COMPSCI 632: Approximation Algorithms September 4, 2017

Lecture 3

Lecturer: Debmalya Panigrahi *Scribe: Reza Alijani*

1 Overview

In this lecture, we introduce the *TSP* and *Stainer tree problems*. We give MST based approximation algorithms for the restricted version of them called *metric TSP* and *metric Steiner tree*. We then consider two clustering problems: *metric k-center* and *metric k-median*. We give a greedy approximation algorithm for the metric k-center and a local search approximation algorithm for the metric k-median.

2 Metric Space

All of the algorithms in this lecture works when the input is a metric space. In this section, we define a metric space:

Definition 1. *A **metric space** is a pair (V,d) where V is a set of points and $d : V \times V \to \mathbb{R}_{\geq 0}$ is a distance function satisfying the following properties:*

- $d(i,i) = 0$
- $d(i,j) = d(j,i)$
- $d(i,j) + d(j,k) \geq d(i,k)$ *(*triangle inequality*)*

Equivalently, we can think of the metric space as a complete undirected graph with nonnegative edge costs such that for any pair of vertices (i,j) the edge (i,j) is a shortest path between i and j. In other words, edge costs satisfy the triangle inequality.

3 Metric Traveling Salesman

The traveling salesman problem is defined as follows:

Definition 2. *The traveling salesman problem (TSP): Given a complete undirected graph $G = (V,E)$ with nonnegative edge costs $d(.)$, find a minimum cost cycle visiting every vertex of G exactly once.*

It turns out that this problem is hard to approximate, unless P=NP. However, if edge costs satisfy the triangle inequality (the input is a metric space) we can give constant factor approximation algorithms for the problem. This problem is called *metric TSP*. In the rest of this lecture, we assume that the input is a metric space.

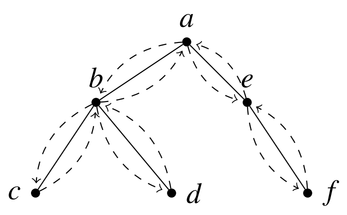

Figure 1: An example of in-order traversal of the MST is : *abcbdbaefea*

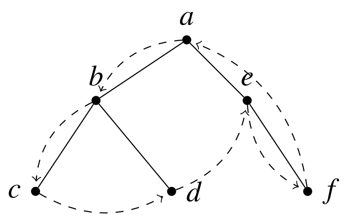

Figure 2: An example of shortcutting applied on the closed walk of Figure 1. The resulting cycle is *abcdefa*.

3.1 Factor 2 approximation using MST

We present a 2-approximation algorithm which uses the *minimum spanning tree (MST)* of the input graph to find a solution for TSP. Let $OPT(TSP)$ and $OPT(MST)$ be the cost of the optimal solution of the TSP and MST, respectively. Removing any edge of the optimal solution of TSP generates a spanning tree. Since the cost of each edge is nonnegative, the cost of those spanning trees are not more than $OPT(TSP)$. Therefore, the cost of the minimum spanning tree is at most $OPT(TSP)$:

$$OPT(TSP) \geq OPT(MST) \tag{1}$$

Let T be the MST of the graph G. We use the in-order traversal of T to visits every node of the graph G. The total cost of this walk is $2OPT(MST)$ because it uses every edge of T exactly twice. You can see an example of such a walk in Figure 1.

The result of the in-order traversal of the MST is a closed walk (with repeated vertices and edges) but in TSP the goal is to find a cycle. We use *shortcutting* to solve the problem: whenever the traversal repeats a vertex, shortcut directly to the next unvisited vertex. An example of shortcutting can be seen in Figure 2. Repeated use of the triangle inequality can show that shortcutting does not increase the total cost. If we use the shortcutting on a closed walk with total cost C we can obtain a cycle with total cost at most C. We state this as a fact:

Fact 1. *Given a complete graph G with nonnegative edge costs satisfying the triangle inequality, applying shortcutting on a closed walk which visits every node of G and its total cost is C gives us a cycle which visits every vertex of G exactly once and its total cost is at most C .*

Theorem 2. *The MST-based algorithm using in-order traversal and shortcutting for metric TSP is a 2 approximation.*

Proof. The algorithm uses the in-order traversal of MST which visits every edge of the MST twice. Therefore, total cost of the resulting closed walk is $2OPT(MST)$. Then, it applies shortcutting process on the closed walk which returns a feasible solution for MST (a cycle visiting every vertex exactly once) with total cost of at most $2OPT(MST)$. Therefore, based on inequality (1) the total cost of the solution is at most $2OPT(TSP)$. □

3.2 Christofides Algorithm

In Fact 1 we have seen that it is enough to find a closed walk for the problem which visits every vetrex. One way of finding such a closed walk is using an Eulerian circuit:

Definition 3. *Given a connected graph G, an* ***Eulerian circuit*** *is a circuit that visits every edge of the graph G exactly once.*

Note that an Eulerian circuit visits all the vertices of the graph. Therefore, finding an Eulerian circuit with total cost of C guarantees a solution to the TSP with total cost of at most C. We use the following theorem to find the Eulerian circuit:

Theorem 3. *A connected graph has an Eulerian circuit if and only if the degree of each vertex is even. In addition, an Eulerian circuit of a graph (if there exists one) can be found in polynomial time.*

We omit the proof of the theorem. We can think of the 2-approximation algorithm in Section 3.1 as finding an Eulerian circuit in a graph that has two copies of each edge of the MST. Note that when there are two copies of each edge, all the degrees are even. Therefore, there always exists an Eulerian circuit in that graph (Theorem 3 holds for multigraph as well). The total cost of any Eulerian circuit of that graph (including in-order traversal) is $2OPT(MST)$ because it visits every edge exactly once. Note that this circuit in the new graph is a closed walk (can repeat edges) in the first graph because we add second copy of the edges. However, Fact 1 holds for closed walk. Therefore, we can make multiple copies of one edge and find the Eulerian cycle in the new graph.

The previous algorithm adds another copy of each MST edge. However, the degree of some of the vertices was already even and we might be able to add fewer edges to satisfy the even degree vertices condition. Consider the following lemma:

Lemma 4. *Handshaking lemma: The number of odd degree vertices in a finite graph is even.*

Proof. The sum of the degrees in a graph is two times the number of edges because each edge is counted in the degree of its two endpoints. Therefore, sum of the degrees is an even number and there should be even number of odd degrees in the sum. □

We can consider the odd degree vertices of the MST T and pair them up by adding an edge between each pair (since the graph is complete and there are even number of odd degree vertices we can always do this). The natural choice is minimum perfect matching among the odd degree vertices of T. Then, we can find the Eulerian circuit in the new graph. The complete algorithm is as follows [Chr76]:

Algorithm 1 Christofides Algorithm

```
  $T \leftarrow MST$ of G
  $M \leftarrow$ minimum matching of the odd degree vertices of $T$
  $G' \leftarrow T \cup M$
  $C \leftarrow$ Eulerian circuit of $G'$
  $C' \leftarrow$ the resulting cycle after applying shortcutting on $C$
  return $C'$
```

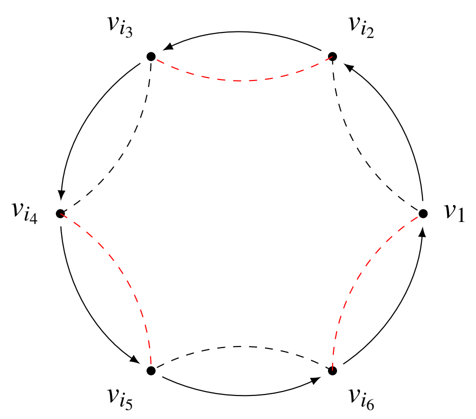

Figure 3: An example with 6 odd degree vertices. The black dashed edges show perfect matching M_1 and the red dashed edges show the perfect matching M_2. Based on the traingle inequlity we know that the cost of the path between each two consecutive odd degree vertices in the cycle C is not less than the cost of the direct edge between them. Therefore, $c(M_1)+c(M_2) \leq c(C)$

Theorem 5. *Christofides algorithm is a* $3/2$ *approximation for metric TSP.*

Proof. Let $c(M)$ be the cost of the minimum matching on the odd vertices of T. The cost of the Eulerian circuit is $OPT(MST)+c(M)$ because we use each edge exactly once. Based on Fact 1 and inequality 1 it suffices to show that $c(M) \leq OPT(TSP)/2$.

For this purpose, consider the optimal solution of TSP: let $C=(v_1,v_2,\dots,v_n,v_1)$ be an optimal solution which is a cycle that visits every vertex of G exactly once. Without loss of generality assume that v_1 is an odd degree vertex of T. Let $(v_1=v_{i_1},v_{i_2}\dots,v_{i_{2k}})$ be the sequence of odd degree vertices of T with the same order that they appear on the cycle C if we start the cycle from v_1. In other words, $i_1<i_2<\cdots<i_{2k}$. Consider the following two matchings of the odd vertices of T:

$$M_1=\{(v_{i_1},v_{i_2}),(v_{i_3},v_{i_4}),\dots,(v_{i_{2k-1}},v_{i_{2k}})\}$$

$$M_2=\{(v_{i_2},v_{i_3}),(v_{i_4},v_{i_5}),\dots,(v_{i_{2k-2}},v_{i_{2k-1}}),(v_{i_{2k}},v_{i_1})\}.$$

Now consider $M_1\cup M_2$:

$$M_1\cup M_2=\{(v_{i_1},v_{i_2}),(v_{i_2},v_{i_3}),\dots,(v_{i_{2k-1}},v_{i_{2k}}),(v_{i_{2k}},v_{i_1})\}.$$

We can use the triangle inequality and see that the cost of $M_1\cup M_2$ is at most $OPT(TSP)$:

$$c(M_1\cup M_2)=c(M_1)+c(M_2)=d(v_1,v_{i_2})+d(v_{i_2},v_{i_3})+\cdots+d(v_{i_{2k-1}},v_{i_{2k}})+d(v_{i_{2k}},v_1)\leq$$

$$(d(v_1,v_2)+d(v_2,v_3)+\cdots+d(v_{i_2-1},v_{i_2}))+(d(v_{i_2},v_{i_2+1})+d(v_{i_2+1},v_{i_2+2})+\cdots+d(v_{i_3-1},v_{i_3}))$$
$$+\cdots+(d(v_{i_{2k-1}},v_{i_{2k-1}+1})+d(v_{i_{2k-1}+1},v_{i_{2k-1}+2})+\cdots+d(v_{i_{2k}-1},v_{i_{2k}}))$$
$$+\cdots+(d(v_{i_{2k}},v_{i_{2k}+1})+d(v_{i_{2k}+1},v_{i_{2k}+2})+\cdots+d(v_{n-1},v_n)+d(v_n,v_1))=c(C)$$

Figure 3 illustrates an example with 6 odd degree vertices.

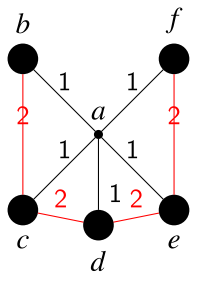

Figure 4: The Steiner tree must contain the nodes shown by large circles ($T = \{b,c,d,e,f\}$). In addition, the graph is a complete graph and for each edge which is not drawn in the figure the cost is the shortest path between the two endpoints. The cost of optimal tree is 5 (black edges) and the cost of the MST of $G[T]$ is 8 (red edges).

Based on the above inequality, we can say that the cost of the smaller matching between M_1 and M_2 is not more than half of the cost of the cycle. On the other hand, we know that M is a minimum matching and smaller than both of those matchings:

$$c(M) \leq \min(c(M_1), c(M_2)) \leq c(C)/2 = OPT(TSP)/2.$$

□

4 Steiner Tree

Definition 4. *Steiner Tree: Given a graph $G = (V,E)$ with non-negative edge costs and a subset T of the vertices, find the minimum cost tree that contains all the vertices of T.*

Again, we consider the *metric Steiner tree problem* in which the points lie in a metric space. Note that in the metric Steiner tree problem, the induced subgraph containing only the vertices in T (which is denoted by $G[T]$) is connected because G is a complete graph. Therefore, the MST of $G[T]$ is a valid solution. However, as we can see in the Figure 4 this is not necessarily optimal. We claim that this solution is a 2-approximation:

Theorem 6. *The MST of the induced subgraph $G[T]$ is a 2-approximation.*

Proof. We use the same argument as we used in Section 3.1. Consider the optimal Steiner tree: let OPT be the cost of this optimal tree. If we use in-order traversal (starting from a vertex in T) of the optimal Steiner tree and shortcut every vertex in $V - T$ and every repeated vertex of T, we will find a path that visits all the vertices of T (which is a spanning tree) with cost at most $2OPT$. Therefore, we can conclude that the cost of the MST is not more than $2OPT$. □

5 Clustering

Clustering is the problem in which the goal is to group the given data (points) in a way that similar data (close points) end up in the same group. There are different variants of the clustering problem. In this section, we consider the centroid based variant: Given a set V of n points in a metric space, we wish to find a set $S = \{c_1, c_2, \ldots, c_k\} \subset V$ ($k << n$) of the points as centers and assign each point $j \in V$ to the cluster of the closest center denoted by $\sigma_S(j)$. In other words, $\sigma_S(j) = \arg\min_{i \in S} d(i,j)$ where $d(.)$ is the distance function between two points of the metric space. The goal is to find the set S with the smallest objective $F(S)$, which can be defined is various ways. We will examine two variants of the problem (two different objective functions):

- The *k-center problem*: The goal is to minimize the distance between any point and its closest center. In other words, we want to minimize the following objective function:

$$F(S) = \max_{j \in V} d(j, \sigma_S(j))$$

- The *k-median*: The goal is to minimize the sum of the distances of the points to their closest center:

$$F(S) = \sum_{j \in V} d(j, \sigma_S(j))$$

5.1 Greedy algorithm for k-center

Consider the following natural greedy algorithm:

Algorithm 2 Greedy Algorithm for k-center

```
Choose an arbitrary $c_1 \in V$
$S_1 \leftarrow \{c_1\}$
for $i = 1$ to $k-1$ do
    $c_{i+1} \leftarrow \arg\max_{j \in V - S_i} d(j, \sigma_{S_i}(j))$
    $S_{i+1} \leftarrow S_i \cup \{c_{i+1}\}$
end for
return $S_k$
```

Theorem 7. *The greedy algorithm is a 2-approximation for k-center.*

Proof. Let S^* be the optimal solution and $F(S^*) = R$. If we consider a ball of radius R around each $c^* \in S^*$, any of the points in V (including the centers chosen by greedy algorithm denoted by $S = S_k$) are contained in at least one of these balls. We consider two different possibilities:

- If there is a center $c^* \in S^*$ such that the ball around it contains $c_i, c_j \in S$: without loss of generality assume that c_i has been chosen after c_j by the greedy algorithm ($j < i$). This means that when c_i was chosen the distance between any point in V and the closest center in S_{i-1} was at most $d(c_i, c_j)$. On the other hand based on the triangle inequality and the fact that both of c_i and c_j contains in the ball of radius R around c^*, we have:

$$d(c_i, c_j) \leq d(c_i, c^*) + d(c^*, c_j) \leq 2R$$

  Therefore, we can conclude that the distance between any point in V and the closest center to it in S_{i-1} is at most $2R$. On the next steps by adding new centers this distance can only decrease, completing the argument for this case.

- The ball around each center $c^* \in S^*$ contains exactly one of the centers of the greedy algorithm. Let c_i^* denote the optimal center whose ball contains $c_i \in S$. In this case, we know that there is a center $c_i^* \in S^*$ for each point $j \in V$ such that $d(j, c_i^*) \leq R$. Based on the triangle inequality $d(j, c_i) \leq d(j, c_i^*) + d(c_i^*, c_i) \leq 2R$. Therefore, for each point j, $d(j, \sigma_S(j)) \leq d(j, c_i) \leq 2R$, and thus the proof is complete.

□

5.2 Local Search algorithm for k-median

In this section, we present a local search algorithm for the k-median problem. In a local search algorithm, we start with an arbitrary feasible solution of the problem. We define a local move and at each step check if there is a local move from the current solution that can improve the objective by moving to a neighboring feasible solution.

For the k-median problem, the initial feasible solution is a set of k arbitrary points of V as centers. On each step we check if there is a local move that can reach a better solution. If there is such a move we use it and find a better solution. Otherwise, we stop and return the local optimum as the solution. The local move of this problem is to swap a single point in the current solution with another point. Let $swap(i, j)$ denote the local move that update S to $(S \cup \{j\}) - \{i\}$.

Lemma 8. *The local move algorithm terminates.*

Proof. At each step we move to a solution with k points which is strictly better (the objective function is strictly smaller) than the previous solution. Therefore, we never revisit a solution by a sequence of local moves. On the other hand, there are $\binom{n}{k}$ (finite) different feasible solutions for this problem. Thus, the algorithm terminates. □

We omit the modification of the algorithm that yields a polynomial time algorithm. You can find it in [WS11].

Theorem 9. *The local search algorithm is a 5-approximation for the k-median problem.*

Proof. Let S be the set of the centers chosen by the local search algorithm and S^* be the set of the centers of an optimal solution. We partition the points in S to 3 different sets:

- A: The set of points $i \in S$ such that there exists exactly one point $i^* \in S^*$ for which $\sigma_S(i^*) = i$.
- B: The set of points $i \in S$ such that there exists no point $i^* \in S^*$ for which $\sigma_S(i^*) = i$.
- C: The set of points $i \in S$ such that there exists at least two points $i^*, i'^* \in S^*$ for which $\sigma_S(i^*) = \sigma_S(i'^*) = i$.

Let A^* denote the set of all the centers of S^* corresponding to a center in A, and C^* be $S^* - A^*$. We can see that $|A| = |A^*|$ because for any center $i \in A$ there is a corresponding $i^* \in S^*$ such that $\sigma_S(i^*) = i$. Also each center in C corresponds to at least two distinct centers in C^*. Therefore, $|C| \leq |C^*|/2$. Finally we have:

$$|S| - |A| = k - |A| = |B| + |C|$$

And:

$$|S^*| - |A^*| = k - |A^*| = k - |A| = |C^*| \Rightarrow |C^*| = |B| + |C| \leq |B| + |C^*|/2 \Rightarrow |B| \geq |C^*|/2$$

Since S is a local optimum there is no local move that can decrease the objective value. Consider the following subset of local moves:

- 1) $\forall i \in A$: $swap(i, \sigma_S^{-1}(i))$
- 2) For each center in B we specify one or two arbitrary and distinct centers at C^* in a way that each center in C^* will be matched to exactly one center in B. Note that this is always possible because $|C^*/2| \leq |B| \leq |C^*|$. Then, consider swapping the center in B with the corresponding center(s) in $|C^*|$.

We will find an upper bound for the change in the cost of the solution after each of the swaps mentioned above. Consider a swap (i, i^*). After this swap, we will consider an updated solution that is constructed as follows:

- 1) If $\sigma_{S^*}(j) = i^*$, then j is now assigned to i^*.
- 2) If $\sigma_{S^*}(j) \neq i^*$ and $\sigma_S(j) = i$, then j is now assigned to $\sigma_S(\sigma_{S^*}(j))$.

Note that if $\sigma_S(j) \neq i$ then 0 is an upper bound for the change of the term corresponding to point j in the objective function. Therefore, we can only consider the mentioned updates to find an upper bound on the change of the cost after this move.

We need to show that the above updates are valid, i.e., each point is assigned to a chosen center after this move. Update 1 is valid because i^* will be in the set S after the swap. For update 2, we need to show that $\sigma_S(\sigma_{S^*}(j)) \neq i$ because i is the only center of S that will not remain in the solution the swap: if $i \in A$ then i^* is the only center in S^* such that $\sigma_S(i^*) = i$. Therefore if $\sigma_{S^*}(j) \neq i^*$, then $\sigma_S(\sigma_{S^*}(j))$ cannot map to i, but this is true based on the first condition of the update 2. If i is in B we know that for no center i^* in S^* (including $\sigma_{S^*}(j)$), $\sigma_S(i^*) = i$. Therefore, these updates are valid. Note that after these updates, j is not necessarily the closest center after the update. However, since we only want to find an upper bound for the change after each of these swaps, we only need to assign each point to some center after the move (assigning each point to its closest center will only improve the objective of the solution).

Therefore, the change in the cost after swap (i, i^*) which is denoted by $\Delta(swap(i, i^*))$ can be upper bounded:

$$\Delta(swap(i,i^*)) \leq \sum_{j:\sigma_{S^*}(j)=i^*} (d(i^*, j) - d(\sigma_S(j), j)) + \sum_{j:\sigma_{S^*}(j)\neq i^* \text{and } \sigma_S(j)=i} (d(\sigma_S(\sigma_{S^*}(j)), j) - d(\sigma_S(j), j))$$

Based on the triangle inequality and the definition of σ_S we find can upper bound $d(\sigma_S(\sigma_{S^*}(j)), j)$ as follows:

$$d(\sigma_S(\sigma_{S^*}(j)), j) \leq d(\sigma_S(\sigma_{S^*}(j)), \sigma_{S^*}(j)) + d(\sigma_{S^*}(j), j) \leq d(i, \sigma_{S^*}(j)) + d(\sigma_{S^*}(j), j) \leq$$

$$d(i, j) + d(j, \sigma_{S^*}(j)) + d(\sigma_{S^*(j)}, j) = d(i, j) + 2d(\sigma_{S^*(j)}, j).$$

Now, we can find an upper bound for the whole expression inside the second summation by using the obtained upper bound for its first term and the fact that $\sigma_S(j) = i$ in the summation:

$$d(\sigma_S(\sigma_{S^*}(j)), j) - d(\sigma_S(j), j)) \leq d(i, j) + 2d(\sigma_{S^*(j)}, j) - d(\sigma_S(j), j)) = 2d(j, \sigma_{S^*}(j)).$$

Then, we can write the final upper bound after $swap(i, i^*)$:

$$\Delta(swap(i,i^*)) \leq \sum_{j:\sigma_{S^*}(j)=i^*} (d(i^*, j) - d(\sigma_S(j), j)) + \sum_{j:\sigma_{S^*}(j)\neq i^* \text{and } \sigma_S(j)=i} 2d(j, \sigma_{S^*}(j)).$$

If we sum up these upper bounds over all the subset of swaps that we mentioned earlier it should be a non-negative value. This is because the current solution is a local optimum and no swaps can decrease the objective value. Therefore, the change after each swap is non-negative and sum of them is also non-negative. We can conclude that any upper bound on the sum of the changes after those swaps is also non-negative. The last step is to find the result of adding up all the upper bounds for the swaps: the expression inside the first summation appears exactly once for each $j \in V$ because $i^* = \sigma_{S^*}(j)$ is the second element of exactly one swap. The expression inside the second summation can appear in at most 2 swaps for each $j \in V$ because $i = \sigma_S(j)$ is the first element of at most two swaps. Therefore, the result of the sum is at most:

$$\sum_{j\in V}(d(\sigma_{S^*}(j),j) - d(\sigma_S(j),j)) + 2\sum_{j\in V} 2d(j,\sigma_{S^*}(j)) = F(S^*) - F(S) + 4F(S^*) = 5F(S^*) - F(S)$$

Which is a non-negative value:

$$5F(S^*) - F(S) \geq 0 \Rightarrow F(S) \leq 5F(S^*)$$

□

References

[Chr76] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[WS11] David P Williamson and David B Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.