Approximation Algorithms for Graphs

Arijit Bishnu
(arijit@isical.ac.in)

Advanced Computing and Microelectronics Unit
Indian Statistical Institute
Kolkata 700108, India.

Talk at NWCDMA, Jadavpur University, March 13, 2014

Organization

Outline

Decision and optimization versions

Clique: Decision and optimization versions

- (Optimization version:) Given an undirected graph $G = (V, E)$, find the clique – the largest complete subgraph.

Decision and optimization versions

Clique: Decision and optimization versions

- (Optimization version:) Given an undirected graph $G = (V, E)$, find the clique – the largest complete subgraph.
- (Decision version:) Given an undirected graph $G = (V, E)$, does there exist a clique of size k?

Decision and optimization versions

Clique: Decision and optimization versions

- (Optimization version:) Given an undirected graph $G = (V, E)$, find the clique – the largest complete subgraph.
- (Decision version:) Given an undirected graph $G = (V, E)$, does there exist a clique of size k?
- Similarly, we have for independent set, vertex cover, chromatic number, etc.

Decision and optimization versions

Clique: Decision and optimization versions

- (Optimization version:) Given an undirected graph $G = (V, E)$, find the clique – the largest complete subgraph.
- (Decision version:) Given an undirected graph $G = (V, E)$, does there exist a clique of size k?
- Similarly, we have for independent set, vertex cover, chromatic number, etc.
- Notice that the optimization versions and decision versions are polynomially equivalent.

Decision and optimization versions

Clique: Decision and optimization versions

- (Optimization version:) Given an undirected graph $G = (V, E)$, find the clique – the largest complete subgraph.
- (Decision version:) Given an undirected graph $G = (V, E)$, does there exist a clique of size k?
- Similarly, we have for independent set, vertex cover, chromatic number, etc.
- Notice that the optimization versions and decision versions are polynomially equivalent.
- Consider any decision problem Π and any instance I of Π. The problem is to classify any such instance I to either a yes or no instance.

The class **P** and beyond

The key idea

- We encounter certain problems that are difficult to solve – seems that the problems are not tractable.

The class **P** and beyond

The key idea

- We encounter certain problems that are difficult to solve – seems that the problems are not tractable.
- On the other hand, we have seen problems for which there are efficient solutions – solutions in polynomial time, polynomial in the input size.

The class **P** and beyond

The key idea

- We encounter certain problems that are difficult to solve – seems that the problems are not tractable.
- On the other hand, we have seen problems for which there are efficient solutions – solutions in polynomial time, polynomial in the input size.

The class **P**

Problems that have deterministic algorithms solving them in polynomial time constitute the class of problems **P**.

The class **P** and beyond

The key idea

- We encounter certain problems that are difficult to solve – seems that the problems are not tractable.
- On the other hand, we have seen problems for which there are efficient solutions – solutions in polynomial time, polynomial in the input size.

The class **P**

Problems that have deterministic algorithms solving them in polynomial time constitute the class of problems **P**.

Beyond the class **P**

How do we grade problems based on their difficulty level? From here, starts the notion of polynomial-time reductions.

Polynomial-time reduction

Polynomial-time reducibility between decision problems Π and Π'

We say that Π reduces to Π' in deterministic polynomial-time, symbolized as $\Pi \leq_P \Pi'$, if there exists a deterministic polynomial time algorithm A that takes I, an instance of Π, as input and transforms it into I', an instance of Π' such that I is a yes-instance if and only if I' is an yes-instance.

Polynomial-time reduction

Polynomial-time reducibility between decision problems Π and Π'

We say that Π reduces to Π' in deterministic polynomial-time, symbolized as $\Pi \leq_P \Pi'$, if there exists a deterministic polynomial time algorithm A that takes I, an instance of Π, as input and transforms it into I', an instance of Π' such that I is a yes-instance if and only if I' is an yes-instance.

An important offshoot of the above definition

Suppose $\Pi \leq_P \Pi'$. If Π' can be solved in polynomial time, then Π can also be solved in polynomial time.

Polynomial-time reduction

Polynomial-time reducibility between decision problems Π and Π'

We say that Π reduces to Π' in deterministic polynomial-time, symbolized as $\Pi \leq_P \Pi'$, if there exists a deterministic polynomial time algorithm A that takes I, an instance of Π, as input and transforms it into I', an instance of Π' such that I is a yes-instance if and only if I' is an yes-instance.

An important offshoot of the above definition

Suppose $\Pi \leq_P \Pi'$. If Π' can be solved in polynomial time, then Π can also be solved in polynomial time.

The contrapositive statement

Suppose $\Pi \leq_P \Pi'$. If Π cannot be solved in polynomial time, then Π' cannot be solved in polynomial time.

Polynomial-time reduction

Relative hardness among problems

Suppose $\Pi \leq_P \Pi'$. Then Π' is a harder problem than Π. Precisely, Π' is at least as hard as Π.

Polynomial reductions

- You have already done a polynomial reduction. Do you remember?

Polynomial reductions

- You have already done a polynomial reduction. Do you remember?
- Bipartite Matching $\leq_P$ Max Flow

Polynomial reductions

- You have already done a polynomial reduction. Do you remember?
- Bipartite Matching $\leq_P$ Max Flow
- Let us do some more.

Polynomial reducibility is transitive

Transitivity of reductions

If $\Pi_1 \leq_P \Pi_2$, and $\Pi_2 \leq_P \Pi_3$, then $\Pi_1 \leq_P \Pi_3$.

SAT and 3SAT

- Given a boolean formula f, we say that it is in CNF (Conjunctive Normal Form), if it is the conjunction ($\wedge$) of clauses.

SAT and 3SAT

- Given a boolean formula f, we say that it is in CNF (Conjunctive Normal Form), if it is the conjunction ($\wedge$) of clauses.
- A clause is the disjunction ($\vee$) of literals. A literal is a boolean variable or its negation.

SAT and 3SAT

- Given a boolean formula f, we say that it is in CNF (Conjunctive Normal Form), if it is the conjunction ($\wedge$) of clauses.
- A clause is the disjunction ($\vee$) of literals. A literal is a boolean variable or its negation.
- A formula is said to be satisfiable if there is a truth assignment to its variables that makes it TRUE.

SAT and 3SAT

- Given a boolean formula f, we say that it is in CNF (Conjunctive Normal Form), if it is the conjunction ($\wedge$) of clauses.
- A clause is the disjunction ($\vee$) of literals. A literal is a boolean variable or its negation.
- A formula is said to be satisfiable if there is a truth assignment to its variables that makes it TRUE.
- An example. $f = (x_1 \vee x_2) \wedge (\overline{x_1} \vee x_3 \vee x_4 \vee \overline{x_5}) \wedge (x_1 \vee \overline{x_3} \vee x_4)$. If x_1 and x_3 are set to TRUE, then f is TRUE.

SAT and 3SAT

- Given a boolean formula f, we say that it is in CNF (Conjunctive Normal Form), if it is the conjunction ($\wedge$) of clauses.
- A clause is the disjunction ($\vee$) of literals. A literal is a boolean variable or its negation.
- A formula is said to be satisfiable if there is a truth assignment to its variables that makes it TRUE.
- An example. $f = (x_1 \vee x_2) \wedge (\overline{x_1} \vee x_3 \vee x_4 \vee \overline{x_5}) \wedge (x_1 \vee \overline{x_3} \vee x_4)$. If x_1 and x_3 are set to TRUE, then f is TRUE.
- If the number of literals per clause is 3, then the boolean formula is a 3SAT formula.

SAT $\leq_P$ 3SAT

Lemma

SAT $\leq_P$ 3SAT

Proof

- Map a CNF formula ϕ into a 3CNF formula ψ such that ψ is satisfiable iff ϕ is.

SAT $\leq_P$ 3SAT

Lemma

SAT $\leq_P$ 3SAT

Proof

- Map a CNF formula ϕ into a 3CNF formula ψ such that ψ is satisfiable iff ϕ is.
- Any clause C of size $k > 3$ can be changed to an equivalent pair of clauses C_1 of size $k - 1$ and C_2 of size 3 by using an additional auxiliary variable.

SAT $\leq_P$ 3SAT

Lemma

SAT $\leq_P$ 3SAT

Proof

- Map a CNF formula ϕ into a 3CNF formula ψ such that ψ is satisfiable iff ϕ is.
- Any clause C of size $k > 3$ can be changed to an equivalent pair of clauses C_1 of size $k-1$ and C_2 of size 3 by using an additional auxiliary variable.
- Say $C = \overline{x_1} \vee x_2 \vee x_3 \vee \overline{x_4}$. Let $C_1 = \overline{x_1} \vee x_2 \vee z$ and $C_2 = x_3 \vee \overline{x_4} \vee \overline{z}$. Clearly, if C is true, then there is an assignment to z that satisfies both C_1 and C_2 and vice versa.

Outline

Polynomial reductions: SAT $\leq_P$ CLIQUE

CLIQUE

Given an undirected graph $G = (V, E)$ and a positive integer k, does G contain a clique of size k?
(A clique in G of size k is a complete subgraph of G on k vertices.)

Polynomial reductions: SAT $\leq_P$ CLIQUE

CLIQUE

Given an undirected graph $G = (V, E)$ and a positive integer k, does G contain a clique of size k?
(A clique in G of size k is a complete subgraph of G on k vertices.)

The Polynomial Reduction

Polynomial reductions: SAT $\leq_P$ CLIQUE

CLIQUE

Given an undirected graph $G = (V, E)$ and a positive integer k, does G contain a clique of size k?
(A clique in G of size k is a complete subgraph of G on k vertices.)

The Polynomial Reduction

- Given an instance of SAT $f = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ with m clauses and n Boolean variables $x_1, \ldots, x_n$,

Polynomial reductions: SAT $\leq_P$ CLIQUE

CLIQUE

Given an undirected graph $G = (V, E)$ and a positive integer k, does G contain a clique of size k?
(A clique in G of size k is a complete subgraph of G on k vertices.)

The Polynomial Reduction

- Given an instance of SAT $f = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ with m clauses and n Boolean variables $x_1, \ldots, x_n$,
- we construct a graph $G = (V, E)$, where $V \equiv$ all occurrences of the literals in f and
  $E = \{(x_i, x_j) \mid x_i \text{ and } x_j \text{ are in two clauses and } x_i \neq x_j\}$.

Polynomial reductions: SAT $\leq_P$ CLIQUE

CLIQUE

Given an undirected graph $G = (V, E)$ and a positive integer k, does G contain a clique of size k?
(A clique in G of size k is a complete subgraph of G on k vertices.)

The Polynomial Reduction

- Given an instance of SAT $f = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ with m clauses and n Boolean variables $x_1, \ldots, x_n$,
- we construct a graph $G = (V, E)$, where $V \equiv$ all occurrences of the literals in f and
  $E = \{(x_i, x_j) \mid x_i \text{ and } x_j \text{ are in two clauses and } x_i \neq x_j\}$.
- The construction can be done in polynomial time.

SAT $\leq_P$ CLIQUE: An example

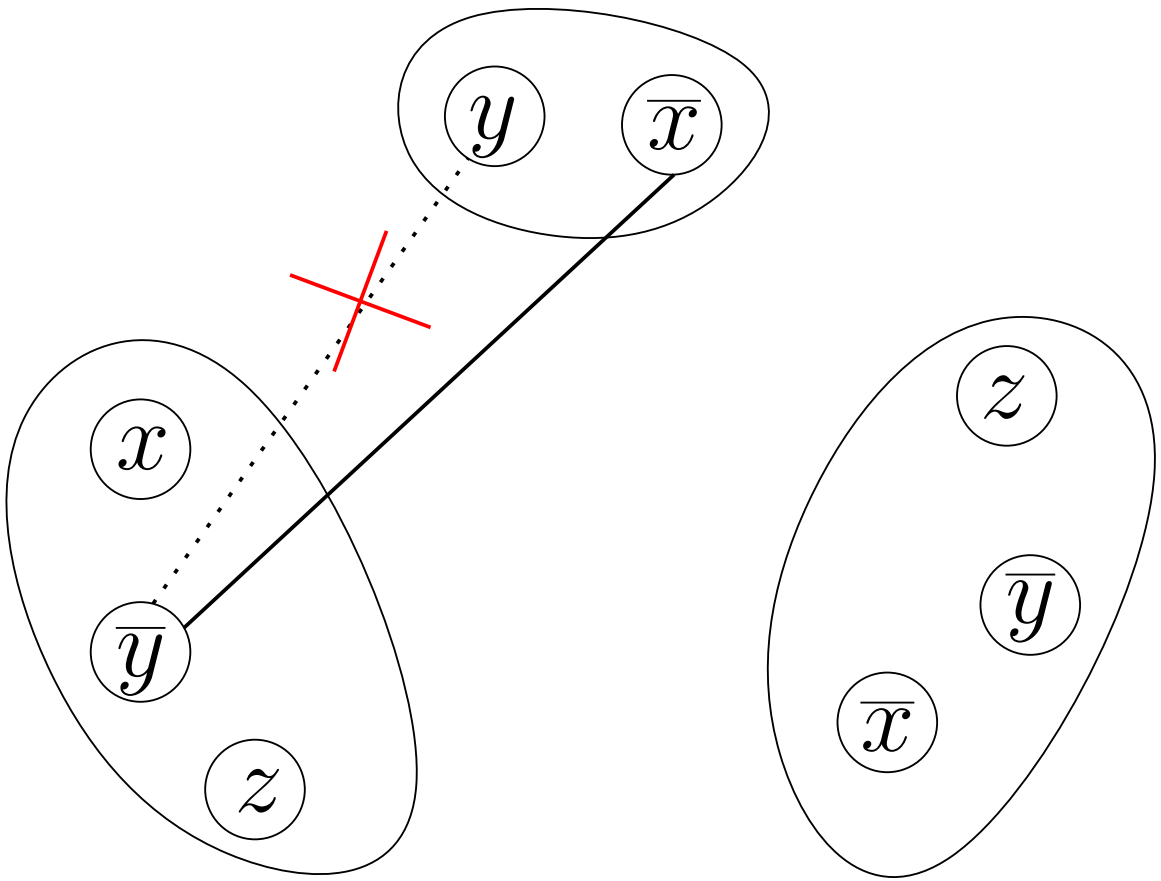
- A SAT example $f = (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee y) \wedge (\overline{x} \vee \overline{y} \vee z)$.

SAT $\leq_P$ CLIQUE: An example

- A SAT example $f = (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee y) \wedge (\overline{x} \vee \overline{y} \vee z)$.
- Construct G, where $V \equiv$ all occurrences of literals in f and $E = \{(x_i, x_j) \mid x_i \text{ and } x_j \text{ are in two diff. clauses and } x_i \neq x_j\}$.

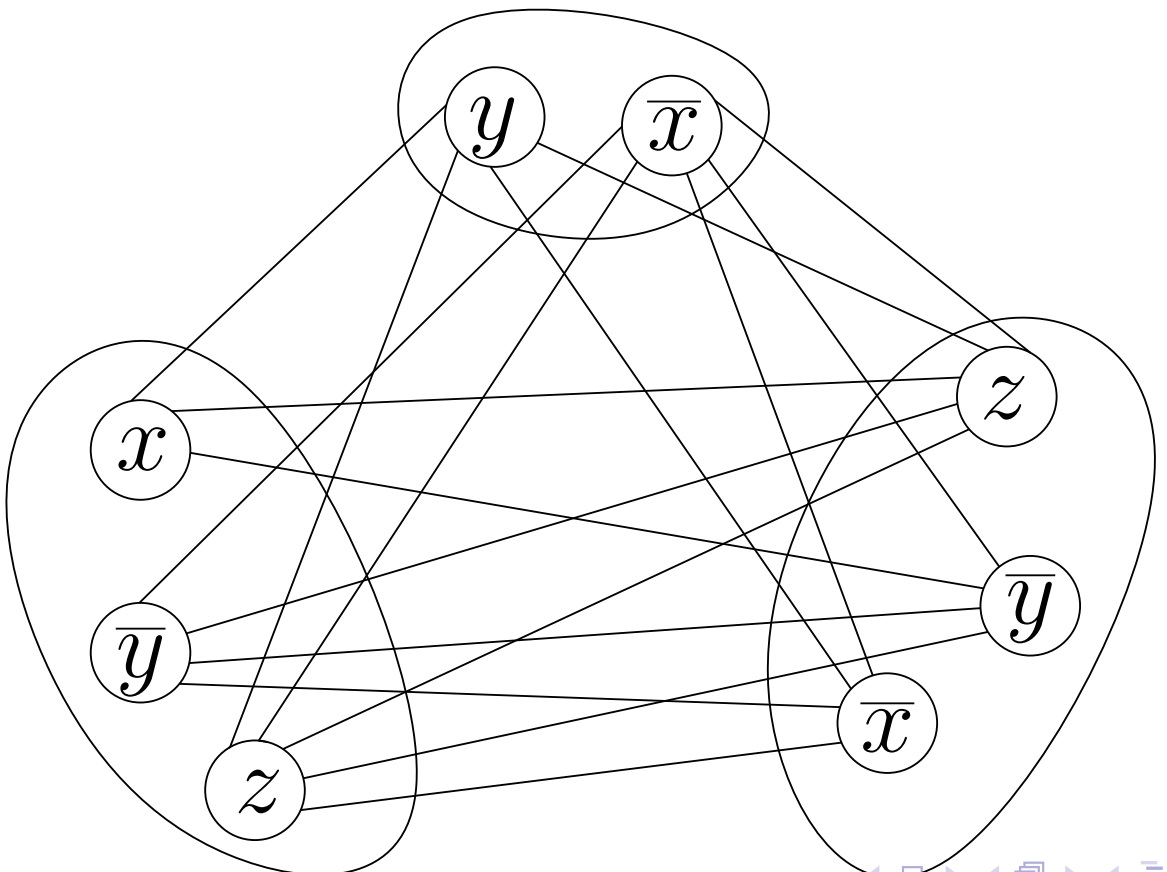
SAT $\leq_P$ CLIQUE: An example

- A SAT example $f = (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee y) \wedge (\overline{x} \vee \overline{y} \vee z)$.
- Construct G, where $V \equiv$ all occurrences of literals in f and $E = \{(x_i, x_j) \mid x_i \text{ and } x_j \text{ are in two diff. clauses and } x_i \neq x_j\}$.



SAT $\leq_P$ CLIQUE: An example

- A SAT example $f = (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee y) \wedge (\overline{x} \vee \overline{y} \vee z)$.
- Construct G, where $V \equiv$ all occurrences of literals in f and $E = \{(x_i, x_j) \mid x_i \text{ and } x_j \text{ are in two diff. clauses and } x_i \neq x_j\}$.



SAT $\leq_P$ CLIQUE

Lemma

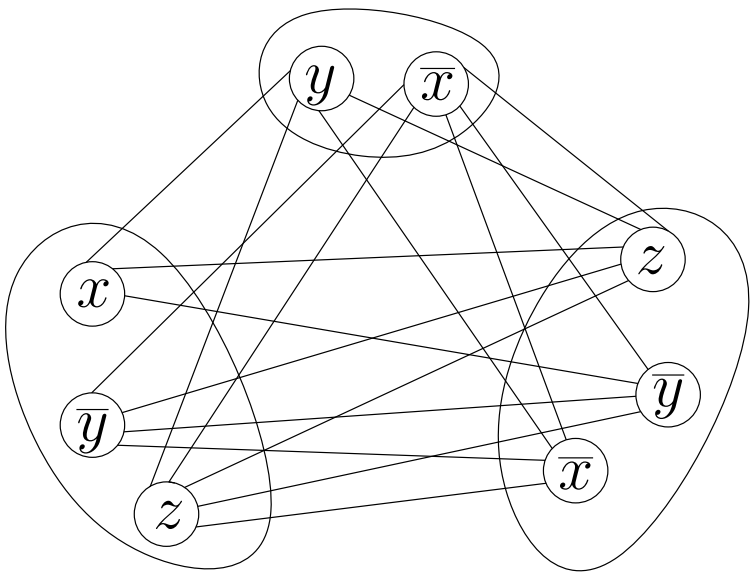
f is satisfiable $\implies$ G has a clique of size m.

SAT $\leq_P$ CLIQUE

Lemma

f is satisfiable $\implies$ G has a clique of size m.

Proof



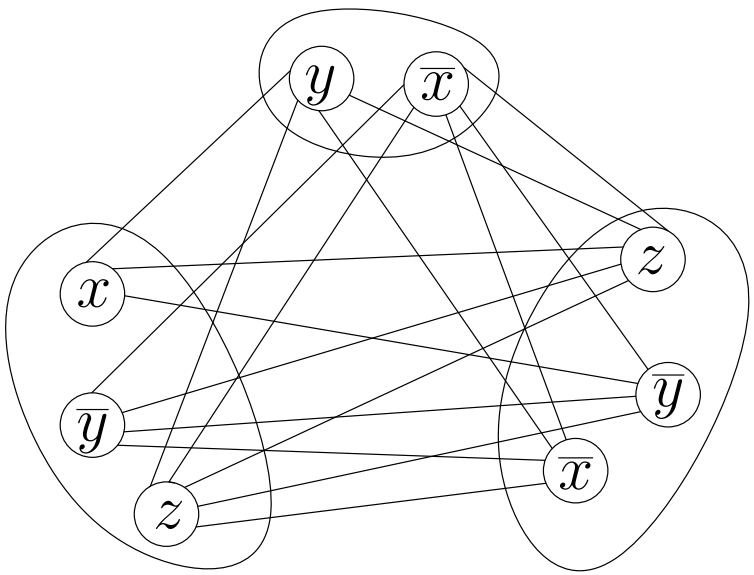
SAT $\leq_P$ CLIQUE

Lemma

f is satisfiable $\implies$ G has a clique of size m.

Proof

- f is satisfiable $\implies$ there is a noncontradictory assignment of TRUE to m literals in m different clauses.



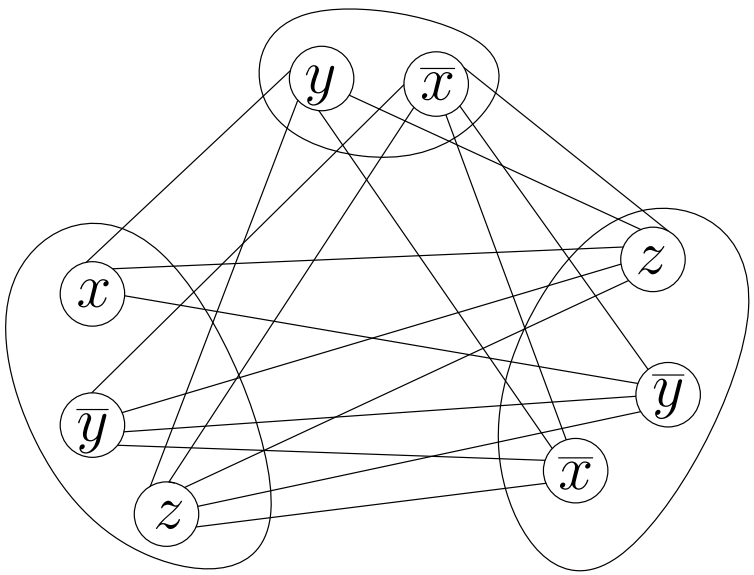
SAT $\leq_P$ CLIQUE

Lemma

f is satisfiable $\implies$ G has a clique of size m.

Proof

- f is satisfiable $\implies$ there is a noncontradictory assignment of TRUE to m literals in m different clauses.
- By construction, there are edges between all pairs of these m different vertices, and hence, a clique of size m.



SAT $\leq_P$ CLIQUE

Lemma

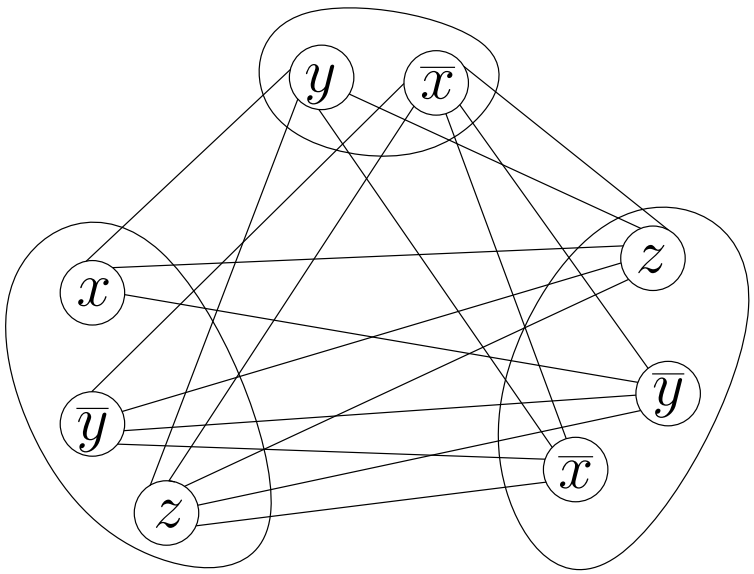
G has a clique of size $m \implies f$ is satisfiable.

SAT $\leq_P$ CLIQUE

Lemma

G has a clique of size $m \implies f$ is satisfiable.

Proof



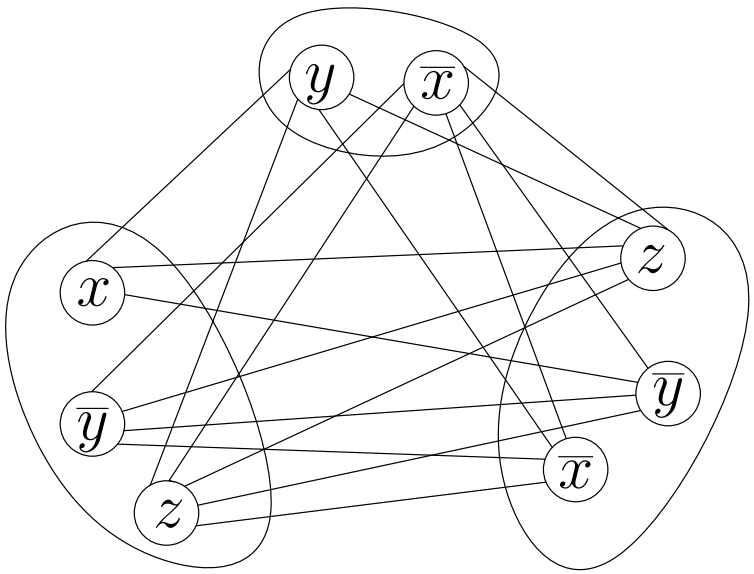
SAT $\leq_P$ CLIQUE

Lemma

G has a clique of size $m \implies f$ is satisfiable.

Proof

- G has a clique of size $m \implies$ assignment of TRUE to m literals in m different clauses, and hence, f is satisfiable.



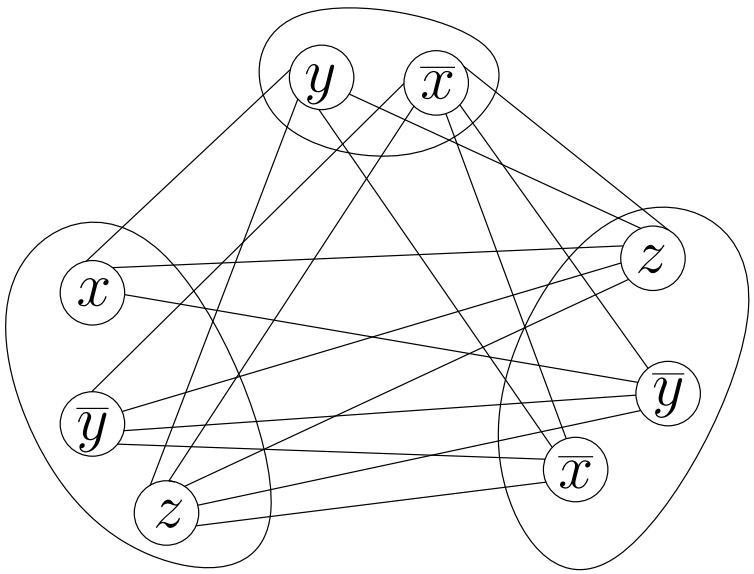
SAT $\leq_P$ CLIQUE

Lemma

G has a clique of size $m \implies f$ is satisfiable.

Proof

- G has a clique of size $m \implies$ assignment of TRUE to m literals in m different clauses, and hence, f is satisfiable.



Theorem

f is satisfiable if and only if G has a clique of size m.

VERTEX COVER

VERTEX COVER

Given an undirected graph $G = (V, E)$ and a positive integer k, is there a subset $C \subseteq V$ of size k such that each edge in E is incident to at least one vertex in C?
(Such a set C is a vertex cover of G.)

Exercise

For a clique of n vertices, what is the size of minimum vertex cover?

Polynomial reductions: SAT $\leq_P$ VERTEX COVER

The Polynomial Reduction

Polynomial reductions: SAT $\leq_P$ VERTEX COVER

The Polynomial Reduction

- Given an instance I of SAT $f = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ with m clauses and n Boolean variables $x_1, \ldots, x_n$,

Polynomial reductions: SAT $\leq_P$ VERTEX COVER

The Polynomial Reduction

- Given an instance I of SAT $f = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ with m clauses and n Boolean variables $x_1, \ldots, x_n$,
- we construct an instance I' of vertex cover as follows

Polynomial reductions: SAT $\leq_P$ VERTEX COVER

The Polynomial Reduction

- Given an instance I of SAT $f = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ with m clauses and n Boolean variables $x_1, \ldots, x_n$,
- we construct an instance I' of vertex cover as follows
- for each Boolean variable $x_i \in f$, G contains a pair of vertices x_i and $\overline{x_i}$ joined by an edge.

Polynomial reductions: SAT $\leq_P$ VERTEX COVER

The Polynomial Reduction

- Given an instance I of SAT $f = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ with m clauses and n Boolean variables $x_1, \ldots, x_n$,
- we construct an instance I' of vertex cover as follows
- for each Boolean variable $x_i \in f$, G contains a pair of vertices x_i and $\overline{x_i}$ joined by an edge.
- for each clause C_j containing n_j literals, G contains a clique C_j of size n_j.

Polynomial reductions: SAT $\leq_P$ VERTEX COVER

The Polynomial Reduction

- Given an instance I of SAT $f = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ with m clauses and n Boolean variables $x_1, \ldots, x_n$,
- we construct an instance I' of vertex cover as follows
- for each Boolean variable $x_i \in f$, G contains a pair of vertices x_i and $\overline{x_i}$ joined by an edge.
- for each clause C_j containing n_j literals, G contains a clique C_j of size n_j.
- for each vertex $w \in C_j$, there is an edge connecting w to its corresponding literal in the vertex pairs $(x_i, \overline{x_i})$ constructed earlier. These edges are called connection edges.

Polynomial reductions: SAT $\leq_P$ VERTEX COVER

The Polynomial Reduction

- Given an instance I of SAT $f = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ with m clauses and n Boolean variables $x_1, \ldots, x_n$,
- we construct an instance I' of vertex cover as follows
- for each Boolean variable $x_i \in f$, G contains a pair of vertices x_i and $\overline{x_i}$ joined by an edge.
- for each clause C_j containing n_j literals, G contains a clique C_j of size n_j.
- for each vertex $w \in C_j$, there is an edge connecting w to its corresponding literal in the vertex pairs $(x_i, \overline{x_i})$ constructed earlier. These edges are called connection edges.
- Let $k = n + \sum_{j=1}^{m}(n_j - 1)$. k is a part of the reduction.

Polynomial reductions: SAT $\leq_P$ VERTEX COVER

The Polynomial Reduction

- Given an instance I of SAT $f = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ with m clauses and n Boolean variables $x_1, \ldots, x_n$,
- we construct an instance I' of vertex cover as follows
- for each Boolean variable $x_i \in f$, G contains a pair of vertices x_i and $\overline{x_i}$ joined by an edge.
- for each clause C_j containing n_j literals, G contains a clique C_j of size n_j.
- for each vertex $w \in C_j$, there is an edge connecting w to its corresponding literal in the vertex pairs $(x_i, \overline{x_i})$ constructed earlier. These edges are called connection edges.
- Let $k = n + \sum_{j=1}^{m}(n_j - 1)$. k is a part of the reduction.
- The construction can be done in polynomial time.

SAT $\leq_P$ VERTEX COVER: An example

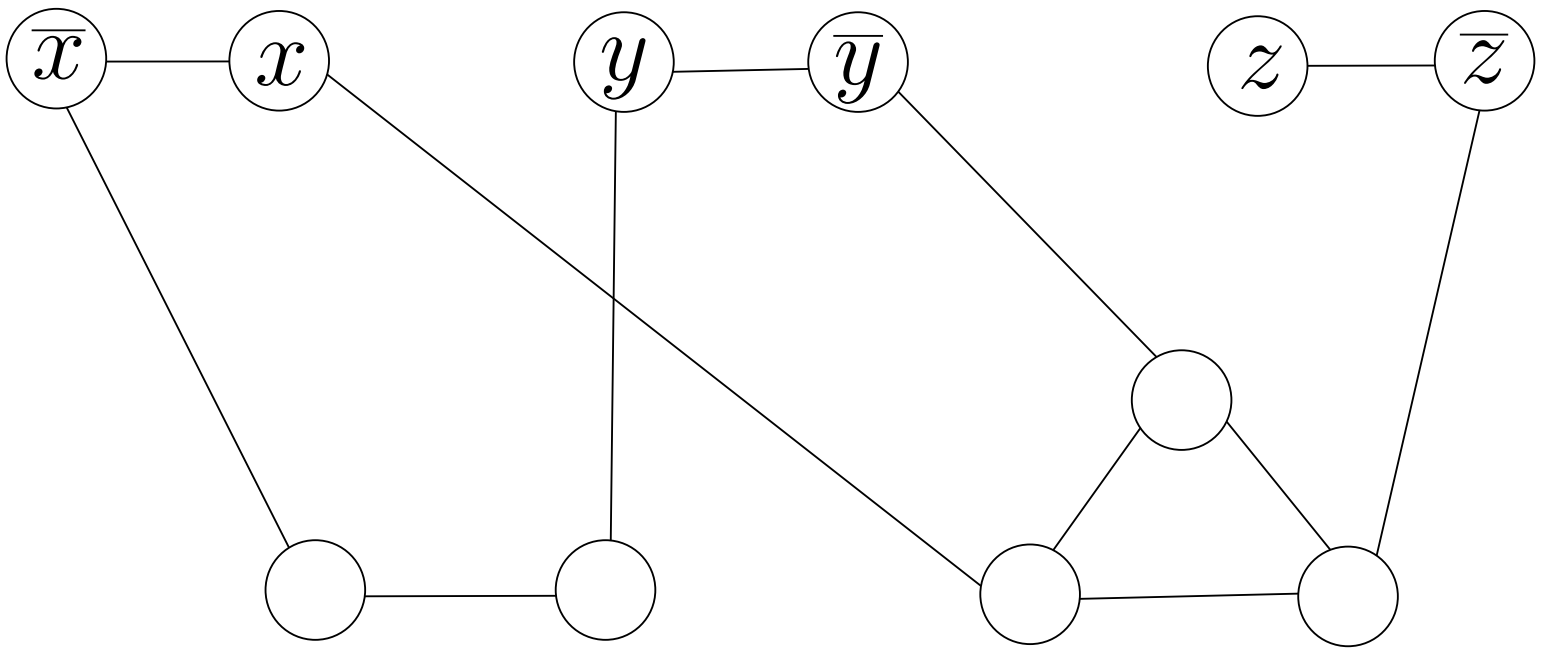
- A SAT example $f = (x \vee \overline{y} \vee \overline{z}) \wedge (\overline{x} \vee y)$.

SAT $\leq_P$ VERTEX COVER: An example

- A SAT example $f = (x \vee \overline{y} \vee \overline{z}) \wedge (\overline{x} \vee y)$.
- Construct G as described above.

SAT $\leq_P$ VERTEX COVER: An example

- A SAT example $f = (x \vee \overline{y} \vee \overline{z}) \wedge (\overline{x} \vee y)$.
- Construct G as described above.



SAT $\leq_P$ VERTEX COVER (VC)

Lemma

f is satisfiable $\implies$ G has a vertex cover (VC) of size k.

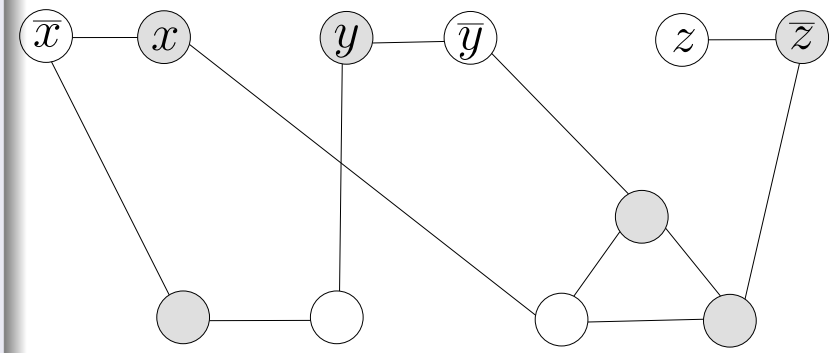
SAT $\leq_P$ VERTEX COVER (VC)

Lemma

f is satisfiable $\implies$ G has a vertex cover (VC) of size k.

Proof

- If x_i is assigned TRUE, add vertex x_i to the VC; otherwise, add $\overline{x_i}$ to the VC.



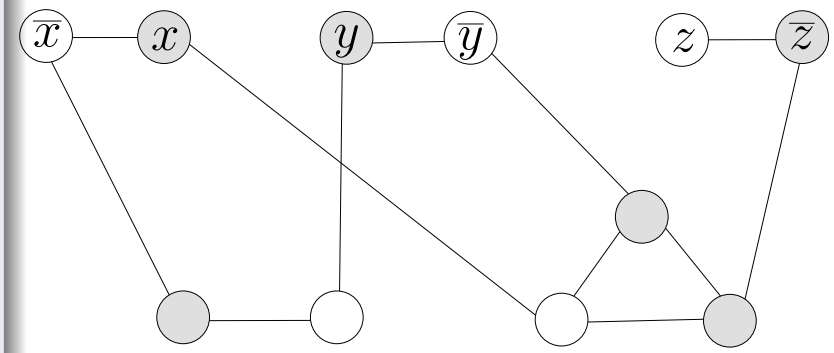
SAT $\leq_P$ VERTEX COVER (VC)

Lemma

f is satisfiable $\implies$ G has a vertex cover (VC) of size k.

Proof

- If x_i is assigned TRUE, add vertex x_i to the VC; otherwise, add $\overline{x_i}$ to the VC.
- Since f is satisfiable, in each clique C_j there is a vertex w whose corresponding literal is TRUE; so a connection edge is covered. We add other $n_j - 1$ vertices in each C_j to the VC.



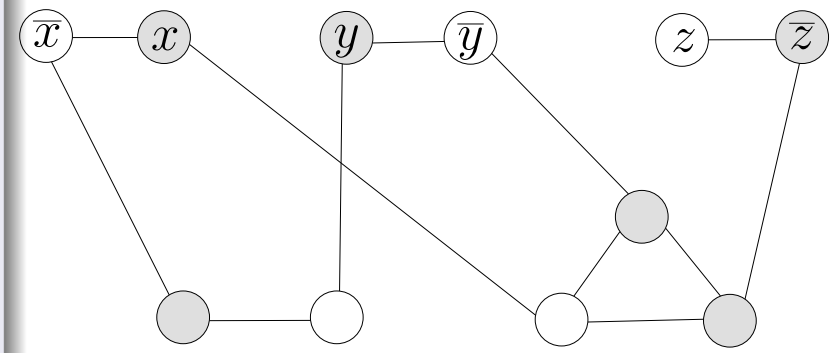
SAT $\leq_P$ VERTEX COVER (VC)

Lemma

f is satisfiable $\implies$ G has a vertex cover (VC) of size k.

Proof

- If x_i is assigned TRUE, add vertex x_i to the VC; otherwise, add $\overline{x_i}$ to the VC.
- Since f is satisfiable, in each clique C_j there is a vertex w whose corresponding literal is TRUE; so a connection edge is covered. We add other $n_j - 1$ vertices in each C_j to the VC.
- The size of the VC is $k = n + \sum_{j=1}^{m}(n_j - 1)$.



SAT $\leq_P$ VC

Lemma

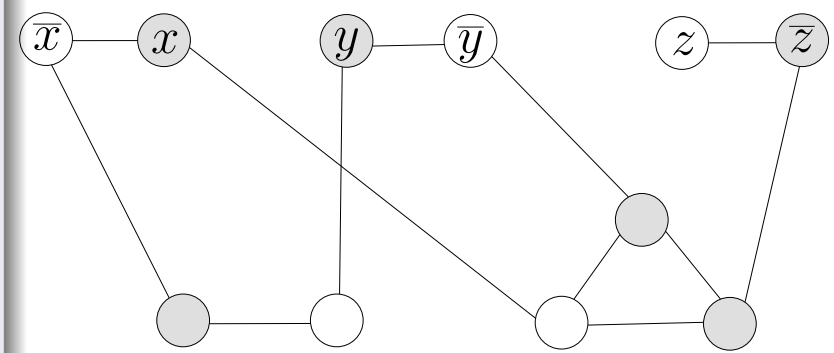
G has a VC of size $k \implies f$ is satisfiable.

SAT $\leq_P$ VC

Lemma

G has a VC of size $k \implies f$ is satisfiable.

Proof



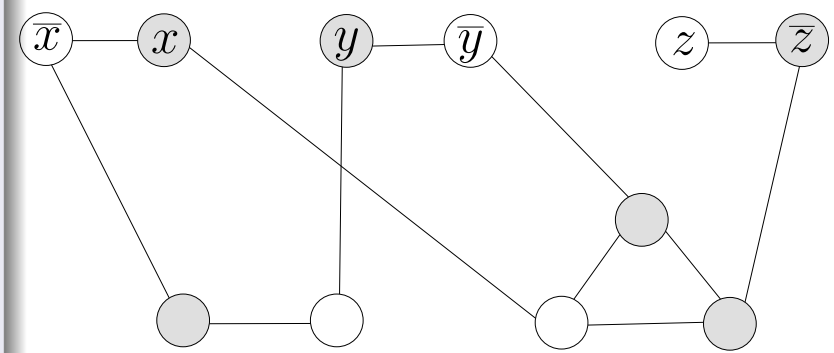
SAT $\leq_P$ VC

Lemma

G has a VC of size $k \implies f$ is satisfiable.

Proof

- At least one vertex of each edge $(x_i, \overline{x_i})$ must be in the VC. We are left with $k - n = \sum_{j=1}^{m}(n_j - 1)$ vertices.



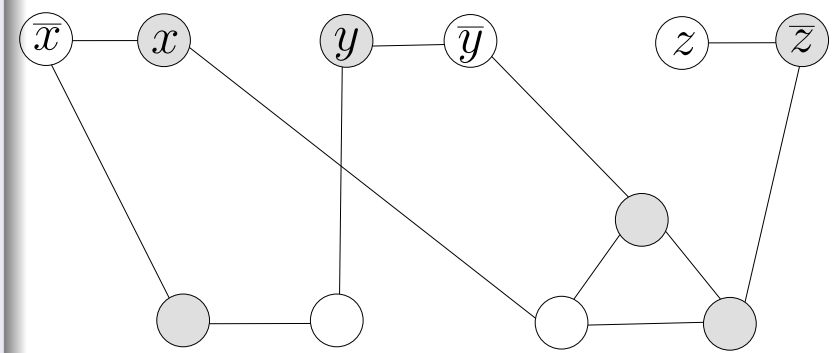
SAT $\leq_P$ VC

Lemma

G has a VC of size $k \implies f$ is satisfiable.

Proof

- At least one vertex of each edge $(x_i, \overline{x_i})$ must be in the VC. We are left with $k - n = \sum_{j=1}^{m}(n_j - 1)$ vertices.
- For each clique C_j, pick the rest $n_j - 1$ vertices in the VC.



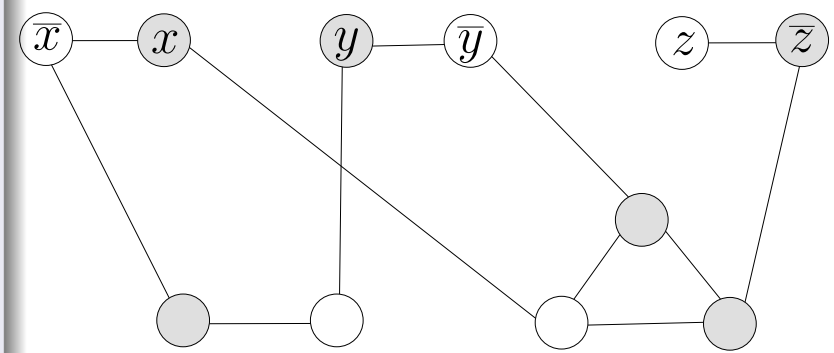
SAT $\leq_P$ VC

Lemma

G has a VC of size $k \implies f$ is satisfiable.

Proof

- At least one vertex of each edge $(x_i, \overline{x_i})$ must be in the VC. We are left with $k - n = \sum_{j=1}^{m}(n_j - 1)$ vertices.
- For each clique C_j, pick the rest $n_j - 1$ vertices in the VC.
- For each vertex x_i, if it is in the VC, let x_i be TRUE; else x_i be FALSE. Thus for each clique, there is a vertex having TRUE.



SAT $\leq_P$ VC

Theorem

f is satisfiable if and only if G has a VC of size k.

Outline

Understanding Certifier

Decision problem and strings

We identify a decision problem L with the set of strings $x \in \{0,1\}^*$ on which the answer is YES.

Understanding Certifier

Decision problem and strings

We identify a decision problem L with the set of strings $x \in \{0,1\}^*$ on which the answer is YES.

Definition: The Class **NP**

A decision problem L is in **NP** if $\exists$ a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time algorithm M such that for every string x,

$$x \in L \iff \exists \text{ a short certificate } u \text{ such that } M(x,u) = \text{YES}$$

u is a short certificate if $|u| = p(|x|)$.
If $x \in L$ and $u \in \{0,1\}^{p(|x|)}$ satisfy $M(x,u) =$ YES, then we call u a certificate for x (w.r.t. problem L and algorithm M).

Understanding Certifier

Decision problem and strings

We identify a decision problem L with the set of strings $x \in \{0,1\}^*$ on which the answer is YES.

Definition: The Class **NP**

A decision problem L is in **NP** if $\exists$ a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time algorithm M such that for every string x,

$$x \in L \iff \exists \text{ a short certificate } u \text{ such that } M(x,u) = \text{YES}$$

u is a short certificate if $|u| = p(|x|)$.
If $x \in L$ and $u \in \{0,1\}^{p(|x|)}$ satisfy $M(x,u) =$ YES, then we call u a certificate for x (w.r.t. problem L and algorithm M).

Examples

Traveling Salesperson, Subset sum, IP, LP, Graph Isomorphism, etc.

Understanding Certifier

An Efficient Certifier

M is an efficient certifier for L if the following holds:

Understanding Certifier

An Efficient Certifier

M is an efficient certifier for L if the following holds:

- M is a polynomial time algorithm (alternately, TM) that takes two input arguments x, u.

Understanding Certifier

An Efficient Certifier

M is an efficient certifier for L if the following holds:

- M is a polynomial time algorithm (alternately, TM) that takes two input arguments x, u.
- There is a polynomial function p so that for every string x, we have $x \in L$ if and only if $\exists$ a string u such that $|u| \leq p(|x|)$ and $M(x, u) = 1$.

Understanding Certifier

An Efficient Certifier

M is an efficient certifier for L if the following holds:

- M is a polynomial time algorithm (alternately, TM) that takes two input arguments x, u.
- There is a polynomial function p so that for every string x, we have $x \in L$ if and only if $\exists$ a string u such that $|u| \leq p(|x|)$ and $M(x, u) = 1$.

A Managerial View of M, an efficient certifier

Understanding Certifier

An Efficient Certifier

M is an efficient certifier for L if the following holds:

- M is a polynomial time algorithm (alternately, TM) that takes two input arguments x, u.
- There is a polynomial function p so that for every string x, we have $x \in L$ if and only if $\exists$ a string u such that $|u| \leq p(|x|)$ and $M(x, u) = 1$.

A Managerial View of M, an efficient certifier

- It will not try to decide whether $x \in L$ on its own.

Understanding Certifier

An Efficient Certifier

M is an efficient certifier for L if the following holds:

- M is a polynomial time algorithm (alternately, TM) that takes two input arguments x, u.
- There is a polynomial function p so that for every string x, we have $x \in L$ if and only if $\exists$ a string u such that $|u| \leq p(|x|)$ and $M(x, u) = 1$.

A Managerial View of M, an efficient certifier

- It will not try to decide whether $x \in L$ on its own.
- It will rather try to efficiently evaluate proposed "proofs" u that $x \in L$ - provided they are not too long.

A Brute Force Algorithm

M's use in solving L

- On an input x, try all strings u, s.t. $|u| \leq p(|x|)$, and see if $M(x, u) = 1$ for any of these strings.

A Brute Force Algorithm

M's use in solving L

- On an input x, try all strings u, s.t. $|u| \leq p(|x|)$, and see if $M(x, u) = 1$ for any of these strings.
- Existence of M does not provide an efficient solver for L.

A Brute Force Algorithm

M's use in solving L

- On an input x, try all strings u, s.t. $|u| \leq p(|x|)$, and see if $M(x, u) = 1$ for any of these strings.
- Existence of M does not provide an efficient solver for L.
- It is upto us to find a string u that will make $M(x, u) = 1$, and there are exponentially many possibilities for u.

The class **NP**

The class **NP**

We define **NP** to be the set of all problems for which there exists an efficient certifier.

The class **NP**

The class **NP**

We define **NP** to be the set of all problems for which there exists an efficient certifier.

Relation between the class **P** and **NP**

$\mathbf{P} \subseteq \mathbf{NP}$

The class **NP**

The class **NP**

We define **NP** to be the set of all problems for which there exists an efficient certifier.

Relation between the class **P** and **NP**

$\mathbf{P} \subseteq \mathbf{NP}$

Relation between the class **P** and **NP**

Does $\mathbf{P} = \mathbf{NP}$? Or, is $\mathbf{P} \subset \mathbf{NP}$?

Outline

NP-Complete Problems

NP-complete problems

- We are interested in defining the hardest problems in the class **NP**.

NP-Complete Problems

NP-complete problems

- We are interested in defining the hardest problems in the class **NP**.
- We use the notion of polynomial reducibility to do it.

NP-Complete Problems

NP-complete problems

- We are interested in defining the hardest problems in the class **NP**.
- We use the notion of polynomial reducibility to do it.
- A problem L is **NP**-complete if it satisfies the following:

NP-Complete Problems

NP-complete problems

- We are interested in defining the hardest problems in the class **NP**.
- We use the notion of polynomial reducibility to do it.
- A problem L is **NP**-complete if it satisfies the following:
  - $L \in$ **NP** and

NP-Complete Problems

NP-complete problems

- We are interested in defining the hardest problems in the class **NP**.
- We use the notion of polynomial reducibility to do it.
- A problem L is **NP**-complete if it satisfies the following:
  - $L \in \mathbf{NP}$ and
  - $\forall L' \in \mathbf{NP}$, $L' \leq_P L$.

NP-Complete Problems

NP-complete problems

Suppose L is an **NP**-complete problem. Then L is solvable in polynomial time if and only if $\mathbf{P} = \mathbf{NP}$.

NP-Complete Problems

NP-complete problems

Suppose L is an **NP**-complete problem. Then L is solvable in polynomial time if and only if $\mathbf{P} = \mathbf{NP}$.

Proof

NP-Complete Problems

NP-complete problems

Suppose L is an **NP**-complete problem. Then L is solvable in polynomial time if and only if $\mathbf{P} = \mathbf{NP}$.

Proof

- If $\mathbf{P} = \mathbf{NP}$, then L can be solved in polynomial time.

NP-Complete Problems

NP-complete problems

Suppose L is an **NP**-complete problem. Then L is solvable in polynomial time if and only if $\mathbf{P} = \mathbf{NP}$.

Proof

- If $\mathbf{P} = \mathbf{NP}$, then L can be solved in polynomial time.
- Suppose L can be solved in polynomial time. Now, fix a problem $L' \in \mathbf{NP}$.

NP-Complete Problems

NP-complete problems

Suppose L is an **NP**-complete problem. Then L is solvable in polynomial time if and only if $\mathbf{P} = \mathbf{NP}$.

Proof

- If $\mathbf{P} = \mathbf{NP}$, then L can be solved in polynomial time.
- Suppose L can be solved in polynomial time. Now, fix a problem $L' \in \mathbf{NP}$.
- As L is **NP**-complete, $L' \leq_P L$. So, L' can also be solved in polynomial time and hence, $L' \in \mathbf{P}$ and $\mathbf{P} \subseteq \mathbf{NP}$.

NP-Complete Problems

NP-complete problems

Suppose L is an **NP**-complete problem. Then L is solvable in polynomial time if and only if $\mathbf{P} = \mathbf{NP}$.

Proof

- If $\mathbf{P} = \mathbf{NP}$, then L can be solved in polynomial time.
- Suppose L can be solved in polynomial time. Now, fix a problem $L' \in \mathbf{NP}$.
- As L is **NP**-complete, $L' \leq_P L$. So, L' can also be solved in polynomial time and hence, $L' \in \mathbf{P}$ and $\mathbf{P} \subseteq \mathbf{NP}$.
- With $\mathbf{P} \subseteq \mathbf{NP}$ already known, we have $\mathbf{P} = \mathbf{NP}$.

Proving new problems **NP**-complete

To prove a new problem L to be NP-complete,

Proving new problems **NP**-complete

To prove a new problem L to be NP-complete,

- polynomially reduce all problems in the class **NP** to L.

Proving new problems **NP**-complete

To prove a new problem L to be NP-complete,

- polynomially reduce all problems in the class **NP** to L.
- Is it feasible to do such a thing??

Proving new problems **NP**-complete

To prove a new problem L to be **NP**-complete,

- polynomially reduce all problems in the class **NP** to L.
- Is it feasible to do such a thing??

Proving new problems **NP**-complete

If L' is an **NP**-complete problem, and a problem $L \in \mathbf{NP}$ with the property that $L' \leq_P L$, then L is **NP**-complete.

Proving new problems **NP**-complete

To prove a new problem L to be **NP**-complete,

- polynomially reduce all problems in the class **NP** to L.
- Is it feasible to do such a thing??

Proving new problems **NP**-complete

If L' is an **NP**-complete problem, and a problem $L \in$ **NP** with the property that $L' \leq_P L$, then L is **NP**-complete.

Proof

Proving new problems **NP**-complete

To prove a new problem L to be **NP**-complete,

- polynomially reduce all problems in the class **NP** to L.
- Is it feasible to do such a thing??

Proving new problems **NP**-complete

If L' is an **NP**-complete problem, and a problem $L \in \mathbf{NP}$ with the property that $L' \leq_P L$, then L is **NP**-complete.

Proof

- Fix any problem $Z \in \mathbf{NP}$. As L' is **NP**-complete, $Z \leq_P L'$.

Proving new problems **NP**-complete

To prove a new problem L to be **NP**-complete,

- polynomially reduce all problems in the class **NP** to L.
- Is it feasible to do such a thing??

Proving new problems **NP**-complete

If L' is an **NP**-complete problem, and a problem $L \in \mathbf{NP}$ with the property that $L' \leq_P L$, then L is **NP**-complete.

Proof

- Fix any problem $Z \in \mathbf{NP}$. As L' is **NP**-complete, $Z \leq_P L'$.
- Now, use transitivity to show $Z \leq_P L' \leq_P L$.

Proving new problems **NP**-complete

To prove a new problem L to be **NP**-complete,

- polynomially reduce all problems in the class **NP** to L.
- Is it feasible to do such a thing??

Proving new problems **NP**-complete

If L' is an **NP**-complete problem, and a problem $L \in$ **NP** with the property that $L' \leq_P L$, then L is **NP**-complete.

Proof

- Fix any problem $Z \in$ **NP**. As L' is **NP**-complete, $Z \leq_P L'$.
- Now, use transitivity to show $Z \leq_P L' \leq_P L$.

The first **NP**-complete problem

How do you get the first such problem Z? Cook Levin Theorem shows SAT to be such a problem.

SAT **NP**-complete

SAT is **NP**-complete.

Basic strategy for proving a problem L to be **NP**-complete

Basic strategy for proving a problem L to be **NP**-complete

- Prove that $L \in \mathbf{NP}$.

Basic strategy for proving a problem L to be **NP**-complete

- Prove that $L \in \mathbf{NP}$.
- Choose a problem L' that is known to be **NP**-complete.

Basic strategy for proving a problem L to be **NP**-complete

- Prove that $L \in \mathbf{NP}$.
- Choose a problem L' that is known to be **NP**-complete.
- Prove that $L' \leq_P L$. Elaborating further, consider an arbitrary instance $x_{L'}$ of L' and show how to construct, in polynomial time, an instance x_L of L that satisfies the following properties:

Basic strategy for proving a problem L to be **NP**-complete

- Prove that $L \in \mathbf{NP}$.
- Choose a problem L' that is known to be **NP**-complete.
- Prove that $L' \leq_P L$. Elaborating further, consider an arbitrary instance $x_{L'}$ of L' and show how to construct, in polynomial time, an instance x_L of L that satisfies the following properties:
  - If $x_{L'}$ is a YES instance of L', then x_L is a YES instance of L;

Basic strategy for proving a problem L to be **NP**-complete

- Prove that $L \in \mathbf{NP}$.
- Choose a problem L' that is known to be **NP**-complete.
- Prove that $L' \leq_P L$. Elaborating further, consider an arbitrary instance $x_{L'}$ of L' and show how to construct, in polynomial time, an instance x_L of L that satisfies the following properties:
  - If $x_{L'}$ is a YES instance of L', then x_L is a YES instance of L;
  - If x_L is a YES instance of L, then $x_{L'}$ is a YES instance of L';

Basic strategy for proving a problem L to be **NP**-complete

- Prove that $L \in \mathbf{NP}$.
- Choose a problem L' that is known to be **NP**-complete.
- Prove that $L' \leq_P L$. Elaborating further, consider an arbitrary instance $x_{L'}$ of L' and show how to construct, in polynomial time, an instance x_L of L that satisfies the following properties:
  - If $x_{L'}$ is a YES instance of L', then x_L is a YES instance of L;
  - If x_L is a YES instance of L, then $x_{L'}$ is a YES instance of L';
  - The above two steps ensure that $x_{L'}$ and x_L have the same answer.

NP-complete problems

As SAT is **NP**-complete, and
3SAT $\in$ **NP** and SAT $\leq_P$ 3SAT,
so 3SAT is **NP**-complete.

Similarly, CLIQUE and VERTEX COVER are
NP-complete.

Outline

Approximation ratio

Definition

Approximation ratio

Definition

- An optimal solution $\mathrm{OPT}_\pi(I)$ for an instance I of a minimization [maximization] problem π is a feasible solution that achieves the smallest [largest] objective function value. We would write OPT or $\mathrm{OPT}(I)$ instead of $\mathrm{OPT}_\pi(I)$.

Approximation ratio

Definition

- An optimal solution $\mathrm{OPT}_\pi(I)$ for an instance I of a minimization [maximization] problem π is a feasible solution that achieves the smallest [largest] objective function value. We would write OPT or $\mathrm{OPT}(I)$ instead of $\mathrm{OPT}_\pi(I)$.
- Let π be a minimization [maximization] problem, and let δ be a function, $\delta : \mathbb{Z}^+ \to \mathbb{Q}^+$ with $\delta \geq 1$ [$\delta \leq 1$].

Approximation ratio

Definition

- An optimal solution $\mathrm{OPT}_\pi(I)$ for an instance I of a minimization [maximization] problem π is a feasible solution that achieves the smallest [largest] objective function value. We would write OPT or $\mathrm{OPT}(I)$ instead of $\mathrm{OPT}_\pi(I)$.
- Let π be a minimization [maximization] problem, and let δ be a function, $\delta : \mathbb{Z}^+ \to \mathbb{Q}^+$ with $\delta \geq 1$ [$\delta \leq 1$].
- 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

Approximation ratio

Definition

- An optimal solution $\mathrm{OPT}_\pi(I)$ for an instance I of a minimization [maximization] problem π is a feasible solution that achieves the smallest [largest] objective function value. We would write OPT or $\mathrm{OPT}(I)$ instead of $\mathrm{OPT}_\pi(I)$.
- Let π be a minimization [maximization] problem, and let δ be a function, $\delta : \mathbb{Z}^+ \to \mathbb{Q}^+$ with $\delta \geq 1$ [$\delta \leq 1$].
- 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
- $f_\pi(I, s) \leq \delta(|I|) \cdot \mathrm{OPT}(I)$ [$f_\pi(I, s) \geq \delta(|I|) \cdot \mathrm{OPT}(I)$].

Approximation ratio

Definition

- An optimal solution $\mathrm{OPT}_\pi(I)$ for an instance I of a minimization [maximization] problem π is a feasible solution that achieves the smallest [largest] objective function value. We would write OPT or $\mathrm{OPT}(I)$ instead of $\mathrm{OPT}_\pi(I)$.
- Let π be a minimization [maximization] problem, and let δ be a function, $\delta : \mathbb{Z}^+ \to \mathbb{Q}^+$ with $\delta \geq 1$ [$\delta \leq 1$].
- 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
- $f_\pi(I, s) \leq \delta(|I|) \cdot \mathrm{OPT}(I)$ [$f_\pi(I, s) \geq \delta(|I|) \cdot \mathrm{OPT}(I)$].
- The running time of $\mathcal{A}$ is bounded by a fixed polynomial in $|I|$.

Lower bounding idea for designing approximation algorithm

Lower bounding OPT

Lower bounding idea for designing approximation algorithm

Lower bounding OPT

- We have to fix a bound for δ, i.e. $\frac{f_\pi(I,s)}{OPT} \leq \delta$, but we know nothing of OPT.

Lower bounding idea for designing approximation algorithm

Lower bounding OPT

- We have to fix a bound for δ, i.e. $\frac{f_\pi(I,s)}{OPT} \leq \delta$, but we know nothing of OPT.
- The technique is to lower bound OPT; i.e. find a k_1 such that OPT $\geq k_1$ and find a k_2 such that $f_\pi(I,s) \leq k_2$. Then, $\frac{f_\pi(I,s)}{OPT} \leq \frac{k_2}{k_1} = \delta$.

Approximation algorithm for Vertex Cover

Some background preparation: Matching

Approximation algorithm for Vertex Cover

Some background preparation: Matching

- Given a graph $G = (V, E)$, a subset of the edges $M \subseteq E$ is said to be a matching if no two edges of M share an endpoint.

Approximation algorithm for Vertex Cover

Some background preparation: Matching

- Given a graph $G = (V, E)$, a subset of the edges $M \subseteq E$ is said to be a matching if no two edges of M share an endpoint.
- A matching of maximum cardinality in G is called a maximum matching, and a matching that is maximal under inclusion is called a maximal matching.

Approximation algorithm for Vertex Cover

Some background preparation: Matching

- Given a graph $G = (V, E)$, a subset of the edges $M \subseteq E$ is said to be a matching if no two edges of M share an endpoint.
- A matching of maximum cardinality in G is called a maximum matching, and a matching that is maximal under inclusion is called a maximal matching.
- A maximal matching can be computed in polynomial time by simply greedily picking edges and removing endpoints of picked edges.

Approximation algorithm for Vertex Cover

Some background preparation: Matching

- Given a graph $G = (V, E)$, a subset of the edges $M \subseteq E$ is said to be a matching if no two edges of M share an endpoint.
- A matching of maximum cardinality in G is called a maximum matching, and a matching that is maximal under inclusion is called a maximal matching.
- A maximal matching can be computed in polynomial time by simply greedily picking edges and removing endpoints of picked edges.

Size of maximal matching in G provides a lower bound

Any vertex cover has to pick at least one endpoint of each matched edge.

Approximation algorithm for Vertex Cover

Algorithm for vertex cover

Algorithm:

Approximation algorithm for Vertex Cover

Algorithm for vertex cover

Algorithm:

- Find a maximal matching M in G.

Approximation algorithm for Vertex Cover

Algorithm for vertex cover

Algorithm:

- Find a maximal matching M in G.
- Output both the endpoints of M in G. Let this set of vertices be V', i.e. $|V'| = 2|M|$.

Approximation algorithm for Vertex Cover

Algorithm for vertex cover

Algorithm:

- Find a maximal matching M in G.
- Output both the endpoints of M in G. Let this set of vertices be V', i.e. $|V'| = 2|M|$.

Proof

Approximation algorithm for Vertex Cover

Algorithm for vertex cover

Algorithm:

- Find a maximal matching M in G.
- Output both the endpoints of M in G. Let this set of vertices be V', i.e. $|V'| = 2|M|$.

Proof

- No edge can be left uncovered, otherwise such an edge could have been added to the matching, contradicting its maximality.

Approximation algorithm for Vertex Cover

Algorithm for vertex cover

Algorithm:

- Find a maximal matching M in G.
- Output both the endpoints of M in G. Let this set of vertices be V', i.e. $|V'| = 2|M|$.

Proof

- No edge can be left uncovered, otherwise such an edge could have been added to the matching, contradicting its maximality.
- Because of the lower bound, $|M| \leq$ OPT.

Approximation algorithm for Vertex Cover

Algorithm for vertex cover

Algorithm:

- Find a maximal matching M in G.
- Output both the endpoints of M in G. Let this set of vertices be V', i.e. $|V'| = 2|M|$.

Proof

- No edge can be left uncovered, otherwise such an edge could have been added to the matching, contradicting its maximality.
- Because of the lower bound, $|M| \leq \text{OPT}$.
- Our algorithm returned a vertex cover of size $2|M|$.

Approximation algorithm for Vertex Cover

Algorithm for vertex cover

Algorithm:

- Find a maximal matching M in G.
- Output both the endpoints of M in G. Let this set of vertices be V', i.e. $|V'| = 2|M|$.

Proof

- No edge can be left uncovered, otherwise such an edge could have been added to the matching, contradicting its maximality.
- Because of the lower bound, $|M| \leq$ OPT.
- Our algorithm returned a vertex cover of size $2|M|$.
- Thus, the approximation ratio is 2.

TSP and Hamiltonian cycle

Hamiltonian (HAM) cycle

Given an undirected graph $G = (V, E)$. does it have a cycle that visits each vertex exactly once?

TSP and Hamiltonian cycle

Hamiltonian (HAM) cycle

Given an undirected graph $G = (V, E)$. does it have a cycle that visits each vertex exactly once?

TSP

Given a complete graph with nonnegative edge weights, find a minimum cost cycle visiting every vertex exactly once.

TSP and Hamiltonian cycle

Hamiltonian (HAM) cycle

Given an undirected graph $G = (V, E)$. does it have a cycle that visits each vertex exactly once?

TSP

Given a complete graph with nonnegative edge weights, find a minimum cost cycle visiting every vertex exactly once.

NP-complete

TSP and Hamiltonian cycle are **NP**-complete problems.

Approximation algorithm for (metric) Traveling Salesman Problem

Inapproximability result for TSP

For any polynomial time computable function $f(n)$, TSP cannot be approximated within a factor of $f(n)$, unless **P**=**NP**.

Approximation algorithm for (metric) Traveling Salesman Problem

Inapproximability result for TSP

For any polynomial time computable function $f(n)$, TSP cannot be approximated within a factor of $f(n)$, unless **P**=**NP**.

Proof

Approximation algorithm for (metric) Traveling Salesman Problem

Inapproximability result for TSP

For any polynomial time computable function $f(n)$, TSP cannot be approximated within a factor of $f(n)$, unless **P**=**NP**.

Proof

- Assume, for a contradiction, there is a $f(n)$-factor approximation algorithm, $\mathcal{A}$ for TSP. We show that $\mathcal{A}$ can be used for deciding the HAM cycle problem.

Approximation algorithm for (metric) Traveling Salesman Problem

Inapproximability result for TSP

For any polynomial time computable function $f(n)$, TSP cannot be approximated within a factor of $f(n)$, unless **P**=**NP**.

Proof

- Assume, for a contradiction, there is a $f(n)$-factor approximation algorithm, $\mathcal{A}$ for TSP. We show that $\mathcal{A}$ can be used for deciding the HAM cycle problem.
- Use a HAM $\leq_P$ TSP as follows:

Approximation algorithm for (metric) Traveling Salesman Problem

Inapproximability result for TSP

For any polynomial time computable function $f(n)$, TSP cannot be approximated within a factor of $f(n)$, unless **P**=**NP**.

Proof

- Assume, for a contradiction, there is a $f(n)$-factor approximation algorithm, $\mathcal{A}$ for TSP. We show that $\mathcal{A}$ can be used for deciding the HAM cycle problem.
- Use a HAM $\leq_P$ TSP as follows:
- Assign a weight of 1 to edges of G and a weight of $f(n) \cdot n$ to nonedges, to obtain G'.

Proof continued

Proof continued

Proof continued

Proof continued

- If G has a HAM cycle, then the corresponding tour in G' has cost n, and $\mathcal{A}$ should return a solution of cost $\leq f(n) \cdot n$.

Proof continued

Proof continued

- If G has a HAM cycle, then the corresponding tour in G' has cost n, and $\mathcal{A}$ should return a solution of cost $\leq f(n) \cdot n$.
- If G has no HAM cycle, any tour in G' must use an edge of cost $f(n) \cdot n$, and therefore, $\mathcal{A}$ should return a solution of cost $> f(n) \cdot n$.

Proof continued

Proof continued

- If G has a HAM cycle, then the corresponding tour in G' has cost n, and $\mathcal{A}$ should return a solution of cost $\leq f(n) \cdot n$.
- If G has no HAM cycle, any tour in G' must use an edge of cost $f(n) \cdot n$, and therefore, $\mathcal{A}$ should return a solution of cost $> f(n) \cdot n$.
- Recall $\mathcal{A}$ runs in polynomial time. Thus, HAM cycle can be decided in polynomial time, and **P**=**NP**.

Approximation algorithm for metric TSP

Metric TSP

Given a complete graph G with nonnegative edge weights that satisfy triangle inequality, find a minimum cost cycle visiting every vertex exactly once.

Metric TSP is **NP**-complete but is approximable.

Approximation algorithm for metric TSP

Metric TSP

Given a complete graph G with nonnegative edge weights that satisfy triangle inequality, find a minimum cost cycle visiting every vertex exactly once.
Metric TSP is **NP**-complete but is approximable.

Approximation algorithm for metric TSP

Approximation algorithm for metric TSP

Metric TSP

Given a complete graph G with nonnegative edge weights that satisfy triangle inequality, find a minimum cost cycle visiting every vertex exactly once.
Metric TSP is **NP**-complete but is approximable.

Approximation algorithm for metric TSP

- Find an MST, T of G.

Approximation algorithm for metric TSP

Metric TSP

Given a complete graph G with nonnegative edge weights that satisfy triangle inequality, find a minimum cost cycle visiting every vertex exactly once.
Metric TSP is **NP**-complete but is approximable.

Approximation algorithm for metric TSP

- Find an MST, T of G.
- Double every edge of T to obtain an Eulerian graph $\mathcal{G}$. (An Eulerian graph is one where all vertices can be visited by traversing each edge exactly once.)

Approximation algorithm for metric TSP

Metric TSP

Given a complete graph G with nonnegative edge weights that satisfy triangle inequality, find a minimum cost cycle visiting every vertex exactly once.
Metric TSP is **NP**-complete but is approximable.

Approximation algorithm for metric TSP

- Find an MST, T of G.
- Double every edge of T to obtain an Eulerian graph $\mathcal{G}$. (An Eulerian graph is one where all vertices can be visited by traversing each edge exactly once.)
- Find an Eulerian tour $\mathcal{T}$ of $\mathcal{G}$.

Approximation algorithm for metric TSP

Metric TSP

Given a complete graph G with nonnegative edge weights that satisfy triangle inequality, find a minimum cost cycle visiting every vertex exactly once.
Metric TSP is **NP**-complete but is approximable.

Approximation algorithm for metric TSP

- Find an MST, T of G.
- Double every edge of T to obtain an Eulerian graph $\mathcal{G}$. (An Eulerian graph is one where all vertices can be visited by traversing each edge exactly once.)
- Find an Eulerian tour $\mathcal{T}$ of $\mathcal{G}$.
- Output the tour that visits vertices of G in order of their first appearance (short cutting)in $\mathcal{T}$. Let $\mathcal{C}$ be this tour.

Approximation algorithm for metric TSP

Approximation ratio for metric TSP is 2

Approximation algorithm for metric TSP

Approximation ratio for metric TSP is 2

- Use lower bound idea again. $\text{cost}(T) \leq \text{OPT}$ as deleting any edge from an optimal solution to TSP gives us a spanning tree.

Approximation algorithm for metric TSP

Approximation ratio for metric TSP is 2

- Use lower bound idea again. $\text{cost}(T) \leq \text{OPT}$ as deleting any edge from an optimal solution to TSP gives us a spanning tree.
- $\mathcal{T}$ contains each edge of T twice, so $\text{cost}(\mathcal{T}) = 2 \cdot \text{cost}(T)$.

Approximation algorithm for metric TSP

Approximation ratio for metric TSP is 2

- Use lower bound idea again. $\text{cost}(T) \leq \text{OPT}$ as deleting any edge from an optimal solution to TSP gives us a spanning tree.
- $\mathcal{T}$ contains each edge of T twice, so $\text{cost}(\mathcal{T}) = 2 \cdot \text{cost}(T)$.
- Because of triangle inequality, after the short cutting step, $\text{cost}(\mathcal{C}) \leq \text{cost}(\mathcal{T})$.

Approximation algorithm for metric TSP

Approximation ratio for metric TSP is 2

- Use lower bound idea again. $\text{cost}(T) \leq \text{OPT}$ as deleting any edge from an optimal solution to TSP gives us a spanning tree.
- $\mathcal{T}$ contains each edge of T twice, so $\text{cost}(\mathcal{T}) = 2 \cdot \text{cost}(T)$.
- Because of triangle inequality, after the short cutting step, $\text{cost}(\mathcal{C}) \leq \text{cost}(\mathcal{T})$.
- Combining the above, we get $\text{cost}(\mathcal{C}) \leq 2 \cdot \text{OPT}$.

At Last!!!

Thank you