

CPSC 311 Lecture Notes

NP-Completeness and Approximation Algorithms

(Chapters 34 and 35)

Acknowledgement: Parts of these course notes are based on notes from courses given by Jennifer Welch at Texas A&M University.

NP-Completeness (Chapt 34)

The class P: class of problems that can be **solved** in time that is **polynomial in the size of the input**.

- i.e., if input size is n , then worst-case running time is $O(n^c)$ for constant c
- problems in P considered ‘tractable’ (if not in P, then not tractable)
- all problems we studied so far are in P
- (e.g., sort a list of numbers)

The class NP: class of problems whose solutions can be **verified** in time that is polynomial in the size of the input.

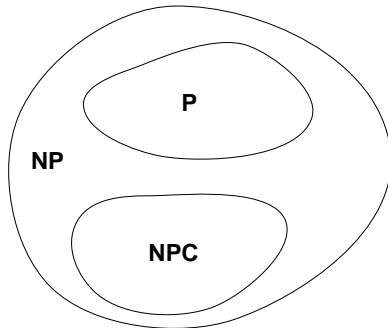
- ... maybe its easier to check than to compute?
- (e.g., check that a list is sorted, rather than sorting it)

The class NP-Complete (NPC): class of the ‘hardest’ problems in NP

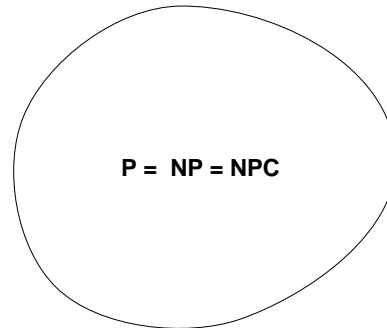
- have property that if any NPC problem can be solved in polynomial time (ptime) then all problems in NP could be solved in ptime.
- (we will use transformations, or reductions, to grow our set of NPC complete problems from one initial problem, SAT)

P, NP, and NPC... and why do we care?

If $P \neq NP$, then world looks like:



If $P = NP$, then world looks like:



Is $P \subseteq NP$? – Yes!

Is $NPC \subseteq NP$? – Yes, by definition.

Is $NP \subseteq P$? – ???

- don't know for sure... but intuition is no!
- probably the most famous 'open problem' in CS
- seems plausible that ability to 'guess and verify' any solution in ptime is more powerful than deterministic ptime (computing solution from scratch)
- so... we *think* $P \neq NP$, but no one has proved it one way or the other (despite many smart people working for many years)

So... why do we care to know whether a problem is NP-Complete?

- if it is, then you are highly unlikely to find a ptime algorithm to solve it, so don't bother!
- instead, you should spend your time looking for:
 - efficient **approximation algorithm** finding solution close to optimal
 - **heuristics** that give correct answer in many, but not all, cases

NP-Completeness and Decision Problems

Decision Problems: we will develop a theory for **decision problems** (problems with yes/no answers).

Example: Shortest Paths

- general problem: What is the length of the shortest x to y path?
- decision problem: Is there an x to y path of length $\leq k$?

Rationale for studying decision problems:

- solving general problem is at least as hard as solving decision problem
- for many natural problems, only need polynomial additional time to solve the general problem
- so we will study decision problems (which are easier to study)

Example: Traveling Salesman Problem (TSP)

The Traveling Salesman Problem (TSP):

- **instance:** a set of cities and the distances between each pair of cities (given as a graph), and a bound B
- **question:** is there a ‘tour’ that visits every city exactly once, returns to the start, and has total distance $\leq B$?

Is $TSP \in NP$?

Need to verify that we can *check* a solution (tour/list of cities) in ptime (i.e., time $O(n)$ where n is the number of cities) ...

- check that list tour visits every city exactly once
- check that the tour returns to the start
- check that total distance $\leq B$

All can be done in $O(n)$ time, so $TSP \in NP$

Use algorithm for decision problem D_{TSP} to solve general problem

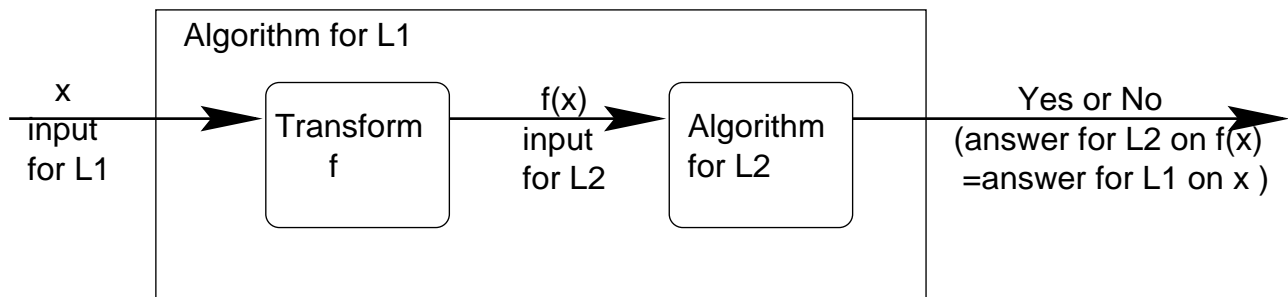
1. set $B = c$, some small constant
2. call D_{TSP} with bound B
3. **if** (D_{TSP} returns YES)
4. **then** decrease B and call D_{TSP} again
5. **else** increase B and call D_{TSP} again
6. **repeat** until converge on actual minimum value

Transformations (Reductions) between Decision Problems

Let L_1 and L_2 be two decision problems. Suppose we have an algorithm A_2 to solve L_2 . Can we use A_2 to solve L_1 as well (still in polynomial time)?

Polynomial-time transformation f from L_1 to L_2 ($L_1 \leq_p L_2$):

- f transforms input for L_1 into an input for L_2 such the transformed input is a yes-input for L_2 if and only if the original input was a yes-input for L_1
- f is computable in ptime (in the size of the input)



- if such an f exists, we say $L_1 \leq_p L_2$

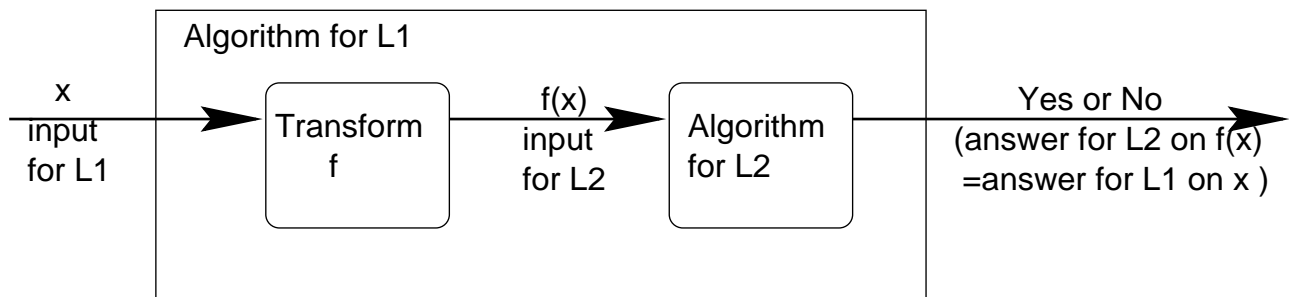
Theorem: If $L_1 \leq_p L_2$ and $L_2 \in P$, then $L_1 \in P$ (L_2 is at least as hard as L_1)

Proof: So... we have a ptime algorithm for L_2 (since $L_2 \in P$) and a ptime transformation f mapping input x for L_1 to an input for L_2 . Here's a ptime algorithm for solving L_1 :

1. take input x for L_1 and compute $f(x)$
2. run ptime alg for L_2 on $f(x)$, and return (same) answer found for L_2

□

Transformations (Reductions) between Decision Problems



Significance of showing $L_1 \leq_p L_2$

- if \exists ptime algorithm for L_2 , then \exists ptime algorithm for L_1
- if \nexists ptime algorithm for L_1 , then \nexists ptime algorithm for L_2

Fact: \leq_p is *transitive*, i.e., $L_1 \leq_p L_2$ and $L_2 \leq_p L_3$ implies that $L_1 \leq_p L_3$.

Ptime Transformation Example: Hamiltonian Circuit and TSP

The Hamiltonian Circuit Problem (HC):

instance: an undirected graph $G = (V, E)$

question: is there a path in G that includes every node exactly once and returns to the start

The Traveling Salesman Problem (TSP):

instance: a set of cities, distances between each city-pair (graph), bound B

question: is there a ‘tour’ that visits every city exactly once, returns to the start, and has total distance $\leq B$?

Claim: $HC \leq_p TSP$

Proof: To prove this we need to do two things:

1. Define the transformation f mapping inputs to HC into inputs for TSP, and show it can be computed in ptime.
 - f must map the input $G = (V, E)$ for HC into a list of cities, distances, and a bound B for input to TSP
 - f must be computable in ptime in size of HC input graph
2. Show that the transformation is correct, that is, show that f transforms yes-inputs for HC into yes-inputs for TSP, and no-inputs for HC into no-inputs for TSP. (G has a HC iff tranformed input has a TSP tour with length $\leq B$)

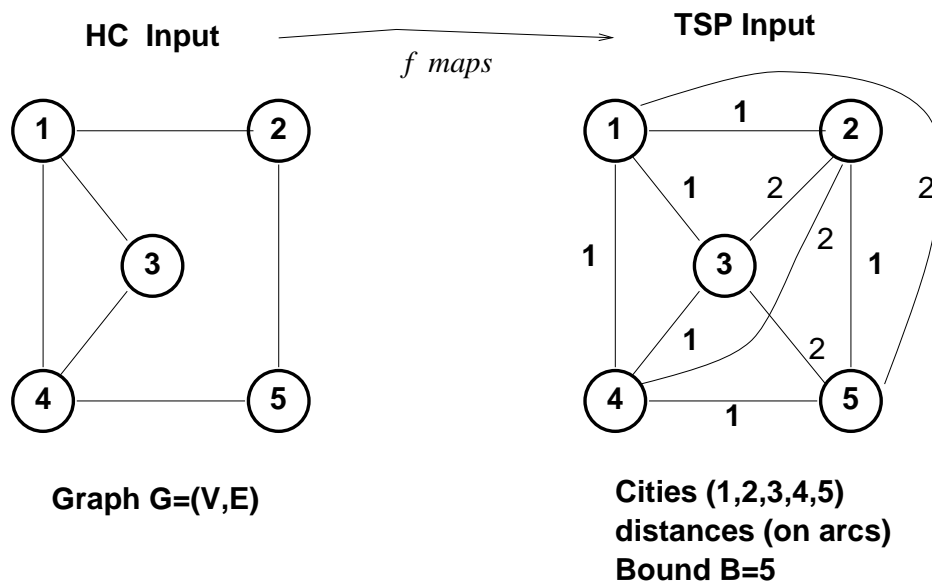
Ptime Transformation Example: Hamiltonian Circuit and TSP

- **HC instance:** an undirected graph $G = (V, E)$
- **TSP instance:** a set of cities and the distances between each pair of cities (given as a graph), and a bound B

1. Definition of transformation f for $HC \leq_p TSP$

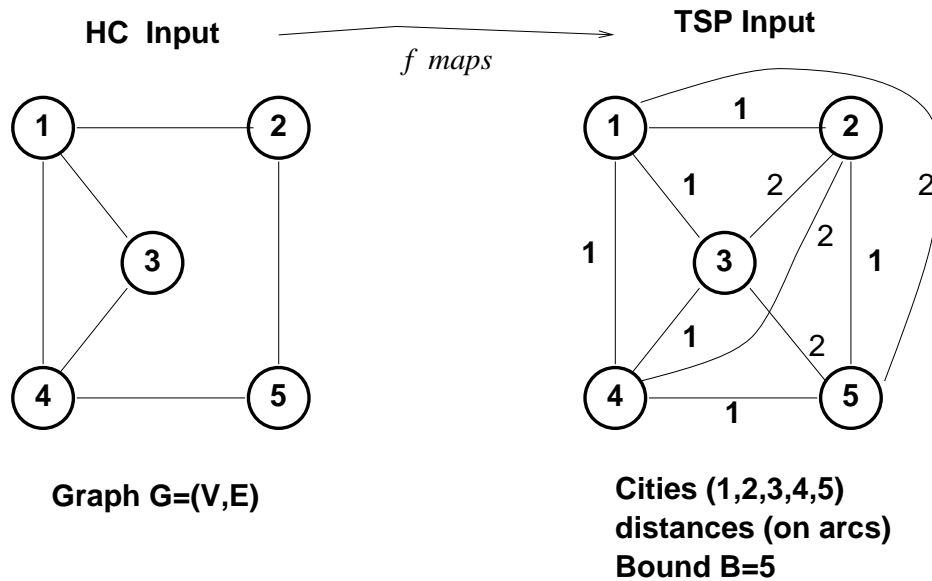
Given the HC input graph $G = (V, E)$ with n nodes:

- create a set of n cities labeled with names in V
- set intercity distances $d(u, v)$ to be $d(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ 2 & \text{if } (u, v) \notin E \end{cases}$
- set bound $B = n$



Note: f is computed in $O(n^2)$ time ($O(V^2)$ from graph notation)

Ptime Transformation Example: Hamiltonian Circuit and TSP



2. Prove the transformation f for $HC \leq_p TSP$ is correct

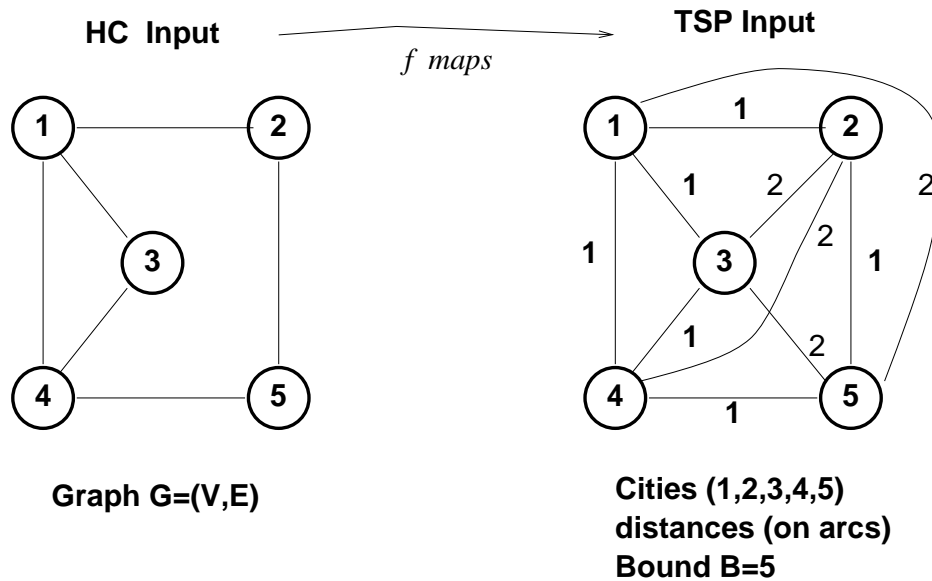
We will prove this by showing that $x \in HC \iff f(x) \in TSP$:

- **2(a)** if $x \in HC$, then $f(x) \in TSP$
- **2(b)** if $f(x) \in TSP$, then $x \in HC$

2(a) if $x \in HC$, then $f(x) \in TSP$

- $x \in HC$ means HC input $G = (V, E)$ has a hamiltonian circuit
 - wlog suppose it is $(v_1, v_2, \dots, v_n, v_1)$
- then $(v_1, v_2, \dots, v_n, v_1)$ is also a tour of the cities in $f(x)$, the transformed TSP instance
- the distance of the tour $(v_1, v_2, \dots, v_n, v_1)$ is $n = B$ since each consecutive pair $(v_i, v_{(i+1) \bmod n})$ is connected by an edge in E , and so has distance 1 in $f(x)$
- thus, $f(x) \in TSP$, as desired

Ptime Transformation Example: Hamiltonian Circuit and TSP



2. Prove the transformation f for $HC \leq_p TSP$ is correct

We will prove this by showing that $x \in HC \iff f(x) \in TSP$:

- **2(a)** if $x \in HC$, then $f(x) \in TSP$
- **2(b)** if $f(x) \in TSP$, then $x \in HC$

2(b) if $f(x) \in TSP$, then $x \in HC$

- $f(x) \in TSP$ means there exists a tour in TSP input that has total distance $\leq n = B$. Wlog suppose it is $(v_1, v_2, \dots, v_n, v_1)$
- since all intercity distances are either 1 or 2 in $f(x)$, and there are n intercity 'legs' in the tour, each 'leg' (edge) must have distance 1
- so G must have an edge between each consecutive pair of cities on the tour, and so $(v_1, v_2, \dots, v_n, v_1)$ must be a hamiltonian circuit in G
- thus, $x \in HC$, as desired

□

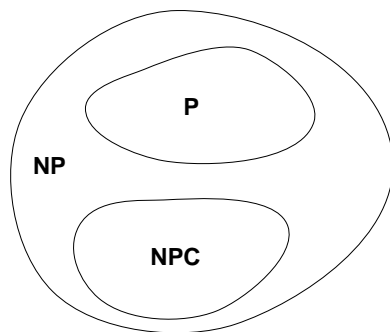
NP-Completeness... and our use for ptime transformations...

Definition: A decision problem L is **NP-Complete** (NPC) if:

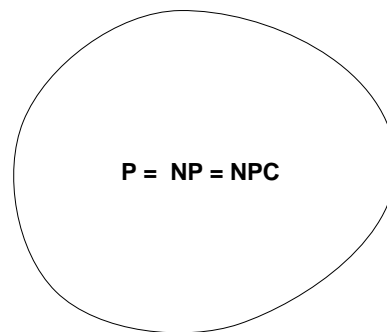
1. $L \in NP$, and
2. for every $L' \in NP$, $L' \leq_p L$ (i.e., every L' in NP can be transformed to L – so L is at least as hard as every problem in NP)

Note: If L only satisfies condition 2, we say it is **NP-Hard**

If $P \neq NP$, then world looks like:



If $P = NP$, then world looks like:



Theorem: Suppose L is NPC:

- if \exists ptime algorithm for L , then \exists ptime algorithm for every $L' \in NP$, i.e., $P = NP$
- if \nexists ptime algorithm for L , then \nexists ptime algorithm for any $L' \in NPC$, i.e., $P \neq NP$

How to show a problem is NPC

Showing a problem L is NPC by reduction

1. show $L \in NP$ (NPC def #1)
2. Select a known NPC problem L' and show $L' \leq_p L$ (shows NPC def #1)

Why does this work? $\forall L'' \in NP$

- since $L' \in NPC$, $L'' \leq_p L'$ for all $L'' \in NP$ (by def of NPC)
- if we show that $L' \leq_p L$, then transitivity of \leq_p gives

$$L'' \leq_p L' \leq_p L$$

or every $L'' \in NP$ is ptime transformable to L (NPC def #2)

But how do we start? need to prove one problem is NPC from scratch...

The Satisfiability Problem (SAT):

- **instance:** boolean expression in CNF, e.g.,

$$(x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee x_4 \vee \overline{x_5})$$
- **question:** does there exist an assignment of truth values to the variables that makes the expression true?

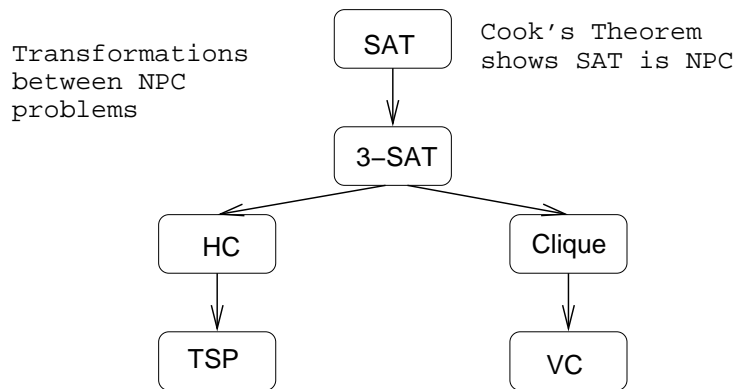
Cook's Theorem: *Satisfiability (SAT) is NPC*

Proof Idea: (shown in CPSC 433)

1. show $SAT \in NP$ (easy, just check if the truth assignment works)
2. show $\forall L' \in NP$, $L' \leq_p SAT$ (i.e., every $L' \in NP$ can be reduced to SAT)

More NPC problems...

After Cook showed SAT was NPC, Dick Karp used reduction (sometimes called Karp-reduction) to show many other problems are NPC (around 20). Since then, hundreds of problems added to NPC set – and growing all the time.



The 3-Satisfiability Problem (3-SAT):

instance: CNF boolean expression where each clause has exactly 3 literals

question: does there exist a satisfying truth assignment?

The Vertex Cover Problem (VC):

instance: a graph $G = (V, E)$ and a positive integer $k \leq V$

question: is there a subset $V' \subseteq V$ of size $\leq k$ such that each edge in E has at least one endpoint in V' ?

The Clique Problem (Clique):

instance: a graph $G = (V, E)$ and a positive integer $k \leq V$

question: does G have a clique of size $\geq k$ (a subset $V' \subseteq V$ of size $\geq k$ such that V' is a complete graph)?

The Independent Set Problem (IS):

instance: a graph $G = (V, E)$ and a positive integer $k \leq V$

question: does G have an independent set of size $\geq k$ (a subset $V' \subseteq V$ of size $\geq k$ such that G has no edge between any pair)?

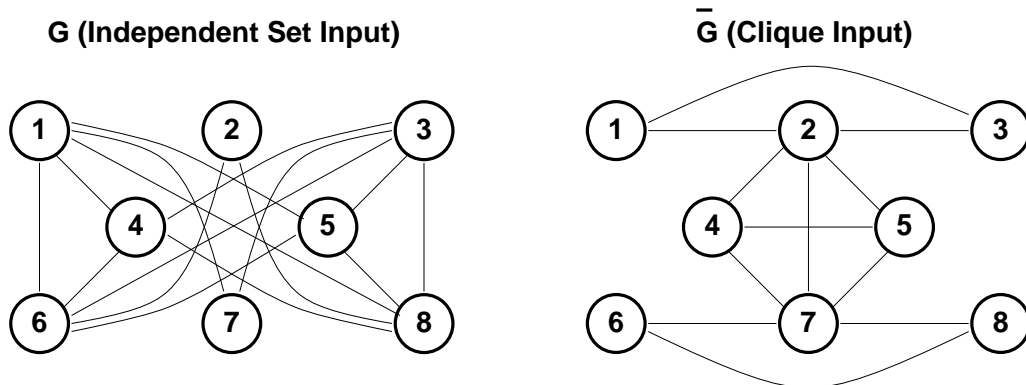
Cliques (and Independent Sets)

Theorem: $Clique \in NPC$

Proof: (a different proof from text)

1. Show $Clique \in NP$

- given a set V' of k nodes and $G = (V, E)$
- for every $x, y \in V'$, check that $(x, y) \in E$
- takes time polynomial in $O(V + E)$, and precisely $O(V^3)$ time in worst-case – $O(k^2) = O(V^2)$ edges to check for, each check takes $O(V)$ time



2. We'll use IS as known NPC problem, and will show $IS \leq_p Clique$

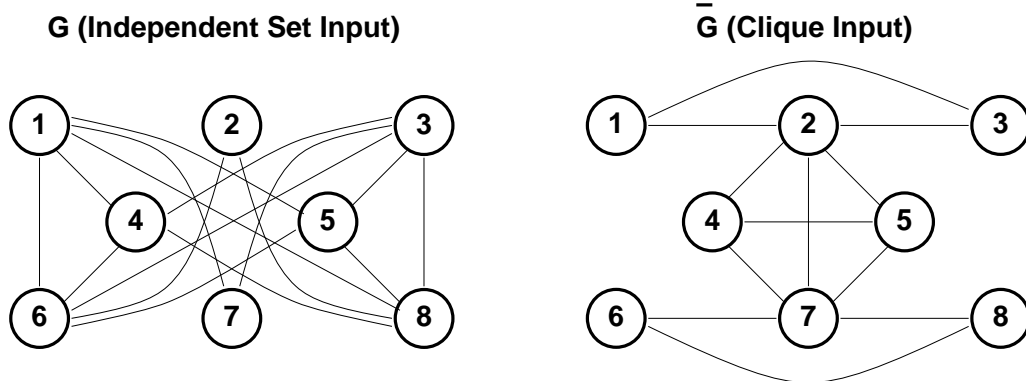
Our p time transformation f :

- given $G = (V, E)$ construct complement graph $\bar{G} = (V, \bar{E})$
- keep k the same
- takes $O(V^2)$ time to construct \bar{G} 's adjacency matrix

Now we have to show the transformation is correct... i.e.,

- **(a)** if $x \in IS$, then $f(x) \in Clique$
- **(b)** if $f(x) \in Clique$, then $x \in IS$

Cliques (and Independent Sets)



2(a) if $x \in IS$, then $f(x) \in Clique$

- suppose $G = (V, E)$ has an IS $V' \subset V$ of size at least k
- in $\bar{G} = (V, \bar{E})$, V' will be Clique of size of at least k
- i.e., G has no edge between any pair of vertices in V' , so \bar{G} must have all these edges – forming a clique

2(b) if $f(x) \in Clique$, then $x \in IS$

- suppose $\bar{G} = (V, \bar{E})$ has a Clique $V' \subset V$ of size at least k
- in $G = (V, E)$, V' will be an IS of size of at least k
- i.e., \bar{G} has an edge between every pair of vertices in V' , and G must have none of these edges – forming an IS

Therefore, Clique is NPC (assuming IS is already known to be NPC)

□

Approximation Algorithms

Suppose you just discovered a problem you'd like to solve is NPC... what should you do???

- hope/show the bad running time doesn't happen for the inputs you will use
- find heuristics to quickly find correct solutions in many cases
- find an approximation algorithm that (is guaranteed) to find a solution that is "close to" optimal

Approximation Algorithm for TSP Problem

- **instance:** finite set of cities and distances
- **feasible solutions:** all permutations of cities (all possible tours)
- **value of a solution:** length of the tour for that solution
- **approximation algorithm:** returns some feasible solution

Measuring the performance of an Approximation Algorithm A

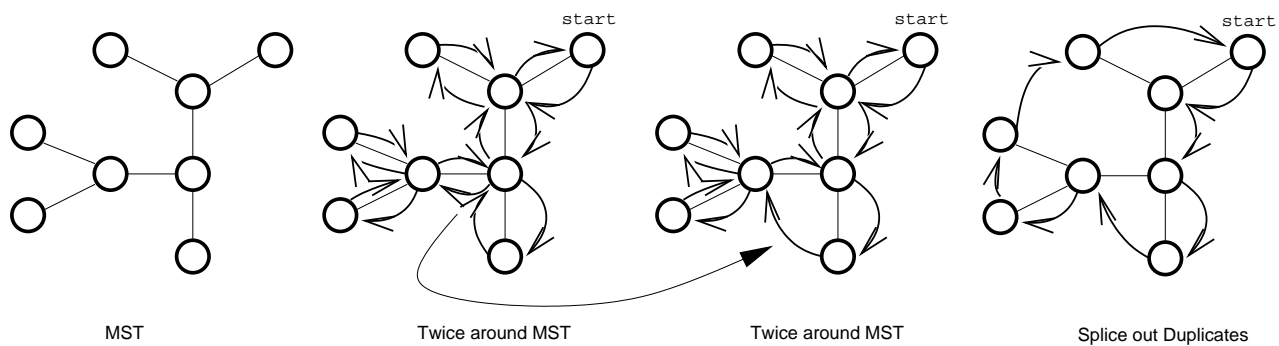
- $v_l(A(I))$ – value of soln returned by Approx Alg A on instance I
- $OPT(I)$ – value of optimal solution for instance I
- for minimization probs: (reverse for max) $r_A(I) = \frac{v_l(A(I))}{OPT(I)}$
- $R_A = l.u.b.\{r \mid r_A(I) \leq r \quad \forall I\}$
– i.e., smallest r s.t. for all inputs ratio is at most r

Ex: Approximation Algorithm for special case of TSP

Assumption: intercity distances satisfy triangle inequality, i.e., shortest distance between 2 cities is direct route so $d(c_1, c_2) \leq d(c_1, c_3) + d(c_3, c_2)$

Approximation Algorithm A

1. convert TSP input (cities, distances) to complete weighted graph $G = (V, E)$ with $V =$ cities, edges weighted with distances (or ∞ if none)
2. compute MST of G
3. go “twice around” MST to get a tour
4. “adjust” tour (splice out cities/vertices) so don't visit same city twice



Running Time: polynomial – $O(V \log V)$

Quality of the Solutions Found: $A(I)$ is length of tour found

- $A(I) \leq 2 \cdot$ length of MST (sum of edge weights)
 - triangle inequality says wt of new edges \leq than edges they replace
- $A(I) < 2 \cdot OPT(I)$ because $MST(I) \leq OPT(I)$
- $R_A < 2$ since $v_l(A(I))/OPT(I) < 2$, i.e., solution at most twice optimal

The second to last step can be shown by contradiction. If MST length \geq min tour, then deleting any edge of tour give spanning tree with weight $<$ MST. A contradiction.

TSP without Triangle Inequality

Theorem: *If $P \neq NP$, then no polytime approximation algorithm A for TSP can have $R_A < \infty$. That is, no A can approximate the tour within a constant factor of optimal.*

Proof: Suppose in contradiction there is such an algorithm A with $R_A \leq k$. We will use A to solve HC in ptime (since HC is NPC and we assumed $P \neq NP$, this is a contradiction).

Algorithm for HC

1. convert G to an instance I of TSP, with cities = V , and

$$d(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ kV & \text{if } (u, v) \notin E \end{cases} \quad (\text{doesn't satisfy triangle inequality})$$
2. Run A on I
3. if $v_l(A(I)) \leq kV$, then “Yes” G has HC, else “No”.

Why Correct?

if G has HC: then an optimal tour in I is that cycle and it has weight V (all edge weights 1), so $OPT(I) = V$, and $v_l(A(I)) \leq kOPT(I) = kV$

if G has no HC: then an optimal tour in I must use some edge not in G , which has weight kV , so $OPT(I) > kV$, and $v_l(A(I)) \geq OPT(I) > kV$