the example, $f(x) = f(G = (V, E)) = 2/3 \cdot |V|$ and $\alpha(|G|) > 1$ is a constant.

For maximization problems, it can be defined analogously. (There: $\alpha(|x|) < 1$.)

Gap-Preserving Reductions III

Once we have obtained a gap-introducing reduction from SAT (or any other NP-hard problem) to an optimization problem Π_1, we can prove a hardness result for another optimization problem Π_2 by giving a *gap-preserving reduction* from Π_1 to Π_2.

Schema:

$$\begin{tikzcd}
\text{SAT} & \overset{\alpha}{\leq} \arrow[d] & \Pi_1 & \overset{\alpha,\beta}{\leq} \arrow[d] & \Pi_2. \\
& \text{gap-introducing} & & \text{gap-preserving} &
\end{tikzcd}$$

Gap-Preserving Reductions IV

Definition

Let Π_1 be a minimization problem and Π_2 a maximization problem. A *gap-preserving reduction* from Π_1 to Π_2 comes with four parameters (functions), f_1, α, f_2, and β. Given an instance x of Π_1, it computes, in polynomial time, an instance y of Π_2 such that

- OPT$(x) \leq f_1(x) \Rightarrow$ OPT$(y) \geq f_2(y)$,
- OPT$(x) > \alpha(|x|) \cdot f_1(x) \Rightarrow$ OPT$(y) < \beta(|y|) \cdot f_2(y)$ with $\alpha(|x|) \geq 1$ and $\beta(|y|) \leq 1$.

For other combinations of minimization and maximization problems analogous definitions.

Gap-Preserving Reductions V

“Composing” a gap-introducing reduction with a gap-preserving reduction with $f = f_1$:

$$\begin{tikzcd}
\text{SAT} & \overset{f,\alpha}{\leq} \arrow[d] & \Pi_1 & \overset{f_1,\alpha,f_2,\beta}{\leq} \arrow[d] & \Pi_2, \\
& \text{gap-introducing} & & \text{gap-preserving} &
\end{tikzcd}$$

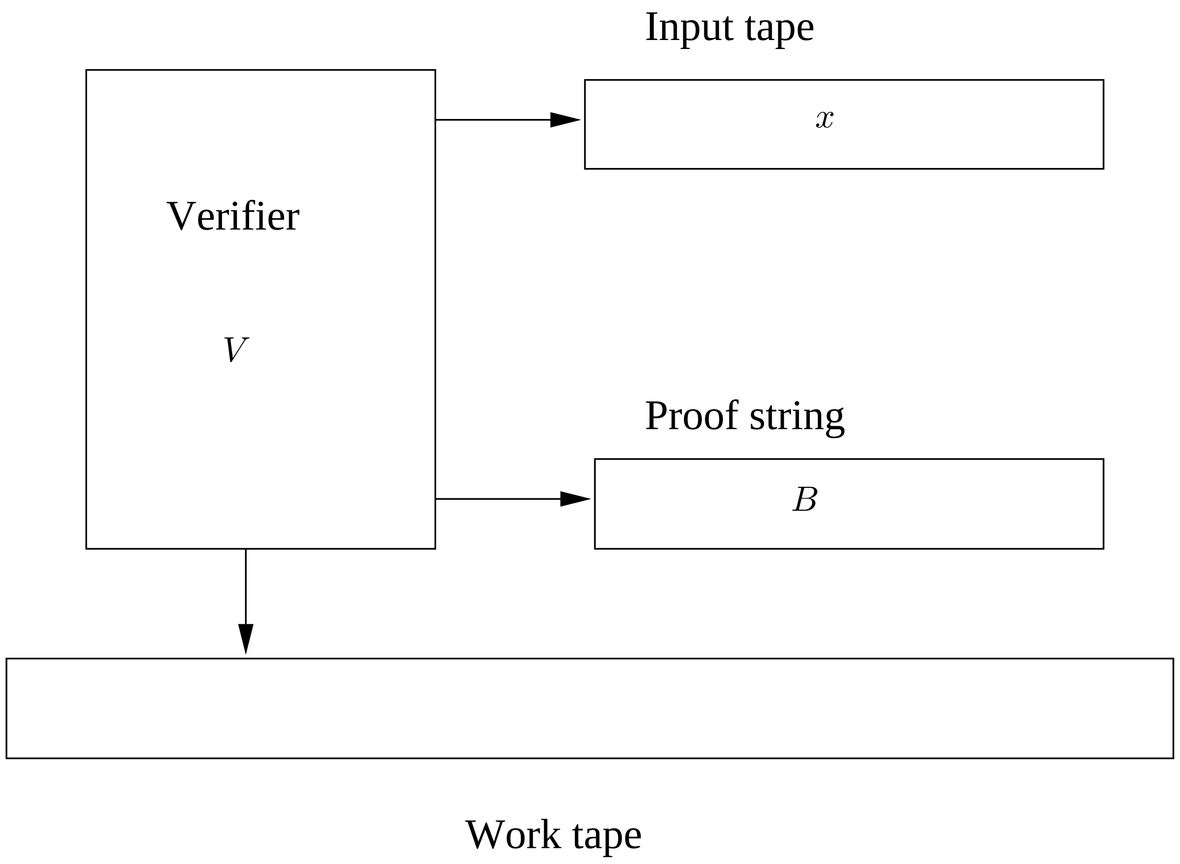
yields a gap-introducing reduction:

$$\begin{tikzcd}
\text{SAT} & \overset{f_2,\beta}{\leq} \arrow[d] & \Pi_2 \\
& \text{gap-introducing} &
\end{tikzcd}$$

This composed reduction shows that there is no $\beta(|y|)$-factor approximation algorithm for Π_2 unless P$=$NP.

The PCP Theorem I

A characterization of NP:



The PCP Theorem II

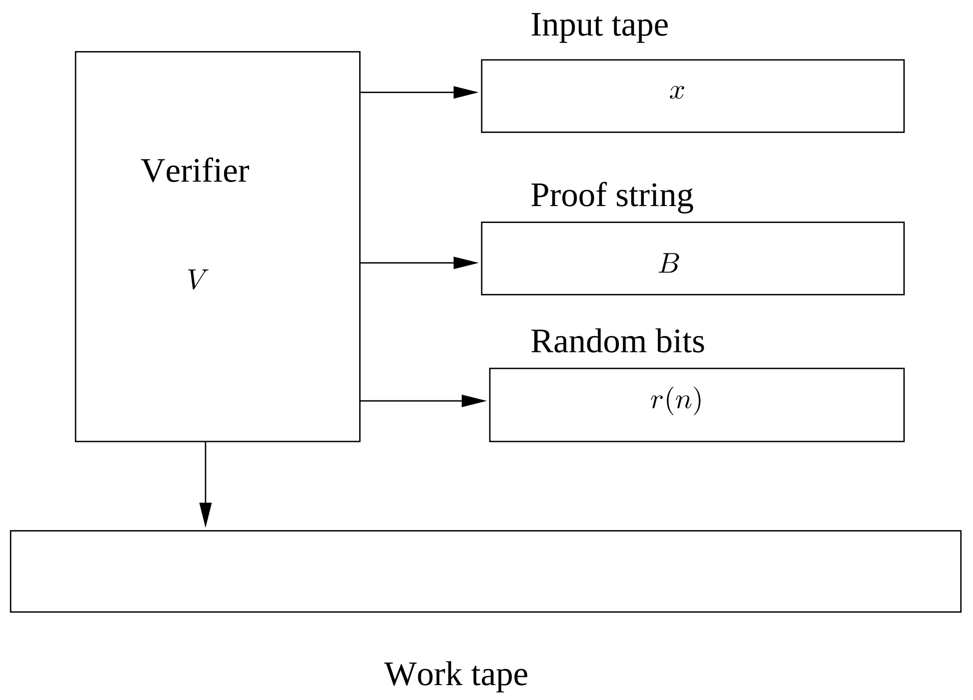
A probabilistic characterization of the class NP: *PCP (probabilistically checkable proof systems)*.

Definition

A language L is in the complexity class PCP$(r(n), q(n))$, if there is a "verifier" (algorithm) V that, for each input x with $|x| = n$, works in polynomial time, uses $O(r(n))$ random bits, and queries $O(q(n))$ proof bits such that

- if $x \in L$, then there is a proof string B that makes V accept with probability 1,
- if $x \notin L$, then, for every proof string B, V accepts with probability $< 1/2$.

The PCP Theorem III



Note that by definition, NP $= \cup_{k=1}^{\infty}$ PCP$(0, n^k)$.

Theorem (*PCP Theorem*)

NP $=$ PCP$(\log n, 1)$.

The PCP Theorem IV

Some intuition for the PCP theorem: A verifier for 3SAT whose error probability (that is, probability of accepting unsatisfiable formula) is $\leq 1 - 1/m$ where m denotes the number of clauses in the input formula F.

The verifier expects a satisfying truth assignment to F as the proof string.

1. It uses the $O(\log n)$ random bits to pick a random clause of F.
2. Then, it reads the truth assignments for the three variables of this clause. Note that this is only a constant number of bits.
3. It accepts iff the truth setting for these three variables satisfies the clause.

Clearly, if F is satisfiable, then there is a proof string that makes the verifier accept with probability 1, and if not, on every proof string, the verifier accepts with probability $\leq 1 - 1/m$.

The PCP Theorem V

The PCP theorem directly leads to an optimization problem:

MAXIMUM ACCEPT PROBABILITY

- ☞ Input: A Boolean formula F and a PCP$(\log n, 1)$-verifier V.
- ☞ Task: Find a proof string B that maximizes the probability of acceptance of V.

Theorem

Assuming P$\neq$NP, there is no polynomial-time factor-0.5 approximation algorithm for MAXIMUM ACCEPT PROBABILITY.

Proof: Blackboard.

The PCP Theorem V

Application of the PCP theorem to inapproximability of CLIQUE:

Theorem

Assuming P$\neq$NP, there is no polynomial-time constant-factor approximation algorithm for CLIQUE.

Proof: Blackboard.