Algorithms and Computation

(grad course)

Lecture 8: Approximation algorithms for NP-Hard problems

Instructor: Hossein Jowhari

jowhari@kntu.ac.ir

Department of Computer Science and Statistics
Faculty of Mathematics
K. N. Toosi University of Technology

Fall 2021

NP-Hard Problems

- Decision problems are sets of strings

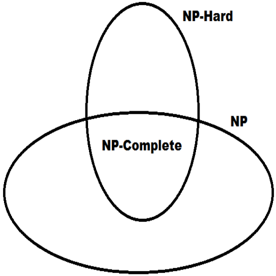
$$\text{HamCycle } = \{G \mid G \text{ is a graph with a Hamiltonian cyle.}\}$$

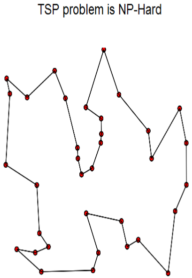
- We extend the definition of a *problem* to include non-decision problems as well.
- In this lecture we consider discrete optimization problems.
- Known as NP-Hard problems, these problems are at least as hard as NP-complete problems.
- For example the problem of $\mathrm{LONGEST} - s - t - \mathrm{PATH}$ asks for the length of the longest (simple) path from s to t in a given graph.

Definition: Problem X is NP-Hard if any problem in NP is polynomial-time reducible to X

Note: By definition, an NP-Complete problem is NP-Hard. However the converse is not necessarily true. An NP-Hard problem may not be in NP.

For example, it is not clear if the TSP problem is in NP. The non-decision version of TSP asks for a tour of the cities with the shortest length in the given input.





Examples of NP-Hard problems

- **MAX-SAT**: Find an assignment to the variables of a Boolean formula ϕ (in CNF format) that satisfies the maximum number of clauses.

$$\phi = (x_1 \vee \overline{x_2} \vee x_3 \vee x_4) \wedge \ldots \wedge (\ldots)$$

- **MAX-2SAT**: Find an assignment to the variables of a Boolean formula ϕ (in $2 - CNF$ format) that satisfies the maximum number of clauses.

$$\phi = (x_1 \vee \overline{x_2}) \wedge (x_3 \vee \overline{x_1}) \wedge \ldots \wedge (\ldots)$$

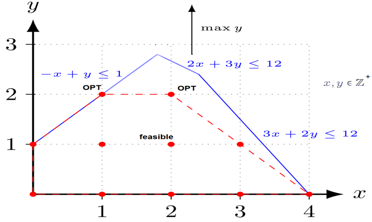
- **MAX-IND-SET**: Given an undirected graph G find a independent set in G with maximum cardinality.

- **MIN-VERTEX-COVER**: Given an undirected graph G find a vertex cover in G with minimum cardinality.
- **TSP**: Given a set of cities with pairwise distances between them find a tour of the cities with minimum length.
- **K-CENTER**: Given a set of n points A with pairwise distances between them choose a set of k points in A as centers such that the maximum distance to the nearest center is minimized.
- ...

Question: Are the above problems in NP?

Feasible solutions

- An optimization problem has a set of *feasible* solutions. The problem asks for a feasible solution that maximizes/minimizes a certain objective function (optimal solution)



- An optimization problem may have several optimal solutions.

- For example in the LONGEST-S-T-PATH problem, the set of all paths from the vertex S to the vertex T is the feasible set.

Approximation algorithms for NP-Hard problems

- There is very little hope in obtaining exact polynomial time algorithms for NP-hard problems
- A great body of research has been devoted to the design of efficient algorithms that find **near-optimal solutions (approximate solutions)** for these problems.
- Let
$$\mathrm{cost} : \mathrm{Feasible(X)} \to \mathbb{R}^+$$
be a cost function defined over the **feasible solutions** of the NP-Hard problem X. Suppose X is a minimization problem where the objective is to find a solution x with minimum $\mathrm{cost}(x)$.

- **Definition**: Let $f \geq 1$. A polynomial time f-factor approximation algorithm for the minimization problem X, finds a feasible solution y for X in polynomial time where

$$\mathrm{cost}(opt) \leq \mathrm{cost}(y) \leq f\,\mathrm{cost}(opt)$$

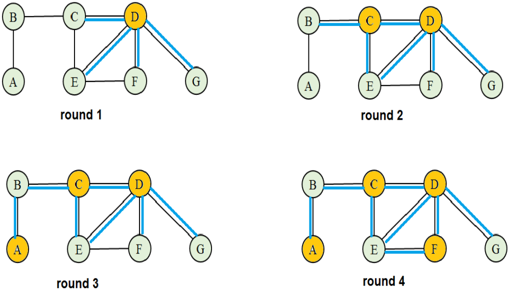
Here opt is an optimal solution of the problem X.

Let $0 < f' \leq 1$. In case X is a maximization problem, a polynomial time f'-factor approximation algorithm for the problem X finds a feasible solution y for X in polynomial time where

$$f'\,\mathrm{value}(opt) \leq \mathrm{value}(y) \leq \mathrm{value}(opt)$$

First approximation algorithm

Greedy algorithm for MIN-VERTEX-COVER: The algorithm works in rounds. In each round, the algorithm picks a vertex that covers the maximum number of uncovered edges. The algorithm stops when no edge is left to be covered.



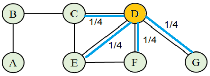
There is a better solution with $\{D, E, B\}$ as the vertex cover.

Analysis of the greedy algorithm

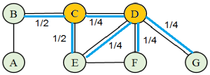
Notation: Let $p(e)$ be the price we pay for covering the edge e.

Note that when we pick a new vertex u, our cost is increased by 1. If u has k uncovered edges, then the price $p(e)$ of each uncovered edge on u would be $\frac{1}{k}$.

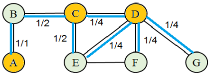
$$\text{The cost of the solution } = \sum_{e \in E} price(e)$$



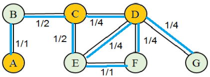
round 1



round 2



round 3



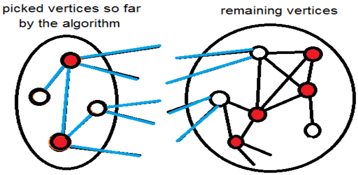
round 4

We number the edges $e_1, e_2, \ldots, e_m$ according to the order they were covered by the algorithm (ties broken arbitrarily.)

Lemma: For each $k \in \{1, \ldots, m\}$ we have $price(e_k) \le \frac{OPT}{m-k+1}$. Here OPT is the cost of the optimal solution.

Proof: At each point in the algorithm the unpicked vertices of the optimal solution can cover all the uncovered edges (why?) Let OPT' be the number of unpicked optimal vertices.

Therefore when we cover e_k, there must be an edge with price at most $\frac{OPT'}{m-k+1}$ (why? averaging argument.) Note that the algorithm looks for edges with small prices.



$$\min\{\frac{1}{x}, \frac{1}{y}, \frac{1}{z}\} \le \frac{3}{x+y+z}$$

Therefore we must have $price(e_k) \le \frac{OPT'}{m-k+1} \le \frac{OPT}{m-k+1}$. $\square$

Theorem: Greedy vertex cover algorithm is a $(\log m + 1)$-factor approximation algorithm.

$$\text{The cost of the solution } = \sum_{e \in E} price(e)$$

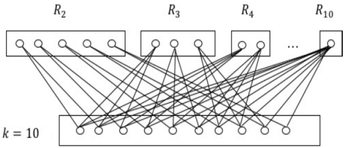
$$\le \sum_{e \in E} \frac{OPT}{m-k+1} \le OPT\underbrace{\left(\frac{1}{1} + \frac{1}{2} + \ldots + \frac{1}{m}\right)}_{H_m}$$

$$\le OPT(\ln m + 1) \le OPT(\log m + 1)$$

Near tight example

Consider a bipartite graph $G = (A \cup B, E)$ where

- $|A| = k$,
- $B = R_2 \cup \ldots \cup R_k$
- Vertices in R_i have degree i
- $|R_i| = \lfloor \frac{k}{i} \rfloor$



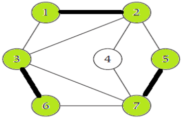
The greedy algorithm picks the vertices in R_k then R_{k-1}, then R_{k-2} and so on.

Greedy cost $= \sum_{i=2}^{k} |R_i| = \sum_{i=2}^{k} \lfloor \frac{k}{i} \rfloor = \Omega(k \log k)$

Optimal solution picks L. Optimal cost $= k$

A 2-approximation algorithm for MIN-VERTEX-COVER

- Let M be a **maximal matching** in G. Note that we can find M in polynomial time.
- Let C be the set containing both endpoints of the edges in M.
- C is a vertex cover for G. (If edge is not covered them M cannot be a maximal.)
- $|C| \leq 2OPT$. (In every graph we have $OPT \geq |M|$)



An integer program for weighted MIN-VERTEX-COVER

- For each vertex $u \in V$, we have a variable $x_u \in \{0,1\}$.
- $x_u = 1$ if and only if u has been chosen
- For each edge (u,v) we have a constraint $x_u + x_v \geq 1$.

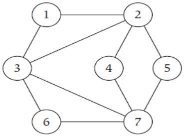
$$\min \sum_{i=1}^{n} c_i x_i$$

$$x_1 + x_2 \geq 1, \quad x_1 + x_3 \geq 1, \quad x_2 + x_3 \geq 1$$

$$x_2 + x_5 \geq 1, \quad x_3 + x_6 \geq 1, \quad x_4 + x_7 \geq 1$$

$$x_3 + x_7 \geq 1 \quad x_2 + x_7 \geq 1 \quad x_6 + x_7 \geq 1$$

$$x_2 + x_4 \geq 1, \qquad x_1, \ldots, x_6 \in \{0,1\}$$



Note that $\min \sum_{i=1}^{n} c_i x_i = OPT$

A linear programming relaxation

We relax the integer constraints on the variables $\{x_u\}_{u \in V}$.

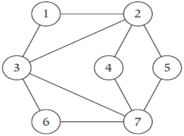
$$\min \sum_{i=1}^{n} c_i x_i$$

$$x_1 + x_2 \geq 1, \quad x_1 + x_3 \geq 1, \quad x_2 + x_3 \geq 1$$

$$x_2 + x_5 \geq 1, \quad x_3 + x_6 \geq 1, \quad x_4 + x_7 \geq 1$$

$$x_3 + x_7 \geq 1 \quad x_2 + x_7 \geq 1 \quad x_6 + x_7 \geq 1$$

$$x_2 + x_4 \geq 1, \qquad x_1, \ldots, x_6 \in [0, 1]$$

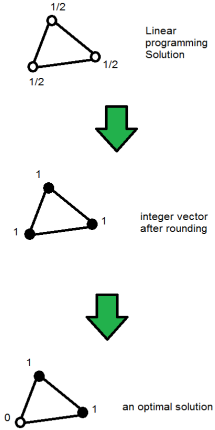


Note that $\min \sum_{i=1}^{n} c_i x_i \leq OPT$

Let P be a linear program with n variables and m constraints. Let W be a largest number in the program. We can solve the linear program P in time $\mathrm{poly}(n, m, \log W)$.

Another 2-factor approximation algorithm for MIN-VERTEX-COVER

- Construct the linear program for the vertex cover instance as described earlier.
- Solve the linear program.
- Let $\{x_i^*\}_{i\in\{1,\ldots,n\}}$ be the solution.
- Note that $\sum_{i=1}^{n} c_i x_i^* \leq OPT$
- (**Rounding**) If $x_i^* \geq \frac{1}{2}$ we set $z_i = 1$ otherwise $z_i = 0$
- Note that $\{z_i\}$ is an integer vector and describes a vertex cover of G



Since $z_i \leq 2x_i^*$ and $\sum_{i=1}^{n} c_i x_i^* \leq OPT$, we have

$$\sum_{i=1}^{n} c_i z_i \leq 2OPT$$

As result the LP rounding algorithm is a 2-factor approximation algorithm that runs in polynomial time.

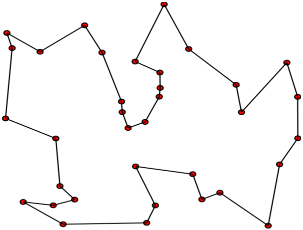
Exercise

Consider the **MAX-3SAT** problem. Suggest a trivial $\frac{1}{2}$-factor approximation algorithm for this problem.

$$\phi = (x_1 \vee \overline{x_2} \vee x_3) \wedge \ldots \wedge (\ldots)$$

There is a *randomized* $\frac{7}{8}$-factor approximation algorithm for this problem. (We return to this problem in the future lectures!)

Approximation algorithm for TSP



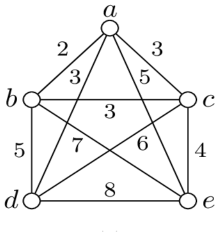
Given a complete graph with non-negative edge weights find a minimum cost cycle that visits every vertex exactly once.

There is no polynomial time approximation algorithm for TSP! (why?)

A 2-factor approximation algorithm for metric TSP

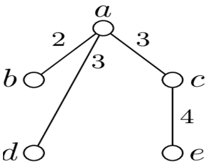
In the metric TSP problem the distance function between the vertices describes a metric.

- $d(i,j) \geq 0$
- $d(i,j) = d(j,i)$
- $d(i,j) \leq d(i,k) + d(k,j)$



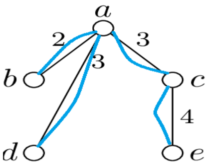
Algorithm:

- Let T be an MST (Minimum Spanning Tree) of the complete graph G.



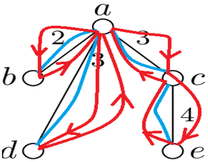
$$\text{weight}(T) \leq OPT$$

- Double the edges of M resulting in a graph G'



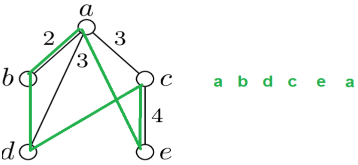
$$\text{weight}(G') \leq 2 \times OPT$$

- Find an Eulerian tour of G'. Let R be Eulerian tour.



$$\text{weight}(R) = \text{weight}(G') \leq 2 \times OPT$$

- Output the vertices in the order of first appearance in the Eulerian tour.



$$\text{weight}(solution) \leq \text{weight}(R) \leq 2 \times OPT$$

The left inequality follows from the metric property.