COMPSCI 590.1: Graph Algorithms March 24, 2015

Lecture # 19

Lecturer: Debmalya Panigrahi *Scribe: Xiangyu Wang*

1 Overview

In the previous lecture, we covered a series of online/offline edge-weighted Steiner tree/forest problems. This lecture extends the discussion to the node-weighted scope. In particular, we will study the node-weighted Steiner tree/forest problem and introduce an offline $O(\log n)-$approximation polynomial-time algorithm[KR95]. It is well known there is no polynormial-time algorithm that achieves $o(\log n)-$approximation [LY94], so the greedy algorithm provided in this lecture is optimal up to some constant factor.

2 Set covering problem

Before proceeding to the main topic, we first look at some preliminary results for set covering problem. Set covering problem is closely related to the node-weighted Steiner tree problem. The concrete setup is described as follows.

Problem 1 (Set covering)**.** *Assume U is the universe of elements and $S \subset 2^U$ is the candidate set containing all covering sets, where $|U| = n, |S| = m$. Each element in S is then assigned a cost, which defines a mapping $c(\cdot) : S \to \mathscr{R}_+$. The goal is to find the minimum cost set $T \subseteq S$ such that*

$$\bigcup_{A \in T} A = U$$

Lund and Yannakakis show the hardness of the set covering problem via the following result[LY94].

Theorem 1. *Set covering problem cannot be approximated with ratio $c \log n$ for any $c < 1/4$ unless NP is contained in DTIME($n^{poly \log n}$).*

Therefore, the greedy algorithm introduced in this lecture is optimal up to some constant factor in terms of approximation ratio. The basic idea of the algorithm is to greedily select the optimal set which covers the most of the elements that remain uncovered at the cheapest expense, i.e., it minimize the cost-benefit ratio. The full algorithm is described in Algorithm 1.

Notice that for each iteration, one just need to enumerate all possible sets and pick the one that minimizes the cost-benefit ratio ($O(m)$) and the number of iterations will not exceed n, indicating the algorithm terminates in polynomial time. In addition, we have the following result.

Theorem 2. *Algorithm 1 achieves a performance ratio of $1 + \log n$.*

Proof. Assume for step i there are ϕ_{i-1} elements remaining uncovered, h_i is the number of elements covered at step i and the associate cost is C_i. So we have

$$\phi_i = \phi_{i-1} - h_i, \quad i = 1, 2, \cdots, p \tag{1}$$

Algorithm 1 Algorithm 1

Require: $U, S, c(\cdot)$, $T = \emptyset, Tu = \emptyset$

▷ T is used to store selected sets and Tu stores elements that have already been covered

while $Tu \not\supseteq U$ **do**

  $Add = \emptyset$, $InvRatio = 0$

  for $A \in S \setminus T$ **do**

    $CurrentInvRatio = card(A \setminus Tu)/c(A)$ ▷ $card(A \setminus Tu)$ is the benefit

    if $CurrentInvRatio > InvRatio$ **then**

      $Add = A$, $InvRatio = CurrentInvRatio$

    end if

  end for

  $T = T \cup \{Add\}$, $Tu = Tu \cup Add$

end while

return T

It is clear that $\phi_0 = n$, $\phi_{p-1} \geq 1$ and $\phi_p = 0$. The benefit of a covering set is defined as

$$h(A) = card(A \setminus Tu),$$

where $Tu \subseteq U$ is the set of elements that are already covered in previous steps. Algorithm 1 attempts to pick a covering set satisfying that

$$A_{pick} = arg\min_{A \in S} \frac{c(A)}{h(A)}.$$

For cost analysis, we use $\{X_1, X_2, \cdots, X_t\}$ to denote the optimal solution and their cost is denoted by OPT. Notice that for the optimal solution we have

$$\sum_{i=1}^{t} h(X_i) \geq h(X_1 \cup X_2 \cup \cdots \cup X_t) = \phi_{i-1}, \qquad \sum_{i=1}^{t} c(X_i) = OPT,$$

which implies

$$\frac{C_i}{h_i} = \frac{c(A_{pick})}{h(A_{pikc})} \leq \min_{i=1}^{t} \frac{c(X_i)}{h(X_i)} \leq \frac{OPT}{\phi_{i-1}}. \tag{2}$$

Therefore, combining with (1) we have

$$\phi_i \leq \phi_{i-1}\left(1 - \frac{C_i}{OPT}\right). \tag{3}$$

Applying the above result to $i = 1, 2, \cdots, p-1$ we have

$$\phi_{p-1} \leq \phi_0 \prod_{i=1}^{p-1}\left(1 - \frac{C_i}{OPT}\right).$$

Now taking logarithm and using the inequality that $log(1+x) < x$ we have

$$\sum_{i=1}^{p-1} C_i < \log\left(\frac{\phi_0}{\phi_{p-1}}\right) \cdot OPT \leq (\log n) OPT.$$

For step p, since $h_p = \phi_{p-1}$, immediately from (2) we have $C_p \leq OPT$, so together we have

$$\sum_{i=1}^{p} C_i \leq (\log n + 1)OPT.$$

□

Remark 1. *The proof technique follows that used for Steiner tree problem in [KR95] and is therefore slightly different from that given in the lecture. In later section, the definition of benefit will also be sligtly different, but we will see it makes the analysis of benefit slightly easier.*

3 Node-weighted Steiner tree

We now turn our attention to Steiner tree problem. We first define the problem.

Problem 2 (Node-weighted Steiner tree)**.** *Given a graph $G = (V,E)$, the terminal set R with $|R| = n$ and weights on the nodes $c(\cdot) : V \to \mathscr{R}_+$, find the minimum cost subgraph $H \subseteq G$ connecting all nodes in R.*

As mentioned before, node-weighted Steiner tree problem is closely related to the set covering problem. In particular, we have the following result[KR95] that shows its hardness. The proof relies on the reduction of a set covering problem to a node-weighted Steiner tree problem.

Lemma 3. *Node-weighted Steiner tree problem cannot be approximated with ratio $c\log n$ for any $c < 1/4$ unless NP is contained in DTIME($n^{poly\ \log n}$).*

Proof. We reduce a set covering problem to a node-weighted Steiner tree problem. Given a set covering problem with n elements and m covering sets, we can construct a graph as follows. The graph contains $n+m$ nodes each corresponds to either an element or a covering set. If one element is covered by a covering set, we put an edge between the corresponding two nodes. All set nodes are pair-wisely connected. The costs of set nodes equal to their original value and the costs of element nodes are all zero. Finally, the terminal set is the collection of all element nodes. □

The above lemma illustrates the difficulty of the node-weighted Steiner tree problem, as we cannot expect a polynomial algorithm that achieves an approximation ratio of $o(\log n)$. Our goal of this lecture is to modify Algorithm 1 and apply to the Steiner tree problem. The key idea for the modification is to find analogy between Steiner tree and set covering problem. Clearly, the elements in set covering problem correspond to the terminals in Steiner tree. To find the counterpart for the covering set in Steiner tree problem, we introduce the following concept.

Definition 1 (Spider)**.** *A **spider** is a tree with at most one node of degree greater than 2. A **center** of a spider is a node from which there are edge-disjoint paths to leaves of the spider. A **foot** of a spider is a leaf, or, if it has at least three leaves, the center as well. A **nontrivial** spider is a spider with at least two leaves. A simple example is illustrated in Fig 1.*

We hope spider would perform similar as the covering set in the sense that it connects certain terminals. In set covering problem, we are given a collection of covering sets. To find a collection of spiders that can be used to connect terminals in Steiner tree problem, we use the following decomposition theorem[KR95].

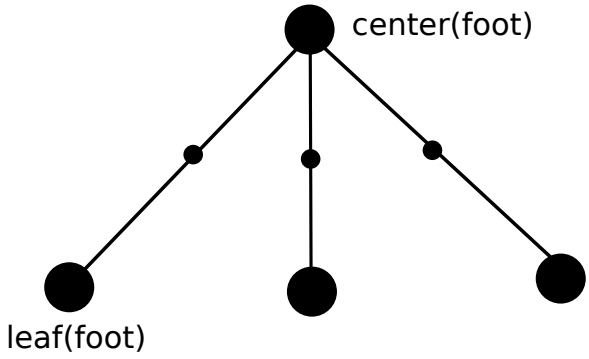

Figure 1: Illustration of a spider with three leaves. Notice that when a spider has at least three leaves, its center is unique.

Theorem 4 (Spider decomposition)**.** *Let G be a connected graph, and let M be a subset of its node with $|M| \geq 2$, then there exists a set of node-disjoint nontrivial spiders $H = \{A_1, A_2, \cdots, A_t\}$ where $A_i \subseteq G$ such that the union of feet of A_i contains M. In particular, if the Steiner tree in G contains M only as leaves, then the union of leaves of A_i also contains M. We refer to H as the spider decomposition of M in G.*

Remark 2. *The first half of Theorem 4 is the original statement in [KR95]. Notice that it is possible that some terminal appears as the internal node in the Steiner tree, so the authors define both the leaves and the center of a spider as feet, and use feet to cover all terminals, which solves this issue. In the lecture, we use a trick to modify the original graph, by adding an associated node (with zero cost) to all terminals and making the newly added nodes as our targeted terminals. This change renders the final Steiner tree contains all terminals as its leaves. Consequently, we can just use the leaves of spiders to cover all terminals, which corresponds to the second half of Theorem 4.*

We illustrate the construction of a spider decomposition via Fig 2 and 3. The proof follows exactly the same way.

Proof. We prove the construction by induction. Consider any Steiner tree in G connect nodes in M. We first pick a node v of the maximum depth possible such that the subtree rooted at v contains at least two nodes of M. Because v is of the maximum depth possible, the paths from v to the nodes in M contained in the subtree must be node-disjoint. This gives a nontrivial spider.

Now we delete the subtree rooted at v as well as v itself. If the tree still contains more than two points in M, we can find the spider decomposition to the remaining part by the induction assumption. See Fig 2. If the tree contains only one node in M, we add the path from that node to v, which is still a valid spider and we are done. See Fig 3. □

The spiders discovered from the decomposition operation will be used essentially as the covering sets when constructing the greedy algorithm for Steiner tree. The basic idea of the algorithm is that at each step we will be given a set of connected components (initially just terminals). The algorithm attempts to select the spider that connects the most of the components but at the cheapest cost, i.e., it minimizes the cost-benefit ratio (benefit to be defined later). We then contract the nodes connected by this spider to form a new connected component and enter the next step with the remaining connected components. This procedure is repeated until all terminals are connected. The concrete ideal algorithm is described in Algorithm 2.

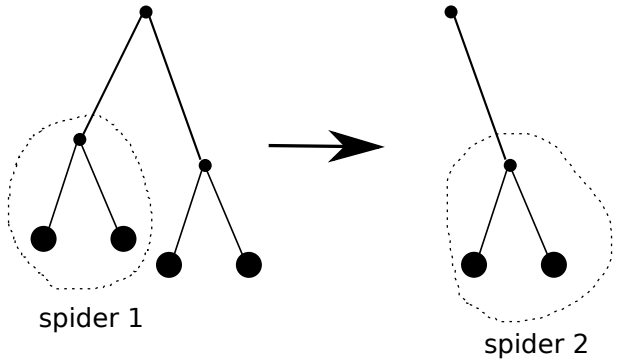

Figure 2: Illustration of spider decomposition construction. Case 1: the remaining nodes are at least two when the first spider is pruned.

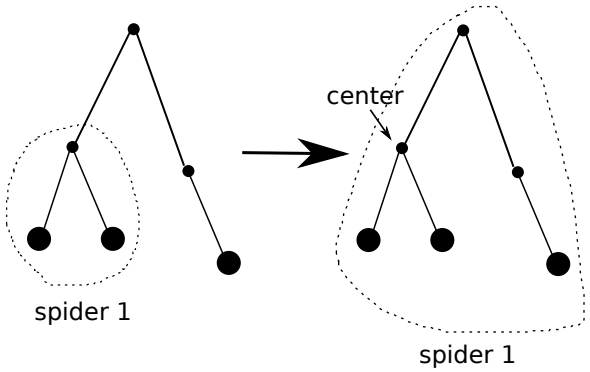

Figure 3: Illustration of spider decomposition construction. Case 1: the remaining nodes are exactly one when the first spider is pruned.

Algorithm 2 Ideal algorithm using spider

Require: $G, R, c(\cdot)$, $T = R$ ▷ T stores the connected components
  while $|T| \geq 1$ **do**
    $Cont = \emptyset$, $InvRatio = 0$
    for all H that forms the spider decomposition of T in G **do**
      for all $A \in H$ **do**
        $CurrentInvRatio = card(\{x \in T \mid x \text{ connected by } A\})/c(A)$
          ▷ The benefit is defined as the number of components connected by the spider
        if $CurrentInvRatio > InvRatio$ **then**
          $Cont = A$, $InvRatio = CurrentInvRatio$
        end if
      end for
    end for
    Contract *Cont* as a supernode (connected component) in G. Add *Cont* as a node to T and delete all elements of *Cont* contained in T.
  end while
  return T

As shown in Algorithm 2, the definition of benefit slightly differs from that used in the lecture. In the lecture, we define the benefit of a spider $h'(A)$ as the reduction of number of connected components (or supernodes) in T, while here the benefit $h(A)$ is defined as the number of connected components (or supernodes) in T merged by a spider. The two quantities differ by exactly one: $h'(A) = h(A) - 1$. We will see the advantage of using $h(A)$ in the analysis of approximation ratio.

Theorem 5. *Algorithm 2 achieves a performance ratio of* $2\log n$.

Proof. This proof is essentially the same as the one for Theorem 4. Assume for step i there are ϕ_{i-1} connected components (supernodes) remaining in T, h_i is the number of nodes merged at step i and C_i is the associate cost. So we have

$$\phi_i = \phi_{i-1} - (h_i - 1), \quad i = 1, 2, \cdots, p \tag{4}$$

It is worth noting that (4) is different from (1). In set covering problem, the reduction of the uncovered elements equals to the number of elements covered at the same step. In the Steiner tree problem, however, the reduction of the connected components equals to the number of merged components minus one. This difference also results in a different approximation ratio for the two problems.

We know that $\phi_0 = n$ and $\phi_p = 1$. Let H be a collection of all possible spiders for the current T in the current G. The benefit of a spider $S \in H$ is defined as

$$h(A) = card\Big(\{x \in T \mid x \text{ connected by } A\}\Big),$$

where T is collection of all connected components (supernodes) at the current step. Algorithm 2 attempts to pick a spider satisfying that

$$A_{pick} = arg\min_{A \in H} \frac{c(A)}{h(A)}.$$

For cost analysis, assume $\{X_1, X_2, \cdots, X_t\}$ is the set of spiders induced by the optimal Steiner tree and the cost of the optimal Steiner tree is denoted by OPT. Because spiders are constructed to be node-disjoint, thus we have

$$\sum_{i=1}^{t} h(X_i) = \phi_{i-1},$$

and the cost of this set of spiders cannot exceed the optimal cost of the entire Steiner tree, i.e.,

$$\sum_{i=1}^{t} c(X_i) \le OPT,$$

which implies

$$\frac{C_i}{h_i} = \frac{c(A_{pick})}{h(A_{pikc})} \le \min_{i=1}^{t} \frac{c(X_i)}{h(X_i)} \le \frac{OPT}{\phi_{i-1}}. \tag{5}$$

Therefore, combining with (4) we have

$$\phi_i \le \phi_{i-1} - (h_i - 1) \le \phi_{i-1} - h_i/2 \le \left(1 - \frac{C_i}{2 \cdot OPT}\right), \tag{6}$$

where the first step is due to the fact that $h_i \leq 2(h_i - 1)$ when $h_i \geq 2$ (For each step, at least two connected components will be merged). Applying the above result to $i = 1, 2, \cdots, p$ we have

$$\phi_p \leq \phi_0 \prod_{i=1}^{p} \left(1 - \frac{C_i}{2 \cdot OPT}\right).$$

Now taking logarithm and using the inequality that $log(1+x) < x$ we have

$$\sum_{i=1}^{p} C_i < 2\log\left(\frac{\phi_0}{\phi_p}\right) \cdot OPT \leq (2\log n) OPT.$$

This completes the proof. □

There is still one remaining issue with the ideal algorithm 2. The algorithm requires to compute the optimal spider at each iteration while there might be exponential many numbers of spiders. One possible solution is to search for some other structure that merges connected components at the minimum cost, and can be accomplished in polynomial time. When this procedure is carried out properly and that structure is carefully chosen, the property of the greedy algorithm 2 might still be pertained. Based on this idea, we modify Algorithm 2 to obtain an algorithm that runs in polynomial time.

Algorithm 3 Klein-Ravi algorithm

Require: $G, R, c(\cdot), T = R$
  while $|T| \geq 1$ **do**
    $Cont = \emptyset$, $InvRatio = 0$ ▷ $Cont$ stores the optimal structure that merges elements in T
    for all $v \in G \setminus T$ **do** ▷ We enumerate all possible centers of $Cont$
      $ContV = \emptyset$ ▷ $ContV$ stores the optimal structure that is centered at v
      for $i = 1$ **To** $|T|$ **do**
        $d_i = c(p(v, x_i))$, where $x_i \in T$ ▷ Compute the shortest path from v to each component
        ▷ $p(v,x)$ stands for the shortest path (minimum cost path) from v to x.
      end for
      for $d \in sort(\{d_1, d_2, \cdots, d_{|T|}\}, ascending)$ **do**
        $ContV = ContV \cup p(v, x_d)$ ▷ x_d stands for the element in T that is associated with d
        ▷ Adding the sorted paths to $ContV$ in an ascending order
        $CurrentInvRatio = card(\{x \in T \mid x \text{ connected by } ContV\})/c(ContV)$
        if $CurrentInvRatio > InvRatio$ **then**
          $Cont = ContV$, $InvRatio = CurrentInvRatio$
        end if
      end for
    end for
    Contract $Cont$ as a supernode in G. Add $Cont$ as a node to T and delete all elements of $Cont$ that are contained in T.
  end while
  return T

Algorithm 3 starts with all terminals. At each iteration, it searches the optimal structure $Cont$ that will be used to connect elements in T. Now for any given node as the center of $Cont$, it computes the shortest path from v to all elements belonging to T, incorporate them in an ascending order and terminates

at the one gives the minimum cost-benefit ratio. We then enumerate over all nodes in $G \setminus T$ and pick the optimal *Cont*. Assume the optimal spider that Algorithm 2 shall pick at the current iteration is centered at v. According to the definition, this spider contains node-disjoint paths from v to elements in T. Therefore, the above procedure selects either this optimal spider or some other structure that possesses an even smaller cost-benefit ratio. So Theorem 5 still holds for Algorithm 3.

4 The node-weighted Steiner forest problem

We extend the result from the previous section to the Steiner forest problem.

Problem 3 (Node-weighted Steiner forest)**.** *Given a graph $G = (V,E)$, the terminal set $R = \{(s_i,t_i), i = 1,2,\cdots,n\}$ and weights on the nodes $c(\cdot) : V \to \mathscr{R}_+$, find the minimum cost subgraph $H \subseteq G$ connecting all pairs in R.*

Algorithm 3 can be easily modified to fit in the node-weighted Steiner Forest problem. Notice that for Steiner forest problem, connecting two connected components might not contribute to our goal, because these two components might not separate any pairs in R. As a result, we need to classify the connected components and assign benefits only to those having contributions. Thus, we introduce the following definition.

Definition 2 (Deficient component)**.** *A **deficient component** is a connected subgraph in G that contains exactly one node from (s_i,t_i) for some i.*

We can then modify Algorithm 2 and 3 accordingly such that T only contains deficient components. Or we can update the benefit function $h(A)$ for a spider A as

$$h(A) = card(\{x \in T \mid x \text{ connected by } A \text{ and } x \text{ is a deficient component}\})$$

and modify the termination condition to be

$$|\{x \in T \mid x \text{ is deficient}\}| == 0.$$

5 Summary

In this lecture, we study the node-weighted Steiner tree problem and analyze the Klein-Ravi algorithm which runs in polynomial time and achieves an $O(\log n)-$approximation ratio. The construction technique of spider decomposition plays a key role in analyzing the performance of the greedy algorithm, which seems to be a generally important technique for all tree-related greedy algorithms.

References

[KR95] Philip Klein and R Ravi. A nearly best-possible approximation algorithm for node-weighted steiner trees. *Journal of Algorithms*, 19(1):104–115, 1995.

[LY94] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM (JACM)*, 41(5):960–981, 1994.