KAIST, School of Computing, Spring 2024 **Scribed By: Eunjung KIM**
Algorithms for NP-hard problems (CS492) **Lecture #21**
Lecture: Eunjung KIM **7 May 2024**

Contents

1 Basics of Approximation Algorithm

From now on, as long as approximation algorithms are concerned, we consider *Optimization* problems instead of decision problems. Regarding the formal definition of optimization problems, see the first lecture note.

The optimization version of VERTEX COVER called MINIMUM VERTEX COVER, takes a graph G as an input, and our goal is to find a vertex cover of maximum size. We cannot say that MINIMUM VERTEX COVER is NP-complete because NP-completeness is a term for decision problems only. However, we can consider the canonical *decision* version of an optimization problem. If the decision version is NP-hard, by convention we say the optimization problem NP-hard. Clearly, the decision variant can be solved by solving the optimization problem, so unless the decision variant is in P, we cannot expect to solve the optimization problem in polynomial time.

In the next couple of week, we learn how to design an algorithm which runs in polynomial time and which produces a solution *as close to an optimal solution as possible*. Such an algorithm is called an *polynomial-time approximation algorithm* or simply *approximation*. Unlike in FPT algorithms or fast exact algorithms, our running time budget is polynomial function and our main concern is to design an algorithm which produces a solution as close to an optimal solution as possible. Before formally defining this performance measure, let us consider an approximation algorithm for MINIMUM VERTEX COVER.

Algorithm 1 Algorithm for MINIMUM VERTEX COVER

```
1: procedure minVC($G$)
2:     $S \leftarrow \emptyset$.
3:     Mark every edge of $G$ as uncovered
4:     while $G$ has an uncovered edge do
5:         Pick an edge $uv$.
6:         $S \leftarrow S \cup \{u, v\}$
7:         Mark every edge incident with $u$ or $v$ covered
8:     return $S$.
```

Note that no two edges chosen in line 5 share a vertex. Indeed, once an edge $e = uv$ is chosen in line 5 all edges sharing a vertex with e will be marked as `covered` and will never be chosen later during the algorithm. That is, the edge set M chosen during the course of the algorithm 1 is a matching of G. Let `opt` be the size of an optimal vertex cover. Because any vertex cover of G must cover M as well, we have $\texttt{opt} \geq |M|$. Now,

$$|S| = 2 \cdot |M| \leq 2 \cdot \texttt{opt},$$

that is, the returned solution of Algorithm 1 does not exceed twice the size of an optimal solution.

In general, for a minimization problem Π, a (polynomial-time) algorithm is said to be *α-approximation* if it always (i.e. for any input instance to Π) outputs a solution whose objective value is at most $\alpha \cdot \texttt{opt}$, where `opt` is the objective value of an optimal solution. For a maximization problem, an *α-approximation* outputs a solution whose objective value is *at least* $\frac{1}{\alpha} \cdot \texttt{opt}$, where `opt` is the objective value of an optimal solution. Clearly, we have $\alpha \geq 1$ and α is called the *approximation ratio* of the said algorithm. Sometimes α can be a function of the input size, e.g. the number of vertices.

2 Approximation for TRAVELING SALESMAN PROBLEM

We formulate TRAVELING SALESMAN PROBLEM as a graph problem.

TRAVELING SALESMAN PROBLEM
Instance: an undirected graph G and a distance function $d : E(G) \to \mathbb{Q}$.
Goal: Find a cycle of minimum total distance visiting every vertex exactly once.

TRAVELING SALESMAN PROBLEM does not admit an ρ-approximation for any constant ρ.

Lemma 1. TRAVELING SALESMAN PROBLEM *does not admit an α-approximation for any constant α unless $P = NP$.*

Proof: We reduce from HAMILTONIAN CYCLE. Given an input graph $G = (V, E)$ of HAMILTONIAN CYCLE, we create an instance (G', d) of TRAVELING SALESMAN PROBLEM as follows.

- G' is a complete graph on V.
- $d(u, v) = 1$ if $uv \in E$ and $d(u, v) = \rho \cdot n + 1$

Notice that

(a) if G has a hamiltonian cycle C, the corresponding tour along C has total distance n. Conversely, any TSP tour of total distance n in G' forms a hamiltonian cycle of G.

(b) any TSP tour of G' of total distance $> n$ has distance at least $(\rho + 1) \cdot n > \rho \cdot n$

Suppose that there is a ρ-approximation $\mathcal{A}$ for TRAVELING SALESMAN PROBLEM. We argue that HAMILTONIAN CYCLE can be solved in polynomial-time using $\mathcal{A}$, a contradiction. To see this, consider the following algorithm. First, we run the above polynomial-time reduction from HAMILTONIAN CYCLE to TRAVELING SALESMAN PROBLEM on the input G of HAMILTONIAN CYCLE, and apply the presumed ρ-approximation $\mathcal{A}$ on the constructed instance (G', d) to TRAVELING SALESMAN PROBLEM. Finally,

(I) If $\mathcal{A}$ outputs a TSP tour W of distance at most $\rho \cdot n$, we declare that G has a hamiltonian cycle.

(II) If $\mathcal{A}$ outputs a TSP tour W of distance strictly larger than $\rho \cdot n$, we declare that G does not have a hamiltonian cycle.

To complete the proof, it suffices to show that the output at Case I and II are correct respectively. For Case I, note that (b) implies that W actually has distance n and thus by the second part of (a), G has a hamiltonian cycle. For Case II, note that if G contains a hamiltonian cycle, then by the first part of (a) and that $\mathcal{A}$ is a ρ-approximation, the TSP tour W must have distance at most $\rho \cdot n$. Therefore, the output of II is correct. □

One of the essential variants of TRAVELING SALESMAN PROBLEM studied extensively in the literature is so-called METRIC TSP; here, the distance function satisfies the *triangle inequality*. That is, for any three vertices u, v, w of G it holds that

$$d(u, v) \leq d(u, w) + d(w, v).$$

METRIC TSP
Instance: an undirected graph $G = (V, E)$ and a distance function $\omega : E \to \mathbb{Q}$.
Goal: Find a cycle of minimum total distance visiting every vertex exactly once.

For METRIC TSP, there is a simple 2-approximation (works of independent groups of authors) and an 1.5-approximation attributed to Christofides. The latter remained as an algorithm achieving the best approximation ratio for the general[1] METRIC TSP until a randomized $(1.5 - 10^{36})$-approximation was proposed by Karlin, Klein, and Gharan (STOC 2021). As a deterministic one, it is not known if Christofides' algorithm can be improved.

2.1 2-Approximation for METRIC TSP

Let T be a minimum spanning tree G. Clearly T can be found in polynomial time using any of the polynomial-time algorithm for constructing a minimum spanning tree of a graph. Let C^* be an optimal TSP tour of (G, ω). The first observation is

$$\omega(T) \le \omega(C^*)$$

because C^* minus an edge of C^* is a spanning tree of G, therefore at least as large as the weight of a minimum spanning tree T.

Consider the auxiliary graph $H := (V, T + T)$, the graph on V obtained by doubling the edges of T. As all vertices of H has even degree (in H), it has an eulerian (closed) walk W. Let $W = w_0, w_1, \ldots, w_t = w_0$ be an eulerian walk of H and note that $\omega(W) = 2 \cdot \omega(T)$. Now we extract a hamiltonian cycle of G using the vertex sequence along W. We take the subsequence (not necessarily contiguous) of W

$$w_{i_0}, w_{i_1}, \ldots, w_{i_{n-2}}, w_{i_{n-1}}$$

defined as follows:

- $i_0 = 0$, i.e. $w_{i_0} = w_0$ and
- for every $1 \le j \le n-1$, i_j is the least index k such that $k \ge i_{j-1} + 1$ and w_k is distinct from all vertices preceding it on W.

Because all n vertices of G appear in the sequence W, such a subsequence exists. Let us add the edge $(w_{i_{n-1}}, w_{i_n})$ to this sequence with $w_{i_n} = w_{i_0}$ and let $\tilde{W} := w_{i_0}, w_{i_1}, \ldots, w_{i_{n-2}}, w_{i_{n-1}}, w_{i_n} = w_{i_0}$ be the resulting cycle of G. Clearly $\tilde{W}$ is a TSP tour. Now,

[1]A nicely compiled list of TRAVELING SALESMAN PROBLEM variants, albeit outdated, is found on the following thread of Stack Exchange; https://cstheory.stackexchange.com/questions/9241/approximation-algorithms-for-metric-tsp.

$$\begin{aligned}\omega(\tilde{W}) &= \sum_{j=0}^{n-1} \omega(w_{i_j}, w_{i_{j+1}}) \\ &\leq \sum_{j=0}^{n-1} \big(\omega(w_{i_j}, w_{i_j+1}) + \cdots + \omega(w_{i_{j+1}-1}, w_{i_{j+1}})\big) \\ &= \sum_{k=0}^{t-1} \omega(w_k, w_{k+1}) \\ &= \omega(W) \\ &= 2 \cdot \omega(T) \\ &\leq 2 \cdot \omega(C^*)\end{aligned}$$

where the second inequality is due to the triangle inequality.

2.2 1.5-Approximation for METRIC TSP

We modify the above 2-approximation algorithm as follows. At the stage when you create an auxiliary graph H, we do not double the edges of T, i.e a minimum spanning tree, and instead you take $H := (V, T + M)$, where M is a minimum weight (with respect to ω) perfect matching on the vertex set O consisting of those vertices having odd degree in T. Clearly $|O|$ is even[2] and there is a perfect matching in the graph $G[O]$.

We claim that $\omega(M) \leq 0.5 \cdot \omega(C^*)$ holds, which yields the claimed approximation ratio immediately:

$$\begin{aligned}\omega(\tilde{W}) &\leq \omega(T) + \omega(M) \\ &\leq \omega(C^*) + 0.5 \cdot \omega(C^*) \\ &\leq 1.5 \cdot \omega(C^*).\end{aligned}$$

To establish the inequality $\omega(M) \leq \omega(C^*)$, consider the TSP tour C^* and how it traverses the vertex set O. Let $o_0, o_1, o_2, \ldots, o_{2p-1}, o_{2p} = o_0$ be the sequence of vertices of O in the order of appearance on C^* (the first vertex $o_0 = o_{2p}$ is chosen arbitrarily). Let P_i be the subwalk of C^* from o_{i-1} to o_i. Observe

$$(\star) \qquad \begin{aligned}\sum_{i=1}^{2p} \omega(o_{i-1}, o_i) &\leq \sum_{i=1}^{2p} \omega(P_i) \\ &= \omega(C^*)\end{aligned}$$

where the first inequality holds due to the triangle inequality. Note that the edge set along the cycle $o_0, o_1, o_2, \ldots, o_{2p-1}, o_{2p} = o_0$ can be partitioned into two matchings, each covering all vertices of O; one matching takes all edges of the form (o_{2i}, o_{2i+1}) and the other matching takes all edges of the form (o_{2i-1}, o_{2i}). This partition and the inequality $(\star)$ together imply that there is a perfect matching of $G[O]$ of weight at most $0.5 \cdot \omega(C^*)$. As M is a minimum weight perfect matching of $G[O]$, we conclude that $\omega(M) \leq 0.5 \cdot \omega(C^*)$, as claimed.

[2]The number of odd degree vertices in any graph G is even thanks to so-called the Handshaking Lemma asserting $|\sum_{v \in V(G)} d_G(v)| = |E(G)|$.

3 Approximation for Steiner Tree and Metric Steiner Tree

Steiner Tree
Instance: a graph $G = (V, E)$, a set K of terminals, a weight function $\omega : E \to \mathbb{Q}$.
Goal: Find a steiner tree for K of minimum weight.

Metric Steiner Tree
Instance: a complete graph $G = (V, E)$, a set K of terminals, a weight function $\omega : E \to \mathbb{Q}$ satisfying the triangle inequality.
Goal: Find a steiner tree for K of minimum weight.

Approximation-factor Preserving Reduction. Let us construct an instance (G', K, ω') to Metric Steiner Tree from an arbitrary instance (G, K, ω) to Steiner Tree as follows.

- G' is a complete graph on the vertex set $V(G)$ (and K is the set of terminals for the new instance as well).
- $\omega'(u, v)$ is the weight of a shortest (w.r.t the weight ω) (u, v)-path in G. When there is no (u, v)-path in G, then we set $\omega'(u, v) = \infty$.

The proof of the next statement is left to the readers.

Lemma 2. *The following holds.*

1. *$\omega'(T) \leq \omega(T)$ for any tree T of G.*
2. *For any K-steiner tree T' of G', there exists a K-steiner tree T of G such that $\omega(T) \leq \omega'(T')$. Moreover, such T can be obtained in polynomial time.*

Using Lemma 2, one can readily see that an α-approximation for Metric Steiner Tree implies an α-approximation for Steiner Tree, and therefore the above reduction from Steiner Tree to Metric Steiner Tree is an *approximation-factor preserving reduction*. Given an input (G, K, ω) to Steiner Tree, use the above reduction to create an instance (G', K, ω') to Metric Steiner Tree. Let T' be a K-steiner tree of G' which achieves

$$\omega'(T') \leq \alpha \cdot \omega'(T^{**}),$$

where T^{**} is an optimal K-steiner tree of G'. Let T^* be an optimal K-steiner tree of G. By (1) of Lemma 2, we have

$$\omega'(T^{**}) \leq \omega'(T^*) \leq \omega(T^*).$$

On the other hand, (2) of Lemma 2 implies that there is a K-steiner tree T of G such that $\omega(T) \leq \omega'(T')$. Combining the three inequalities, we deduce

$$\omega(T) \leq \alpha \cdot \omega'(T^{**}) \leq \alpha \cdot \omega(T^*),$$

as claimed.

2-Approximation for METRIC STEINER TREE. Henceforth, we see how the 2-approximation for METRIC STEINER TREE works. The algorithm itself is simple: given an instance (G, K, ω) to METRIC STEINER TREE, find a minimum weight spanning tree T of $G[K]$ and return T as a K-steiner tree of G.

Let T^* be an optimal K-steiner tree of G. We want to show that $\omega(T) \leq 2 \cdot \omega(T^*)$. The main idea for proving this bound is similar to what we already saw for METRIC TSP and we sketch the proof here.

Consider the graph $H = (V(T), T + T)$, i.e. the graph obtained from T by doubling the edges. Let W be an eulerian walk of H. From the eulerian walk W, we extract a hamiltonian cycle C on K by shortcutting non-terminal vertices of W. Notice that $\omega(C) \leq \omega(W) = 2 \cdot \omega(T^*)$, where the first inequality is due to the triangle inequality. Choose an arbitrary edge e of C. Because the path $C - e$ is a spanning tree of $G[K]$ and T is a minimum weight spanning tree of $G[K]$, we know that $\omega(T) \leq \omega(C - e)$. We conclude $\omega(T) \leq 2 \cdot \omega(T^*)$.

4 Greedy algorithm for SET COVER and MAX COVERAGE

The following inequality will be often used for analysis.

$$1 + x \leq e^x \qquad \text{for all } x \in \mathbb{R}.$$

We consider the problems SET COVER and MAX COVERAGE and analyze a greedy algorithm for them.

SET COVER
Instance: a universe U, a collection $\mathcal{S}$ of subsets of U.
Goal: Find a subcollection $\mathcal{S}' \subseteq \mathcal{S}$ of minimum size (i.e. containing minimum possible sets of $\mathcal{S}$) such that $\bigcup_{S \in \mathcal{S}'} S = U$.

MAX COVERAGE
Instance: a universe U, a collection $\mathcal{S}$ of subsets of U. a positive integer k.
Goal: Find a subcollection $\mathcal{S}' \subseteq \mathcal{S}$ consisting of k sets which maximizes $|\bigcup_{S \in \mathcal{S}'} S|$

Consider the greedy algorithm which chooses a set following a simple, obvious criteria: choose a set in $\mathcal{S}$ which *newly* covers the maximum number of elements. For SET COVER, we do this greedy selection until all elements are covered and for MAX COVERAGE, we stop after choosing k sets.

Algorithm 2 Algorithm for SET COVER

```
1: procedure SetCover($U, \mathcal{S}$)
2:     Mark every element of $U$ as uncovered.
3:     $\mathcal{S}' \leftarrow \emptyset$.
4:     while $U$ has an uncovered element do
5:         Select $X \in \mathcal{S} \setminus \mathcal{S}'$ containing the maximum number of uncovered elements of $U$.
6:         $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{X\}$
7:         Mark every element of $X$ covered (unless it is already covered)
8:     return $\mathcal{S}'$.
```

We analyze the greedy algorithm. Notice that the only difference between Algorithms 2 and 3 is the stopping criterion of WHILE-loop. We develop some arguments which are common to both of them.

Algorithm 3 Algorithm for MAX COVERAGE

```
1: procedure MaxCoverage($U, \mathcal{S}, k$)
2:     Mark every element of $U$ as uncovered.
3:     $\mathcal{S}' \leftarrow \emptyset$.
4:     while $U$ has an uncovered element and $|\mathcal{S}'| < k$ do
5:         Select $X \in \mathcal{S} \setminus \mathcal{S}'$ containing the maximum number of uncovered elements of $U$.
6:         $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{X\}$
7:         Mark every element of $X$ covered (unless it is already covered)
8:     return $\mathcal{S}'$.
```

- Let $\mathcal{S}^*$ be an optimal set cover and let $U^* := \bigcup_{S \in \mathcal{S}^*} S$.
- For SET COVER, it holds that $U^* = U$ and for MAX COVERAGE, we have $|\mathcal{S}^*| = k$.
- For notational convenience, we also set $k := |\mathcal{S}^*|$ for SET COVER.
- Let $X_1, \ldots, X_i, \ldots, X_k$ be the chosen sets by the greedy algorithm.
- Let $g_t := |U^*| - |\bigcup_{i=1}^{t} X_i|$, i.e. the gap between the number of elements that an optimal solution can cover and what the greedy solution covers at t.

The key observation is that for every $t+1$ before the termination of the WHILE-loop, the elements in U_t^* can be covered by k sets (e.g. by an optimal set cover), so there exists a set $Y \in \mathcal{S}^*$ such that

$$|Y \cap (U^* \setminus C)| \geq \frac{1}{k} \cdot |U^* \setminus C| \geq \frac{1}{k} \cdot (|U^*| - |C|) = \frac{g_t}{k},$$

where $C = \bigcup_{i=1}^{t} X_i$. As X_{t+1} chosen so as to maximize $|X_{t+1} \setminus C|$, we have

$$g_t - g_{t+1} = |X_{t+1} \setminus C| \geq |Y \setminus C| = |Y \cap (U^* \setminus C)| \geq \frac{g_t}{k},$$

and

$$g_{t+1} \leq (1 - \frac{1}{k}) \cdot g_t \leq (1 - \frac{1}{k})^{t+1} \cdot g_0.$$

Here, notice that $g_0 = |U^*|$.

Concluding the analysis for MAX COVERAGE. The output set cover $\{X_1, \ldots, X_k\}$ covers as many as $|\bigcup_{i=1}^{t} X_i| = g_0 - g_k$ elements, which is at least

$$(1 - (1 - \frac{1}{k})^k) \cdot g_0 = (1 - (1 - \frac{1}{k})^k) \cdot |U^*| \geq (1 - \frac{1}{e}) \cdot |U^*|.$$

Therefore, the greedy algorithm is a $(1 - e^{-1})$-approximation for MAX COVERAGE.

Concluding the analysis for SET COVER. Let us fix a minimum value d such that $g_t \leq k$, i.e. the number of uncovered elements after d-th iteration is at most k. Then one can choose at most k additional sets to cover the remaining $\leq k$ elements. Let $n := |U|$, and note that $g_0 = n$. In order to achieve $g_d \leq k$, it suffices that d satisfies

$$(1 - \frac{1}{k})^d \cdot g_0 = (1 - \frac{1}{k})^d \cdot n \leq k,$$

or equivalently,

$$d \geq -\frac{\ln \frac{n}{k}}{\ln (1 - \frac{1}{k})}.$$

From

$$1 - \frac{1}{k} \leq e^{-1/k},$$

we know that setting $d := k \ln n/k$ satisfies the desired condition. Therefore, the output set cover of the greedy algorithm has size at most

$$k \ln \frac{n}{k} + k = (1 + \ln \frac{n}{\texttt{opt}}) \cdot \texttt{opt} = \alpha(n) \cdot \texttt{opt}$$

for $\alpha(n) = O(\log n)$. To summarize, the algorithm is an $O(\log n)$-approximation for SET COVER.