1 Approximation Algorithms: Vertex Cover

1.1 Introduction to Approximation Algorithms

There are several optimization problems such as Minimum Spanning Tree (MST), Min-Cut, Maximum-Matching, in which you can solve this exactly and efficiently in polynomial time. But many practical significant optimization problems are NP-Hard, in which we are unlikely to find an algorithm that solve the problem exactly in polynomial time. Examples of the standard NP-Hard problems with some of their brief description are as following:

- Traveling Salesman Problem (TSP) - finding a minimum cost tour of all cities
- Vertex Cover - find minimum set of vertex that covers all the edges in the graph (we will describe this in more detail)
- Max Clique
- Set Cover - find a smallest size cover set that covers every vertex
- Shortest Superstring - given a set of string, find a smallest subset of strings that contain specified words

These are NP-Hard problems, i.e., If we could solve any of these problems in polynomial time, then P = NP. An example of problem that is not known to be either NP-Hard: Given 2 graphs of n vertices, are they the same up to permutation of vertices? This is called Graph Isomorphism. As of now, there is no known polynomial exact algorithm for NP-Hard problems. However, it may be possible to find a near-optimal solutions in polynomial time. An algorithm that runs in polynomial time and outputs a solution close to the optimal solution is called an approximation algorithm. We will explore polynomial-time approximation algorithms for several NP-Hard problem.

Definition: Let P be a minimization problem, and I be an instance of P. Let A be an algorithm that finds feasible solution to instances of P. Let $A(I)$ is the cost of the solution returned by A for instance I, and $OPT(I)$ is the cost of the optimal solution (mimimum) for I. Then, A is said to be an α-*approximation* algorithm for P if

$$\forall I, \quad \frac{A(I)}{OPT(I)} \quad \leq \quad \alpha \tag{1}$$

where $\alpha \geq 1$. Notice that since this is a minimum optimization problem $A(I) \geq OPT(I)$. Therefore, 1-approximation algorithm produces an optimal solution, an an approximation algorithm with a large α may return a solution that is much worse than optimal. So the smaller α is, the better quality of the approximation the algorithm produces.
For instance size n, the most common approximation classes are:

$\alpha = O(n^c)$ for $c < 1$, e.g. Clique.

$\alpha = O(log\, n)$, e.g. Set Cover.

$\alpha = O(1)$, e.g. Vertex Cover.

$\alpha = 1 + \varepsilon$, $\forall \varepsilon > 0$, this is called *Polynomial-time Approximation Scheme (PTAS),* e.g. certain scheduling problems.

$\alpha = 1 + \varepsilon$ in time that is polynomial in $(n, \frac{1}{\varepsilon})$, this is called *Fully Polynomial-time approximation Scheme (FPTAS)*, e.g. Knapsack, Subset Sum.

Now, let us consider an approximation algorithm for NP-Hard problem, Vertex Cover.

1.2 Approximation Algorithm for Vertex Cover

Given a $G = (V, E)$, find a minimum subset $C \subseteq V$, such that C *"covers"* all edges in E, i.e., every edge $\in E$ is incident to at least one vertex in C.

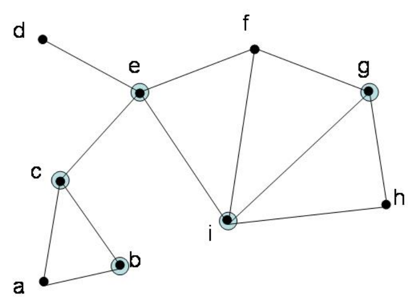

Figure 1: An instance of Vertex Cover problem. An optimal vertex cover is {b, c, e, i, g}.

Algorithm 1: APPROX-VERTEX-COVER(G)

$C \leftarrow \emptyset$
while $E \neq \emptyset$
  pick any $\{u, v\} \in E$
  $C \leftarrow C \cup \{u, v\}$
  delete all eges incident to either u or v

return C

As it turns out, this is the best approximation algorithm known for vertex cover. It is an open problem to either do better or prove that this is a lower bound.

OBSERVATION: The set of edges picked by this algorithm is a matching, no 2 edges touch each other (edges disjoint). In fact, it is a *maximal matching.* We can then have the following alternative description of the algorithm as follows.

Find a maximal matching M
Return the set of end-points of all edges $\in M$.

1.3 Analysis of Approximation Algorithm for VC

Claim 1: This algorithm gives a vertex cover
Proof: Every edge $\in M$ is clearly covered. If an edge, $e \notin M$ is not covered, then $M \cup \{e\}$ is a matching, which contradict to maximality of M. ■

Claim 2: This vertex cover has size $\leq 2\times$minimum size (optimal solution)
Proof:

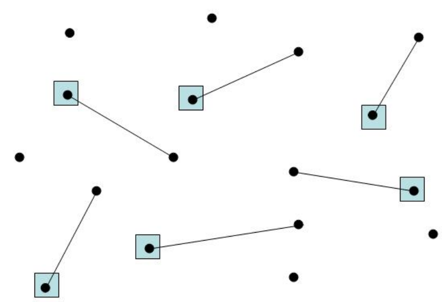

Figure 2: Another instance of Vertex Cover and its optimal cover shown in blue squares

The optimum vertex cover must cover every edge in M. So, it must include at least one of the endpoints of each edge $\in M$, where no 2 edges in M share an endpoint. Hence, optimum vertex cover must have size

$$OPT(I) \geq |M|$$

But the algorithm A return a vertex cover of size $2|M|$, so $\forall I$ we have

$$A(I) \; = \; 2|M| \; \leq \; 2 \times OPT(I)$$

implying that A is a *2-approximation* algorithm. ■

We know that the optimal solution is intractable (otherwise we can probably come up with an algorithm to find it). Thus, we cannot make a direct comparison between algorithm A's solution and the optimal solution. But we can prove Claim 2 by making indirect comparisons of A's solution and the optimal solution with the size of the maximal matching, $|M|$. We often use this technique for approximation proofs for NP-Hard problems, as you will see later on.

But is $\alpha = 2$ a tight bound for this algorithm? Is it possible that this algorithm can do better than 2-approximation? We can show that 2-approximation is a tight bound by a tight example:
TIGHT EXAMPLE: Consider a complete bipartite graph of n black nodes on one side and n red nodes on the other side, denoted $K_{n,n}$.
Notice that size of any maximal matching of this graph equals n,

$$|M| = n$$

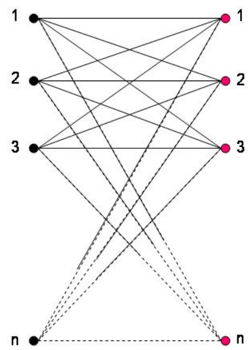

Figure 3: $K_{n,n}$- complete bipartite graph

so the APPROX-VERTEX-COVER(G) algorithm returns a cover of size $2n$.

$$A(K_{n,n}) = 2n$$

But, clearly the optimal solution $= n$.

$$OPT(K_{n,n}) = n$$

Note that a tight example needs to have arbitrarily large size in order to prove tightness of analysis, otherwise we can just use brute force for small graphs and A for large ones to get an algorithm that avoid that tight bound. Here, it shows that this algorithm gives 2-approximation no matter what size n is.

2 Approximation Algorithms: Traveling Salesman Problem

2.1 Last time: α-approximation algorithms

Definition: For a minimization (or maximization) problem P, A is an *α-approximation algorithm* if for every instance I of P, $\frac{A(I)}{\mathrm{OPT}(I)} \leq \alpha$ (or $\frac{\mathrm{OPT}(I)}{A(I)} \leq \alpha$).

Last time we saw a 2-approximation for Vertex Cover [CLRS 35.1]. Today we will see a 2-approximation for the Traveling Salesman Problem (TSP) [CLRS 35.2].

2.2 Definition

A salesman wants to visit each of n cities exactly once each, minimizing total distance travelled, and returning to the starting point.

Traveling Salesman Problem (TSP).

Input: a complete, undirected graph $G = (V, E)$, with edge weights (costs) $w : E \longrightarrow \mathbb{R}^+$, and where $|V| = n$.

Output: a tour (cycle that visits all n vertices exactly once each, and returning to starting vertex) of minimum cost.

2.3 Inapproximability Result for General TSP

Theorem: For any constant k, it is NP-hard to approximate TSP to a factor of k.

Proof: Recall that Hamiltonian Cycle (HC) is NP-complete (Sipser). The definition of HC is as follows.

Input: an undirected (not necessarily complete) graph $G = (V, E)$.

Output: YES if G has a Hamiltonian cycle (or tour, as defined above), NO otherwise.

Suppose A is a k-approximation algorithm for TSP. We will use A to solve HC in polynomial time, thus implying P = NP.

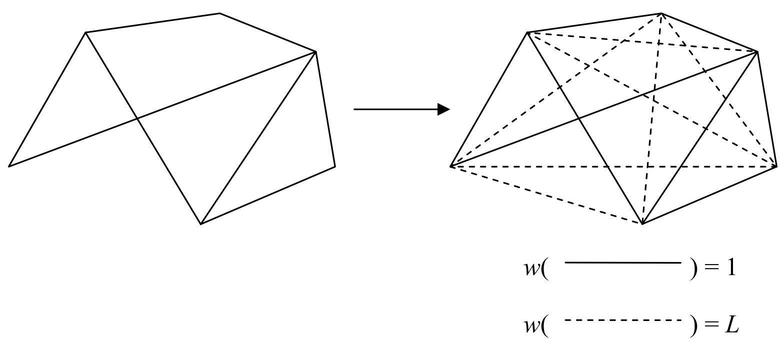

Figure 4: Example of construction of G' from G for HC-to-TSP-approximation reduction.

Given the input $G = (V, E)$ to HC, we modify it to construct the graph $G' = (V', E')$ and weight function w as input to A as follows (Figure 4). Let all edges of G have weight 1. Complete the resulting graph, letting all new edges have weight L for some large constant L. The algorithm for HC is then:

Algorithm 2: HC-Reduction(G)

1 Construct G' as described above.
2 **if** $A(G')$ returns a 'small' cost tour ($\leq kn$) **then**
3   **return** YES
4 **if** $A(G')$ returns a 'large' cost tour ($\geq L$) **then**
5   **return** NO

It then remains to choose our constant $L \geq kn$, to ensure that the 2 cases are clearly differentiated. ■

2.4 Approximation Algorithm for Metric TSP

Definition. A *metric space* is a pair (S, d), where S is a set and $d : S^2 \longrightarrow \mathbb{R}^+$ is a distance function that satisfies, for all $u, v, w \in S$, the following conditions.

1. $d(u, v) = 0$
2. $d(u, v) = d(v, u)$

3. $d(u,v) + d(v,w) \geq d(u,w)$ (triangle inequality)

For a complete graph $G = (V, E)$ with cost $c : E \longrightarrow \mathbb{R}^+$, we say "the costs form a metric space" if $(V, \hat{c})$ is a metric space, where $\hat{c}(u,v) := c(\{u,v\})$.

Given this restriction (in particular, the addition of the triangle inequality condition), we have the following simple approximation algorithm for TSP.

Algorithm 3: MetricTSPApprox(G)

1 Compute a weighted MST of G.
2 Root MST arbitrarily and traverse in pre-order: $v_1, v_2, \ldots, v_n$.
3 Output tour: $v_1 \to v_2 \to \cdots \to v_n \to v_1$.

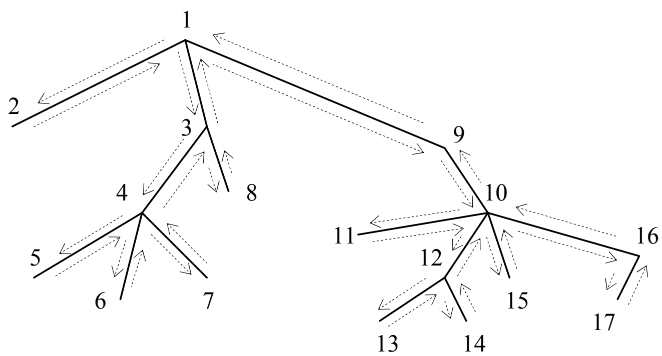

Figure 5: Example MST, where the output tour would be $1 \to 2 \to \cdots \to 17 \to 1$.

2.5 Analysis of Approximation Algorithm for Metric TSP

On an instance I of TSP, let us compare $A(I)$ to $\mathrm{OPT}(I)$, via the intermediate value $\mathrm{MST}(I)$ (the weight of the MST).

Claim: Comparing $A(I)$ to $\mathrm{MST}(I)$: $A(I) \leq 2 \times \mathrm{MST}(I)$.
Proof: Let σ be a *full walk* along the MST in pre-order (that is, we revisit vertices as we backtrack through them). In Figure 5, σ would be the path along all the arrows, wrapping around the entire MST, namely, $1 \to 2 \to 1 \to 3 \to 4 \to 5 \to 4 \to 6 \to 4 \to \cdots \to 1$. It is clear that $\mathrm{cost}(\sigma) = 2 \times \mathrm{MST}(I)$. Now, the tout output by A is a subsequence of the full walk σ, so by the triangle inequality:

$$A(I) \leq \mathrm{cost}(\sigma) = 2 \times \mathrm{MST}(I)$$

proving our claim. ■

Claim: Comparing $\mathrm{OPT}(I)$ to $\mathrm{MST}(I)$: $\mathrm{OPT}(I) \geq \mathrm{MST}(I)$.
Proof: Let σ^* be an optimum tour, that is, $\mathrm{cost}(\sigma^*) = \mathrm{OPT}(I)$. Deleting an edge from σ^* results in a spanning tree T, whose cost by definition is $\mathrm{cost}(T) \geq \mathrm{MST}(I)$. Hence,

$$\mathrm{OPT}(I) = \mathrm{cost}(\sigma^*) \geq \mathrm{cost}(T) \geq \mathrm{MST}(I)$$

as required. ■

Combining these 2 claims, we get:

$$A(I) \leq 2 \times \mathrm{MST}(I) \leq 2 \times \mathrm{OPT}(I)$$

Hence, A is a 2-approximation algorithm for (Metric) TSP.

2.6 Concluding Remarks

It is possible (and relatively easy) to improve the approximation factor to 3/2 for Metric TSP. Note that in the original wording of the problem, with the salesman touring cities, the cost (distance) function is in fact even more structured than just a metric. Here, we have Euclidean distance, and as it turns out, this further restriction allows us to get a PTAS, although this is a more difficult algorithm.