A CONSTANT FACTOR APPROXIMATION FOR MINIMUM λ-EDGE-CONNECTED K-SUBGRAPH WITH METRIC COSTS *

MOHAMMADALI SAFARI † AND MOHAMMAD R. SALAVATIPOUR‡

Abstract. In the (k, λ)-subgraph problem, we are given an undirected graph $G = (V, E)$ with edge costs and two positive integers k and λ and the goal is to find a minimum cost simple λ-edge-connected subgraph of G with at least k nodes. This generalizes several classical problems, such as the minimum cost k-Spanning Tree problem or k-MST (which is a $(k, 1)$-subgraph), and minimum cost λ-edge-connected spanning subgraph (which is a $(|V(G)|, \lambda)$-subgraph). The only previously known results on this problem [12, 4] show that the $(k, 2)$-subgraph problem has an $O(\log^2 n)$-approximation (even for 2-node-connectivity) and that the (k, λ)-subgraph problem in general is almost as hard as the densest k-subgraph problem [12]. In this paper we show that if the edge costs are metric (i.e. satisfy triangle inequality), like in the k-MST problem, then there is an $O(1)$-approximation algorithm for (k, λ)-subgraph problem. This essentially generalizes the k-MST constant factor approximability to higher connectivity.

1. Introduction. Network design is a central topic in combinatorial optimization, approximation algorithms, and operations research. A fundamental problem in network design is to find a minimum cost subgraph satisfying some given connectivity requirements between vertices. Here by a network we mean an undirected graph together with non-negative costs on the edges. For example, with a connectivity requirement $\lambda = 1$ between all the vertices, we have the classical minimum spanning tree problem. For larger values of λ, i.e. finding minimum cost λ-edge-connected spanning subgraphs, the problem is APX-hard. These are special cases of the more general edge-connectivity survivable network design problem (SNDP) or Steiner network, in which we have an edge-connectivity requirement of r_{uv} between every pair u, v of vertices. Even for this general setting there is a 2-approximation algorithm by Jain [11].

A major line of research in this area has focused on problems with connectivity requirements where one has another parameter k, and the goal is to find a subgraph satisfying the connectivity requirements with a lower bound k on the total number of vertices. The most well-studied problem in this class is the minimum k-spanning tree problem, a.k.a. k-MST. In this problem, we have to find a minimum cost connected subgraph spanning at least k-vertices. The approximation factor for this problem was improved from $\sqrt{k}$ by Ravi et al. [14], to $O(\log^2 k)$ by Awerbuch et al. [1], to $O(\log n)$ by Rajagopalan and Vazirani [13], then to a constant by Blum et al. [3], and recently to 2 by Garg [10]. The algorithm of [10] can be used to obtain a constant approximation for the slightly more general setting in which we have a set of nodes T, called terminals, and the goal is to find a minimum cost connected subgraph containing at least k terminals. This is known as the k-Steiner tree problem. The problem of k-TSP, in which one has to find a minimum cost TSP tour containing at least k nodes, can be approximated using very similar technique. We note that in all these problems, the input graph is assumed to be complete and the edge cost function is metric, i.e. satisfies triangle inequality. Most of these problems are motivated from their applications in vehicle routing or profit maximization with respect to a given fixed budget. For example, suppose that we have a battery operated robot and the goal is to find the minimum battery charge required to travel a sequence of at least k nodes in a given graph such that the total length of the tour can be travelled in a single battery charge. See [2, 5] for similar problems.

Recently, Lau et al. [12] considered a very natural common generalization of both the k-MST problem and the minimum cost λ-edge-connected spanning subgraph problem, which they called the (k, λ)-subgraph problem. In this problem, we are given a graph $G = (V, E)$ with a (not necessarily metric) cost function $c : E \to \mathbb{R}^+$ on the edges, two positive integers k and λ; the goal is to find a minimum cost simple λ-edge-connected subgraph of G with at least k vertices. We should emphasize that the solution must be simple, that is, we are not allowed to pick an edge more than once. Otherwise, a 4-approximate solution can be computed by taking a 2-approximate k-MST solution T, and then taking λ copies of T since $\lambda/2$ times the cost of a k-MST is a lower bound for the cost of an optimum solution (details given in the next section). It is easy to observe that the (k, λ)-subgraph problem contains, as special cases, the minimum cost λ-edge-connected spanning subgraph problem (it becomes the $(|V(G)|, \lambda)$-subgraph problem), and the k-MST problem (which becomes the $(k, 1)$-subgraph problem). Lau et al. [12] presented an $O(\log^2 n)$-approximation

* A preliminary version of this paper appeared in the Proceedings of the 11th Intl. Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX) 2008.

† Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. email: msafari@sharif.ac.ir. Most of this work was done while the author was a postdoctoral fellow at the Department of Computing Science of the University of Alberta and was supported by Alberta Ingenuity.

‡ Department of Computing Science, University of Alberta, Edmonton, Alberta T6G 2E8, Canada. email: mreza@cs.ualberta.ca. The author was supported by NSERC and an Alberta Ingenuity New Faculty award.

for $(k,2)$-subgraph and showed that for arbitrary values of λ, (k,λ)-subgraph is almost as hard as the k-densest subgraph problem. In the k-densest subgraph problem, one has to find a subgraph with k nodes in a given graph G that has the largest number of edges. Despite considerable attempts, the best known approximation algorithm for this problem has ratio $O(n^{\frac{1}{4}})$ by Bhaskara et al. [8]. Chekuri and Korula [4] have recently (and independently of [12]) shown that an algorithm similar to the one in [12] yields an $O(\log^2 n)$-approximation for the $(k,2)$-subgraph problem even if we seek a 2-node-connected subgraph.

In light of the result of [12] on the hardness of (k,λ)-subgraph for arbitrary values of λ and general costs, it is natural to try to obtain good approximation algorithms for the class of graphs where the edge cost function is metric. Remember that the constant factor approximation algorithms for k-MST and k-TSP are on graphs with metric cost function. The main result of this paper is the following theorem:

THEOREM 1.1. *Given a (complete) graph G with metric costs on the edges and two positive integers k,λ, there is an $O(1)$-approximation algorithm for finding a (k,λ)-subgraph in G.*

Our algorithm is combinatorial and uses ideas from Cheriyan and Vetta [6] for metric-cost subset node-connectivity problem as well as the algorithm for k-Steiner tree [7, 10, 14]. The constant factor we obtain is relatively large (namely 450), however, most of our efforts have been made to show that the problem has a constant factor approximation rather than trying to obtain the best possible ratio.

The rest of the paper is organized as follows. We start by some definitions and preliminary bounds used throughout the paper. For ease of exposition, we first present an algorithm that finds a λ-edge-connected subgraph with at least $k-\lambda/7$ nodes whose cost is $O(\text{OPT})$. In Section 3 we show how to extend this solution to a feasible solution to the (k,λ)-subgraph problem while keeping the total cost still bounded by $O(\text{OPT})$. We finish the paper with some concluding remarks.

2. Preliminaries. As mentioned earlier, we assume we are given a graph $G=(V,E)$, with a cost function $c: E \to \mathbb{R}^+$ on the edges that satisfies triangle inequality, and two positive integers k and λ. We assume that G is a complete graph as this is a natural assumption when the cost function is metric. For every subgraph $F \subseteq G$, we use $c(F)$ to denote the total cost of the edges in F and $|F|$ denotes the number of vertices in F. Throughout, $G^* \subseteq G$ denotes the optimum solution and $\text{OPT} = c(G^*)$ denotes the optimum solution cost. We will use two lower bounds on OPT in the analysis of our algorithm. These lower bounds were used earlier in [6] for the problem of minimum cost subset k-node-connected subgraph. The first lower bound comes from the cost of a minimum spanning tree of G^*, which we call T^*. Considering the cut-constraint IP formulation of MST, it is easy to see that $\frac{\lambda}{2}\sum_{e\in T^*} c_e \le \text{OPT}$. To see this observe that by the well-known results of Nash-Williams and Tutte (see [15]) any λ-edge-connected graph has $\lambda/2$ edge-disjoint spanning trees. So $\lambda/2$ times the cost of MST of G^* is no more than OPT. The second lower bound comes from the minimum cost subgraph that has minimum degree at least λ. Note that in a λ-edge-connected subgraph, every vertex has degree at least λ. For any λ-edge-connected subgraph $F \subseteq G$ and any vertex $u \in F$ let $S_u(F)$ be the set of λ nearest neighbors of u in F and $s_u(F)$ be $\sum_{v\in S_u(F)} c_{uv}$. Clearly, for any λ-edge-connected subgraph $F \subseteq G$ and any vertex $u \in F$ we have $s_u(G) \le s_u(F)$. We often use S_u and s_u instead of $S_u(G)$ and $s_u(G)$, respectively, unless the graph is different from G. It is easy to see that $\frac{1}{2}\sum_{u\in G^*} s_u \le \frac{1}{2}\sum_{u\in G^*} s_u(G^*) \le \text{OPT}$. Thus, if T^* is a minimum spanning tree of G^*, then we obtain the following two lower bounds for OPT: (i) $\frac{1}{2}\sum_{u\in T^*} s_u \le \text{OPT}$, and (ii) $\frac{\lambda}{2}\sum_{e\in T^*} c_e \le \text{OPT}$, and in particular:

$$\frac{1}{2}\sum_{u\in T^*} s_u + \frac{\lambda}{2}\sum_{e\in T^*} c_e \le 2\text{OPT}. \tag{2.1}$$

In our algorithm we will use these two lower bounds frequently, often without referring to them.

We present an algorithm for a modified version of the problem in which along with G, k, and λ, we are also given a vertex $r \in G$ as the root which belongs to the optimum solution G^* and among all the vertices in G^* it has the smallest value s_u. Clearly, if we can solve this rooted version, then we can try every vertex as the root and return the minimum solution among all as the final answer.

Ravi et al. [14] showed that any α-approximation for k-MST implies a 2α-approximation for k-Steiner tree. Therefore, together with Garg's algorithm [10], we have a 4-approximation for k-Steiner tree. In fact, we can have a 4-approximation algorithm for the rooted version of the problem, in which a specific vertex r is given as the root and the goal is to find a minimum cost Steiner tree rooted at r containing at least k terminals. Our algorithm will use the best known approximation algorithm for finding a minimum cost rooted k-Steiner tree problem; let us denote the

approximation ratio of this algorithm by ρ (by the argument above, we know that $\rho \leq 4$). We denote this approximation algorithm by ST-Algorithm.

Although we use similar lower bounds for the optimum as by Cheriyan and Vetta [6], we also use several new ideas. Roughly speaking, the algorithm of [6] for the subset k-node-connected subgraph builds a "cheap" cycle containing all the terminals, then makes k other cycles parallel to that (called tracks) by taking the k nearest neighbors of terminals. They make a "thick" cycle by making connection between different tracks to ensure high connectivity. One major difficulty in our case is that we do not have a given set of nodes that need to be in the final solution. To overcome this, we use an approximate algorithm for the k-MST problem. Informally, we start from such a tree and then make it "thick" by adding some of the nodes in S_u for some vertices u of the tree to obtain the required connectivity. However, it appears that we cannot obtain the required connectivity while bounding the cost. To overcome this difficulty our main algorithm works in two phases. In the first phase we obtain a λ-edge-connected subgraph with $k - O(\lambda)$ nodes whose cost is within a constant factor of optimum (presented in Section 3). Then in the second phase using a different algorithm we show how to augment this graph to obtain a (k, λ)-graph while keeping the cost $O(\text{OPT})$ (presented in Section 4).

3. Obtaining a low cost $(k - \lambda/7, \lambda)$-subgraph. Observe that to have λ-edge-connectivity while being a simple graph, we must have $k \geq \lambda + 1$. We start by presenting an algorithm that returns a λ-edge-connected subgraph (containing root r) that has at least $k - \lambda/7$ nodes and whose cost is within constant factor of OPT. Our algorithm is influenced by [6] for the minimum cost subset k-node-connected subgraph problem.

3.1. Overview of the Algorithm. The main idea of the algorithm is as follows. We first find a path P_1 over a set $V_1 \subseteq V$ of size k (containing root r) having the following properties:
(R1) $c(P_1) = \sum_{e \in P_1} c(e)$ is $O(\text{OPT}/\lambda)$, and
(R2) $\sum_{v \in P_1} s_v = O(\text{OPT})$.

Note that these two are essentially the same as the bound given in Equation (2.1). If we were to make a clique out of the nodes in $S_v \cup \{v\}$ for each $v \in V_1$ and then for each edge $uv \in P_1$ add an arbitrary matching between the nodes in S_u and S_v we would obtain a λ-edge-connected subgraph. The trouble is, we cannot bound the cost of edges of these cliques. However, if we could prove a property stronger than (R2), such as $\sum_{v \in P_1} s_v = O(\text{OPT}/\lambda)$, then we could bound the total cost of the edges of such cliques by $O(\text{OPT})$. We cannot prove this stronger property but we can show one can select a subset of nodes of P_1 that satisfies a property close to this stronger one. By carefully selecting a subset $V_2 \subseteq V_1$ and another $I \subseteq V_1 - V_2$, we shortcut over the nodes not in V_2 to obtain another path P_2 (from P_1) with the following stronger properties:
(R2-new) $\lambda \sum_{v \in V_2} s_v + \sum_{v \in I} s_v = O(\text{OPT})$,
(R3) $|V_2 \cup I \cup \left(\bigcup_{v \in V_2} S_v\right)| \geq k - \lambda/7$.

For now suppose we have obtained path P_1 and then P_2 as well as set I which satisfy properties mentioned above. Our final graph will contain all the nodes $V_2 \cup I \cup \left(\bigcup_{v \in V_2} S_v\right)$. By (R3) there are at least $k - \lambda/7$ nodes in this solution. For each $v \in V_2$ we form a $(\lambda + 1)$-clique on $S_v \cup \{v\}$ to get λ-edge connectivity among the vertices in $S_v \cup \{v\}$. Each node $u \in I$ will be assigned to a node $v \in V_2$ and becomes connected to all the nodes of the clique on $S_v \cup \{v\}$. Using property (R2-new), we will show that the total cost of all these edges is at most $O(\text{OPT})$. Next, we need to establish λ-edge connectivity among these cliques. For that, for every edge $uv \in P_2$, we add (up to) λ edges between the two $(\lambda + 1)$-cliques corresponding to u and v; the number of edges added will be less than λ if S_u and S_v have nodes in common. Roughly speaking, the total cost of the (at most) λ edges between the cliques will be $O(\lambda)$ times the cost of the edges in P_2 plus the sum of s_v values of the nodes of P_2; since using triangle inequality $c(P_2) \leq c(P_1)$, using (R1) and (R2-new) it follows that the cost of matching edges will be no more than $O(\text{OPT})$. Therefore we will have a solution within constant factor of the optimum that has, according to property (R3), $k - \lambda/7$ vertices.

Many details are skipped over in this overview and are explained in the next subsection.

3.2. Details of the Algorithm. We build a path P_1 in G and then another path P_2 in two phases:

3.2.1. Phase 1: Path P_1 with exactly k vertices satisfying properties (R1) and (R2).. Create a new graph $G'(V \cup V', E')$ from G by hanging a new (dummy) vertex u' (in V') from each vertex $u \in G$ and let $E' = E \cup \{uu' | u \in V\}$. Each edge $uu' \in E'$ has weight s_u. For every other edge in G' (that also exists in G) we multiply its weight by λ. Next we compute an approximate (rooted) k-Steiner tree with terminal set $V' = \{v' | v \in V(G)\}$ and root r' (copy of r) using the ST-Algorithm. Let us call this tree T'.

CLAIM 3.1. *The cost of T' is at most 4ρOPT.*

Proof. Consider the optimum solution G^* of the (k,λ)-subgraph problem in G and let T^* be a MST of G^* (we assume that $r \in T^*$). Then $T^* \cup \{uu' \in G' | u \in T^*\}$ is clearly a Steiner tree in G' containing at least k terminals with total cost at most 4OPT (using bound (2.1) for T^*). Note that for each edge $e' \in G'$ (corresponding to edge $e \in G$): $c_{e'} = \lambda c_e$. The lemma follows by observing that the ST-Algorithm (for k-Steiner tree) has approximation ratio ρ. □

Without loss of generality, we can assume that T' has exactly k terminals, as if it has more, then we can safely delete them. Let $T_0 \subseteq G$ be the tree obtained from $T' \subseteq G'$ by deleting the dummy nodes (i.e. the nodes in V') and V_0 be the vertex set of T_0. Note that by Claim 3.1:

$$\lambda \sum_{e \in T_0} c_e + \sum_{u : u' \in T'} s_u \leq 4\rho\text{OPT}. \tag{3.1}$$

Thus, the cost of edges of T_0 is at most $4\rho\text{OPT}/\lambda = O(\text{OPT}/\lambda)$. We should also point out that V_0 might have some vertices $v \in V$ (and therefore $v \in T'$) but $v' \notin T'$. We obtain a path $P_1 = (V_1, E_1) \subseteq G$ from T_0 with the following properties: (i) $V_1 \subseteq V_0$, (ii) $c(P_1) \leq 2c(T_0)$, and (iii) for every vertex $v \in V_1$, the corresponding vertex $v' \in G'$ belongs to T', thus $|V_1| = k$. This will ensure that P_1 has exactly k nodes and satisfies properties (R1) and (R2). To do this, we duplicate every edge of T_0 and do an Eulerian walk of T_0; now shortcut over every vertex $v \in T_0$ with $v' \notin T'$; also shortcut over repeated nodes. It is easy to see that we are left with a path P_1 whose cost is at most $2c(T_0)$ and every vertex $v \in P_1$ has its copy v' in T'; so it has k vertices and by Equation (3.1) satisfies both (R1) and (R2). Thus:

LEMMA 3.2. *We can compute a path $P_1 = (V_1, E_1)$ such that $V_1 \subseteq V_0$, $|V_1| = k$, and $c(P_1) \leq 2c(T_0)$; i.e. P_1 satisfies both (R1) and (R2).*

3.2.2. Phase 2: Path P_2 satisfying properties (R2-new) and (R3). Next we compute two disjoint sets of vertices $V_2 \subseteq V_1$ and $I \subseteq V_1$ and a path P_2 over V_2 (by shortcutting over nodes not in V_2). Our idea in computing V_2 is to keep $O(k/\lambda)$ vertices of P_1 whose S_u neighborhoods are mostly disjoint and their s_u values are small, in order to establish the stronger (R2-new).

Suppose that we have an ordering of the vertices of P_1, say $v_1 = r, v_2, \ldots, v_k$, such that $s_{v_2} \leq s_{v_3} \leq \ldots \leq s_{v_k}$. Note that although r has the smallest s_u value among all the vertices $u \in G^*$, it is not necessarily the case in P_1. For each $1 \leq i \leq k$, let $\mu_i = \frac{s_{v_i}}{\lambda}$. We call S_{v_i} the ball of v_i. We also define the core of S_{v_i}, denoted by B_{v_i}, to be the set of nodes in S_{v_i} with distance at most $2\mu_i$ to v_i. By a simple averaging argument, one can easily show that $|B_{v_i}| \geq \lambda/2$. We partition the nodes of P_1 into two sets of *active* and *inactive* nodes using the following procedure to cluster the balls. Initially, all the nodes of P_1 are active and we have $\mathcal{S} = \emptyset$ ($\mathcal{S}$ will contain the centers of active balls). For each $1 \leq i \leq k$, if v_i is active and there is no $v_j \in \mathcal{S}$ (with $j < i$) such that $c_{ij} \leq 4\mu_i + 2\mu_j$ then add v_i to $\mathcal{S}$ and make all the nodes in S_{v_i} inactive except v_i itself (the bound $c_{ij} \leq 4\mu_i + 2\mu_j$ is chosen to ensure that the cores of the active nodes in $\mathcal{S}$ are disjoint and is also used in the proof of a technical lemma later). Note that S_{v_i} might include some vertices not in P_1. So at the end, for every two active nodes $v_i, v_j \in \mathcal{S}$ (with $j < i$) we have $c_{ij} > 4\mu_i + 2\mu_j$ and $B_{v_i} \cap B_{v_j} = \emptyset$. Also, for every active node $v_i \in \mathcal{S}$, ball S_{v_i} only contains inactive nodes. Now for every value of $1 \leq i \leq k$ such that v_i is active but $v_i \notin \mathcal{S}$, there exists a $j < i$ such that $v_j \in \mathcal{S}$ and $c_{ij} \leq 4\mu_i + 2\mu_j$. Let j^* be the smallest such index and define $p(i) = j^*$, meaning that v_i is assigned to ball $S_{v_{j^*}}$. So each active node v_i is either the center of an active ball (and it belongs to $\mathcal{S}$) or is assigned to a ball $S_{p(i)}$ with $p(i) \in \mathcal{S}$, and all the remaining nodes (that are inside the balls with centers in $\mathcal{S}$) are inactive. Thus:

CLAIM 3.3. *The cores $B_{v_i}, v_i \in S$, are pairwise disjoint and $|B_{v_i}| > \frac{\lambda}{2}$.*

For every value of i, consider the union of active nodes v_j and their ball S_{v_j} (if $v_j \in \mathcal{S}$), for all $j \leq i$, and define this set of vertices U_i. More precisely, for each active node v_j ($j \leq i$) define $S'_{v_j} = S_{v_j} \cup \{v_j\}$ if $v_j \in \mathcal{S}$, and $S'_{v_j} = \{v_j\}$ otherwise. Then

$$U_i = \bigcup_{\text{active } v_j,\ j \leq i} S'_{v_j}. \tag{3.2}$$

Let $i^* \leq k - \lambda/7$ be the smallest index such that $|U_{i^*}| \geq k - \lambda/7$. It is easy to see from the definition of U_i and the choice of i^* that:

LEMMA 3.4. $k - \frac{\lambda}{7} \leq |U_{i^*}| \leq k + \frac{6\lambda}{7}$.

The solution of our algorithm will be a graph H on vertex set U_{i^*}. Let V_2 be the set of active nodes in $\mathcal{S}$ with index at most i^* and I be the set of active nodes not in $\mathcal{S}$ with index at most i^*. Note that

OBSERVATION 3.5. *From the definitions of $\mathcal{S}$, U_{i^*}, V_2, and I:*
(i) $V_2, I \subseteq V_1$ and they are disjoint, and
(ii) Every node $v_i \in P_1$ with $i \leq i^*$ is in U_{i^*}.

We compute a path $P_2 = (V_2, E_2)$ from P_1 by simply shortcutting over vertices of P_1 that are not in V_2. Using triangle inequality:

$$c(P_2) \leq c(P_1). \tag{3.3}$$

Also, using the definition of V_2, I, and U_{i^*}, and Lemma 3.4, property (R3) is satisfied. Soon, we will show that P_2 and I also satisfy (R2-new). We need the following technical lemma to prove this:

LEMMA 3.6. *For every $v_i \in \mathcal{S}$, with $i \geq 2$, (that is every node in $\mathcal{S}$ except the root $r = v_1$) and every node $v_j \in P_1$ with $c_{v_i v_j} \leq 2\mu_i$ such that v_j became inactive once we added v_i to $\mathcal{S}$: $\mu_i \leq 2\mu_j$.*

Proof. If $i < j$ (i.e. v_i was considered before v_j) then clearly $s_{v_i} \leq s_{v_j}$ and therefore $\mu_i \leq \mu_j$. Now suppose that $i > j$. It means that v_j was an active node but not in $\mathcal{S}$ at the time v_i was examined. This can happen only if there is $\ell < j$ with $v_\ell \in \mathcal{S}$ and

$$c_{v_j v_\ell} \leq 4\mu_j + 2\mu_\ell. \tag{3.4}$$

On the other hand, since v_i was not inactivated by v_ℓ:

$$c_{v_i v_\ell} > 4\mu_i + 2\mu_\ell. \tag{3.5}$$

Using triangle inequality:

$$c_{v_i v_\ell} \leq c_{v_i v_j} + c_{v_j v_\ell}. \tag{3.6}$$

Combining (3.4), (3.5), and (3.6), together with the assumption that $c_{v_i v_j} \leq 2\mu_i$ implies that: $4\mu_i + 2\mu_\ell < c_{v_i v_\ell} \leq c_{v_i v_j} + c_{v_j v_\ell} \leq 2\mu_i + 4\mu_j + 2\mu_\ell$; therefore $\mu_i \leq 2\mu_j$ as wanted. □

Now we are ready to prove property (R2-new) for P_2 and I. More specifically we prove the following Lemma which implies (R2-new):

LEMMA 3.7. $\lambda \sum_{v \in V_2} s_v + 7 \sum_{v \in I} s_v \leq (2 + 28\rho)\text{OPT}$.

Remark: The constant 7 in this lemma may look arbitrary at the moment, however we use this bound in the analysis of the algorithm later.

Proof.

First we show that $\lambda s_r \leq 2\text{OPT}$ and then we prove that

$$\lambda \sum_{v \in V_2, v \neq r} s_v + 7 \sum_{v \in I} s_v \leq 28\rho\text{OPT}. \tag{3.7}$$

Suppose that $v'_1, v'_2, v'_3, \ldots, v'_k$ are the vertices of the optimum solution where $v'_1 = v_1 = r$. Without loss of generality and using the assumption that $v'_1 = r$ has the smallest $s_{v'_i}$ value among all the nodes in the optimum solution we may assume that $s_r \leq s_{v'_2} \leq \ldots \leq s_{v'_k}$. From the second lower bound given for OPT in the previous section: $\sum_{1 \leq i \leq k} s_{v'_i} \leq 2\text{OPT}$. Thus, using the fact that $k \geq \lambda + 1$:

$$\lambda s_r < \sum_{1 \leq i \leq k} s_{v'_i} \leq 2\text{OPT}. \tag{3.8}$$

In order to prove (3.7) we show::

$$\lambda \sum_{v\in V_2, v\neq r} s_v + 7\sum_{v\in I} s_v \leq 7 \sum_{v\in P_1} s_v. \tag{3.9}$$

Note that this, together with Observation 3.5(i), Lemma 3.2, and Equation (3.1) implies Equation (3.7). To prove (3.9), for every vertex $v \in V_2 - \{v_1, v_{i^*}\}$ we find at least $\frac{\lambda}{2}$ distinct inactive vertices $u \in P_1 - I$ whose s_u value is at least $\frac{s_v}{2}$. This implies that the sum of s_v values of the nodes in $V_2 - \{v_1, v_{i^*}\}$ is less than that those of $P_1 - I$ by a factor of $\Omega(\lambda)$. We explain the details below.

We define a one-to-one mapping π from some of the vertices in H (i.e. a subset of U_{i^*}) to the vertices of P_1. For the technical reason that $|H|$ might be larger than k (recall that $k - \frac{\lambda}{7} \leq |H| \leq k + \frac{6\lambda}{7}$) we define our mapping from the vertices in $U_{i^*} - S_{v_{i^*}}$ instead and deal with the nodes in $S_{v_{i^*}}$ separately. More precisely, let $\tilde{H}$ be the set of nodes $U_{i^*} - S_{v_{i^*}}$ and $|\tilde{H}| = \ell$. By definition of i^* and Lemma 3.4: $\ell \leq k - \frac{\lambda}{7}$. We define our one-to-one mapping π from the vertices in $\tilde{H}$ to vertices $v_1, \ldots, v_\ell$ of P_1 in the following way in two rounds:

Round 1: For all $v \in \tilde{H} \cap P_1$ map v to itself (regardless of whether v is active or not).

Round 2: Every (inactive) $v \in \tilde{H} - P_1$ (which must belong to some S_{v_i} with $v_i \in \mathcal{S}$) is mapped to the first vertex $v_j \in P_1$ to which no other vertex of $\tilde{H}$ is mapped to previously.

First we note that the above mapping is well defined (because the number of nodes of $\tilde{H}$ is at most ℓ) and is one-to-one. Also every vertex $v_i \in P_1$ with $1 \leq i \leq i^*$ belongs to $\tilde{H}$ (by definition of $\tilde{H}$ and Observation 3.5(ii)) and is mapped to itself (in Round 1). Moreover, active vertices in $\tilde{H}$ (regardless of whether they are in V_2 or I) map to themselves; so no inactive node is mapped to an active node. Next we prove that for every inactive vertex $v \in \tilde{H}$, if it belongs to some core B_{v_i} for an active vertex $v_i \in \tilde{H} \cap \mathcal{S}$ with $2 \leq i \leq i^* - 1$ (that is an active node in $V_2 - \{v_1, v_i^*\}$), then v is mapped to an inactive vertex v_j with $s_{v_j} \geq \frac{s_{v_i}}{2}$. For this we have two cases:

Case 1 ($v \in P_1$): In this case $v \in \tilde{H} \cap P_1$ and so maps to itself (i.e. $v = v_j$) in Round 1; it suffices to show that $s_{v_j} \geq \frac{s_{v_i}}{2}$. If $j \geq i$ then clearly is $s_{v_j} \geq \frac{s_{v_i}}{2}$. If $j < i$, it means that v_j was de-activated when adding v_i to $\mathcal{S}$; in this case note that $c_{v_j v_i} \leq 2\mu_i$ (by definition of core B_{v_i}); so using Lemma 3.6 (noting that we assumed $2 \leq i \leq i^* - 1$): $\mu_i \leq 2\mu_j$ which implies $s_{v_i} \leq 2s_{v_j}$.

Case 2 ($v \notin P_1$): In this case $v \in \tilde{H} - P_1$ is mapped in Round 2. Note that since $v \in B_{v_i}$ ($2 \leq i \leq i^* - 1$) and each node $v_{i'}$ with $1 \leq i' \leq i^*$ is mapped to itself in Round 1, therefore v_j (the image of v) must come after v_{i^*}, i.e. $j \geq i^* + 1 > i$ which implies $s_{v_i} \leq 2s_{v_j}$.

Thus, we have shown:

CLAIM 3.8. *For every active node $v \in V_2 - \{v_1, v_{i^*}\}$ the images of the at least $\lambda/2$ (inactive) nodes in B_v are distinct inactive vertices $v_i \in P_1$ with $i \leq \ell \leq k - \lambda/7$, whose s_{v_i} value is at least $s_v/2$.*

Therefore:

$$\begin{aligned}
\lambda \sum_{v\in V_2 - \{v_1, v_{i^*}\}} s_v + 7\sum_{v\in I} s_v &= 4 \sum_{v\in V_2 - \{v_1, v_{i^*}\}} \frac{\lambda}{2} \cdot \frac{s_v}{2} + 7\sum_{v\in I} s_v \\
&< 7 \sum_{v\in V_2 - \{v_1, v_{i^*}\}} \frac{\lambda}{2} \cdot \frac{s_v}{2} + 7\sum_{v\in I} s_v \\
&\leq 7 \sum_{\substack{v_i \text{ is inactive,} \\ 1\leq i\leq k-\lambda/7}} s_{v_i} + 7\sum_{v\in I} s_v, \\
&\leq \sum_{1\leq i\leq k-\lambda/7} 7 s_{v_i}, \qquad (3.10)
\end{aligned}$$

where the second last inequality follows from Claim 3.8 and the last inequality follows from the fact that the nodes in I are active and have index at most $i^* \leq k - \lambda/7$. For v_{i^*}, noting that $i^* \leq k - \frac{\lambda}{7}$, we have:

$$\lambda s_{v_{i^*}} \leq \lambda s_{v_{k-\lambda/7}} \leq 7 \sum_{k-\lambda/7 < i \leq k} s_{v_i} \tag{3.11}$$

Therefore, using (3.10) and (3.11):

1 Compute path P_1 (as in Lemma 3.2) and then path P_2 and set I (as in Corollary 3.9).
2 Let H be an empty graph (H will eventually be a graph on vertex set U_{i^*}).
3 **foreach** *active node* v_i *with* $i \leq i^*$ *not already added to* H **do**
4 **if** $v_i \in \mathcal{S}$ **then**
5 Add v_i and every $u \in S_{v_i}$ to H as well as every edge uv, with $u, v \in S_{v_i} \cup \{v_i\}$
6 **else**
7 Add v_i to H and every edge uv_i, with $u \in S_{p(i)}$
8 **end**
9 **end**
10 **foreach** *edge* $v_iv_j \in P_2$ **do**
11 Add an arbitrary matching of size $\lambda - |S_{v_i} \cap S_{v_j}|$ from $S_{v_i} - S_{v_j}$ to $S_{v_j} - S_{v_i}$ in H
12 **end**
13 return H

FIG. 3.1. *Algorithm 1, which returns a $(k - \lambda/7, \lambda)$-subgraph whose cost is within a constant factor of minimum cost (k, λ)-subgraph*

$$\lambda \sum_{v \in V_2, v \neq r} s_v + 7 \sum_{v \in I} s_v \leq \sum_{1 \leq i \leq k - \lambda/7} 7 s_{v_i} + \sum_{k - \lambda/7 < i \leq k} 7 s_{v_i} \leq \sum_{1 \leq i \leq k} 7 s_{v_i}.$$

This, together with the bound proved in Lemma 3.2 for P_1, implies Equation (3.9), and completes the proof of Lemma.
□

COROLLARY 3.9. *We can compute a path $P_2 = (V_2, E_2)$ and set $I \subseteq V_1$ with $V_2 \subseteq V_1$ and $V_2 \cap I = \emptyset$, such that: $c(P_2) \leq 2c(T_0)$, and P_2 and I satisfy properties (R2-new) and (R3).*

3.2.3. Phase 3: Obtaining a $(k - \lambda/7, \lambda)$-subgraph from P_2 and I. The next steps of the algorithm would be to make a $(\lambda + 1)$-clique over $S_{v_i} \cup \{v_i\}$ for each $v_i \in V_2$ which are precisely those $v_i \in \mathcal{S}$ with $i \leq i^*$. For each active node $v_i \in I$, we connect v_i to all the λ vertices in $S_{p(i)}$. It is easy to observe that each ball S_{v_i} with $v_i \in V_2$ together with all the active nodes assigned to it will form a λ-edge-connected subgraph. The final step is to make good connectivity between these balls. For that, we look at every edge $v_iv_j \in P_2$; note that both $v_i, v_j \in \mathcal{S}$. Let $a = |S_{v_i} \cap S_{v_j}|$. We add an arbitrary matching (of size $\lambda - a$) between the $\lambda - a$ vertices in $S_{v_i} - S_{v_j}$ and $S_{v_j} - S_{v_i}$.

The full description of the algorithm, is given in Figure 3.1, while Figure 3.2 illustrates the approximate Steiner tree computed, the balls around the active nodes, and some of the edges added to make the graph λ-edge-connected.

3.3. Analysis of Algorithm. It is easy to see that H contains exactly those active nodes v_i with $i \leq i^*$ as well as all the nodes in $\bigcup_{v_i \in \mathcal{S}, i \leq i^*} S_{v_i}$; which is exactly the set U_{i^*}. Thus, by Lemma 3.4:

LEMMA 3.10. $k - \frac{\lambda}{7} \leq |H| \leq k + \frac{6\lambda}{7}$.

In the remaining of this subsection we show that H has the required connectivity while its cost is bounded by $O(\text{OPT})$.

LEMMA 3.11. *Solution H returned by Algorithm 1 is λ-edge-connected.*

Proof. For every $v \in H$, let us define the hub for v, denoted by $h(v)$, to be (i) v itself if $v \in \mathcal{S}$, (ii) $p(i)$ if $v = v_i$ is an active node but not in $\mathcal{S}$, and (iii) $v_\ell \in \mathcal{S}$ if v_ℓ is the first vertex added to $\mathcal{S}$ with $v \in S_{v_\ell}$. Observe that the set of hub nodes are precisely the nodes in $\mathcal{S}$ with index at most i^*, which is the same as the set of nodes of P_2. First it is easy to see that each v has λ-edge connectivity to its hub (for case (iii) we have made a clique out of $h(v)$ and all the vertices in its ball including v, and for case (ii) v is adjacent to λ vertices in the clique made from the ball of $h(v)$). So it is enough to show that we have λ-edge-connectivity between the hub vertices. For any two adjacent vertices $v_i, v_j \in P_2$, the matching edges added between the balls of v_i and v_j (together with possible nodes in $S_{v_i} \cap S_{v_j}$) establish λ-edge-connectivity between v_i and v_j. By transitivity, we have λ-edge-connectivity between any pair of nodes $v_i, v_j \in P_2$. □

LEMMA 3.12. *The cost of edges of H added in line 11 is at most $8\rho\text{OPT}$.*

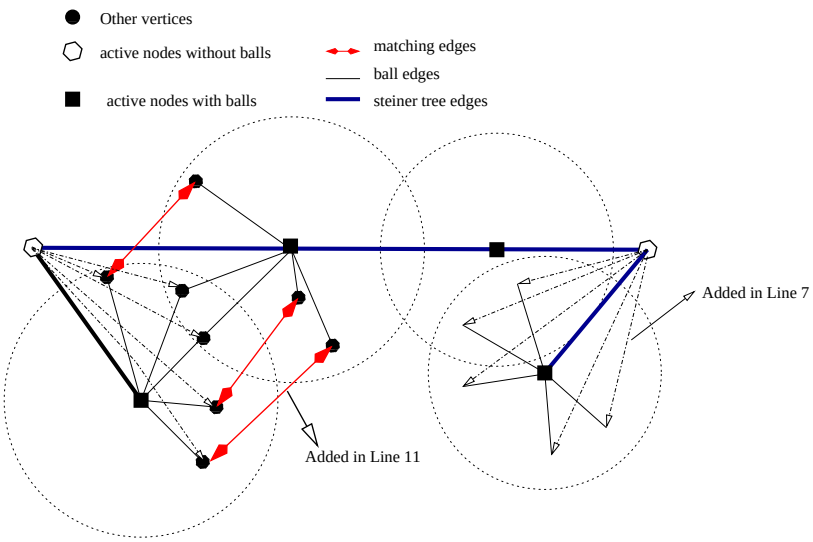

FIG. 3.2. *A sample showing how the edges between the balls of active nodes are added and how the active nodes without a ball are connected to the balls of other active nodes.*

Proof. Consider an arbitrary vertex $v_i \in P_2$. Since degree of v_i is at most two, there are at most two other nodes, say v_j, v_ℓ, with $v_iv_j \in P_2$ and $v_iv_\ell \in P_2$. Note that $v_i, v_j, v_\ell \in \mathcal{S}$. For any edge xy with $x \in S_{v_i}$ and $y \in S_{v_j}$ that we add in line 11: $c_{xy} \leq c_{xv_i} + c_{v_iv_j} + c_{v_jy}$. As the matching added between the balls of v_i and v_j has size at most λ, the cost of this matching is at most $\lambda c_{v_iv_j} + s_{v_i} + s_{v_j}$. Similarly, the cost of matching added between the vertices of S_{v_i} and S_{v_ℓ} is at most $\lambda c_{v_iv_\ell} + s_{v_i} + s_{v_\ell}$. Therefore, the total cost of all the edges added in line 11 is at most: $\lambda \sum_{e \in P_2} c_e + 2\sum_{v_i \in P_2} s_{v_i}$. Using Equation (3.1) and noting that $V_2 \subseteq V_0$ and $c(P_2) \leq 2c(T_0)$ (by Corollary 3.9), the total cost of the edges added in line 11 is at most $2\lambda \sum_{e \in T_0} c_e + 2\sum_{v_i \in T_0} s_{v_i} \leq 8\rho\text{OPT}$. □

LEMMA 3.13. *The cost of edges of H added in lines 5 and 7 is at most $(2 + 28\rho)$OPT.*

Proof. We consider the following cases:

Case 1: First consider an active node $v_i \notin \mathcal{S}$ with $i \leq i^*$ (i.e. $v_i \in I$); so $i \geq 2$ and we have added vertex v_i to H plus every edge uv_i with $u \in S_{p(i)}$ in line 7. Let us assume $p(i) = j^*$. Note that $c_{uv_i} \leq c_{uv_{j^*}} + c_{v_iv_{j^*}}$. So the total cost of edges added at line 7 (for adding vertex v_i to H) is at most $\lambda c_{v_iv_{j^*}} + s_{v_{j^*}}$. By definition of j^*: $c_{v_iv_{j^*}} \leq 4\mu_i + 2\mu_{j^*}$. Noting that $s_{v_{j^*}} \leq s_{v_i}$ (and therefore $\mu_{j^*} \leq \mu_i$), the total cost of edges added for v_i is at most $6\lambda\mu_i + s_{v_i} \leq 7s_{v_i}$.

Case 2: Now consider an active node $v_i \in \mathcal{S}$ for which we add all the vertices in $S_{v_i} \cup \{v_i\}$ to H and make a $(\lambda+1)$-clique on these vertices in line 5. For any two vertices $x, y \in S_{v_i} \cup \{v_i\}$: $c_{xy} \leq c_{v_ix} + c_{v_iy}$. Since each vertex x is incident with λ edges in this clique, the total cost of the edges of the clique is at most $\lambda \sum_{y \in S_{v_i}} c_{v_iy} = \lambda s_{v_i}$.

Thus, the total cost of the edges added in lines 5 and 7 is at most $\sum_{v_i \in V_2} \lambda s_{v_i} + \sum_{v_i \in I} 7s_{v_i}$ which by Lemma 3.7 is bounded by $(2 + 28\rho)$OPT. □

THEOREM 3.14. *Algorithm 1 (in Figure 3.1) returns a (simple) graph of size at least $k - \frac{\lambda}{7}$ which is λ-edge-connected and has cost at most $(2 + 36\rho)$OPT.*

Proof. By Lemma 3.10 and Lemma 3.11, H is a λ-edge-connected graph with at least $k - \frac{\lambda}{7}$ nodes. Using Lemmas 3.12 and 3.13: $c(H) \leq 8\rho\text{OPT} + (2 + 28\rho)\text{OPT} = (2 + 36\rho)\text{OPT}$. □

4. From size $k - \lambda/7$ to size k. As mentioned in the previous section, graph H computed by Algorithm 1 has at least $k - \lambda/7$ vertices and has non-empty intersection with G^* (at least root r belongs to both). If $|H| \geq k$ then we are done. Otherwise, in this section we show how to augment H at low cost to have at least k vertices without loosing its edge-connectivity. For every vertex $u \in G \backslash H$, let the distance of u to H, denoted by $d(u, H)$, be the cost of the cheapest edge from u to a vertex in H. Our construction in this section is based on the following two lemmas.

LEMMA 4.1. *If there is a vertex $u \in G \backslash H$ with $|S_u \cup H| \geq k$ then we can augment H to a (k, λ)-subgraph with an additional cost of $\lambda s_u + \lambda d(u, H) + c(H)$.*

1 H_1:
2 If there is a vertex $u \in G \backslash H$ with the smallest $s_u + d(u, H)$ value such that $|S_u \cup H| \geq k$ then augment H to H_1 using Lemma 4.1.
3 Otherwise let $H_1 = H$.
4 H_2:
5 Find a minimum weight matching M of size $k - |H|$ between $G \backslash H$ and H,
6 Augment H to H_2 according to Lemma 4.2.
7 return either of H_1 or H_2 that has at least k nodes and the least cost.

FIG. 4.1. *Algorithm 2, which augments H (the result of Algorithm 1) to have at least k vertices*

Proof. Consider such a vertex $u \in G \backslash H$. We add u and S_u to H to get a total of at least k vertices. To establish λ-edge-connectivity, we first build a clique around vertices $\{u\} \cup S_u$. This costs at most λs_u. If $S_u \cap H \neq \emptyset$ then u and S_u have λ-edge-connectivity to H. Otherwise, let v be the closest neighbor of u in H. We connect u to the nearest λ neighbors of v in H. For any such neighbor y of v, $c_{uy} \leq c_{uv} + c_{vy}$. Therefore, the cost of adding these edges is at most $\lambda c_{uv} + c(H) = \lambda d(u, H) + c(H)$ and the total cost of the final (k, λ)-graph obtained is at most $\lambda d(u, H) + 2c(H) + \lambda s_u$. □

LEMMA 4.2. *Let M be a matching of size $k - |H|$ from $G \backslash H$ to H. Then we can augment H to a (k, λ)-subgraph with an additional cost of $\lambda c(M) + 2c(H)$, where $c(M)$ and $c(H)$ denote the cost of edges of matching M and graph H, respectively.*

Proof. Let U be the set of vertices in $G \backslash H$ that participate in M. For every $u \in U$ we connect u to the λ nearest neighbors of $M(u)$. This costs at most $\lambda d(u, M(u)) + s_{M(u)}(H)$, by triangle inequality. Thus, the total cost of the edges added is at most:

$$\sum_{u \in U} (\lambda d(u, M(u)) + s_{M(u)}(H)) \leq \lambda c(M) + 2c(H)$$

where the inequality follows from the fact that $\cup_{u \in U} S_{M(u)}(H)$ counts every edge of H at most twice and, therefore, its cost is at most twice as much as the cost of H. □

Using these two lemmas, we compute two different graphs $H_1 \supseteq H$ and $H_2 \supseteq H$ that are λ-edge-connected and return whichever has at least k nodes and the least cost. The description of the algorithm is given in Figure 4.1. In what follows we show that we either have a matching from $G \backslash H$ to H of size $k - |H|$ and cost $O(\text{OPT}/\lambda)$ or there is a vertex $u \in G \backslash H$ with $d(u, H) + s_u = O(\text{OPT}/\lambda)$. Consequently, we can apply one of the above two lemmas and obtain a (k, λ)-subgraph with small cost.

Figure 4.2 shows how graphs H_1 and H_2 are built from expanding H. To perform line 5 of the algorithm, we can use any minimum cost (bipartite) matching algorithm (see [15]).

LEMMA 4.3. *If $H_1 \neq H$ then it is λ-edge connected and has at least k vertices. Also, H_2 is λ-edge-connected and has at least k vertices.*

Proof. By the description of Algorithm 2 (in Figure 4.1), if $H_1 \neq H$ then while building H_1 we have added at least $k - |H|$ new vertices since $|S_u \cup H| \geq k$; so in this case $|H_1| \geq k$. For H_2, we add the vertices of U and $|U| = k - |H|$, so $|H_2| = k$. For λ-edge-connectivity, note that H was originally λ-edge-connected. If $H_1 \neq H$ we have added a vertex u together with S_u. The vertices in $S_u \cup \{u\}$ form a clique, so are λ-edge-connected among themselves. Also, if $S_u \cap H = \emptyset$, u is λ-edge connected to some vertex $v \in H$ by the λ edges added between u and the λ nearest neighbors of v (in H). By transitivity, this implies the λ-edge connectivity of H_1. For the connectivity of H_2, every new vertex $u \in U$ is connected to at least λ vertices in H which makes it λ-edge connected to all the vertices in H. □

LEMMA 4.4. *If H is the solution of Algorithm 1, then the solution of Algorithm 2 has cost at most* $\max\{12\text{OPT} + 3c(H), 13\text{OPT} + 2c(H)\}$.

Proof.

We prove this by considering the following two cases.

Case 1: $|G^* \backslash H| \leq \lambda/3$

In this case we show that cost of H_2 is $O(\text{OPT})$. Every vertex $u \in G^* \backslash H$ is connected, in G^*, to at least $2\lambda/3$ vertices

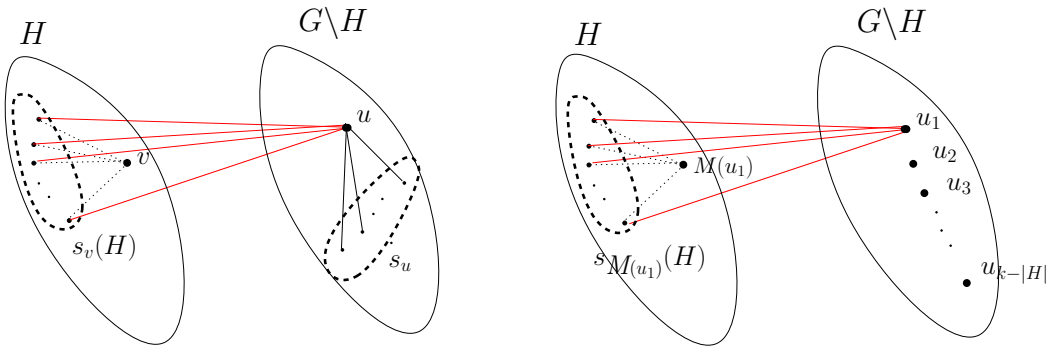

FIG. 4.2. *Constructing H_1 (in the left picture) and H_2 (in the right picture)*

in $G^* \cap H$ and (by a simple averaging argument) the distance of u to at least $\lambda/3$ of them is at most $3s_u(G^*)/\lambda$ (recall that $s_u(G^*)$ is the sum of distances from u to its λ closest neighbors in G^*). Therefore there is a matching $\tilde{M}$ between $G^* \backslash H$ and $G^* \cap H$ such that $d(u, \tilde{M}(u)) \leq 3s_u(G^*)/\lambda$ for every $u \in G^* \backslash H$, and $|\tilde{M}| = |G^* \backslash H| \geq k - |H| = |M|$, where M is the matching we find in the algorithm. Since M is a minimum weight matching:

$$c(M) \leq c(\tilde{M}) \leq \frac{3}{\lambda} \sum_{u \in G^* \backslash H} s_u(G^*) \leq \frac{6\text{OPT}}{\lambda}, \tag{4.1}$$

where we use the lower bound of $\sum_{u \in G^* \backslash H} s_u(G^*) \leq \sum_{u \in G^*} s_u(G^*) \leq 2\text{OPT}$. Therefore, according to Lemma 4.2, $c(H_2) \leq c(H) + \lambda c(M) + 2c(H) \leq 3C(H) + 6\text{OPT}$.

Case 2: $|G^* \backslash H| > \lambda/3$

In this case (again by an averaging argument) there is a set U of $\lambda/6$ vertices in $G^* \backslash H$ such that $s_u \leq 12\text{OPT}/\lambda$ for each $u \in U$ (Otherwise, the remaining at least $\lambda/6$ vertices would have total s value more than $\lambda/6 \times 12\text{OPT}/\lambda \geq 2OPT$ which is a contradiction). If every vertex in U is connected, in G^*, to at least $2\lambda/6$ vertices in $G^* \cap H$ then, with an argument similar to the previous case and by using Lemma 4.2, we can upper bound the cost of H_2 by: $c(H_2) \leq 12\text{OPT} + 3c(H)$,

Otherwise, let $u \in U$ be a vertex such that S_u has more than $4\lambda/6 > \lambda/7$ vertices in $G^* \backslash H$. In this case we show that $c(H_1) = O(\text{OPT})$ (note that in this case $H_1 \neq H$). First note that $\lambda s_u \leq 12\text{OPT}$ as each vertex $u \in U$ has $s_u \leq 12\text{OPT}/\lambda$. Furthermore, since $H \cap G^*$ is non-empty (here we use the assumption that r belongs to both H and $G*$), u must have distance at most OPT/λ to some vertex $v \in H$ (because there are at least λ edge disjoint paths between u and v and the cost of each is at least c_{uv} by the triangle inequality). Now using Lemma 4.1, and the above facts that $\lambda s_u \leq 12\text{OPT}$ and $d(u, H) \leq \text{OPT}/\lambda$, it follows that H_1 costs at most $13\text{OPT} + 2c(H)$ □

Combining the two Algorithms 1 (in Figure 3.1) and 2 (in Figure 4.1), and using Lemmas 4.4 and 4.3, and Theorem 3.14 we have an algorithm that returns a λ-edge-connected subgraph on at least k vertices with cost at most $\max\{12\text{OPT} + 3c(H), 13\text{OPT} + 2c(H)\} \leq 3(2 + 36\rho)\text{OPT} + 12\text{OPT} = (18 + 108\rho)\text{OPT}$. Thus, we have the following theorem which is essentially Theorem 1.1:

THEOREM 4.5. *There is a polynomial time algorithm for the (k, λ)-subgraph problem on graphs with metric edge costs which has approximation factor at most $18 + 108\rho$, with $\rho \leq 4$ being the best approximation factor for the k-Steiner tree problem.*

5. Concluding Remarks. In this paper, we proved that the (k, λ)-subgraph problem with metric costs has a polynomial time $O(1)$-approximation algorithm. However, the approximation ratio of our algorithm is relatively large (namely 450). Although it is very likely that one can achieve an approximation ratio close to 100 using the same algorithm by fine tuning the parameters, getting a small constant factor approximation seems to be challenging. Our algorithm does not seem to work for vertex-connectivity requirements. It is an interesting open question whether one can obtain a constant factor approximation for this variation.

For general cost functions, the only known results on this problem (that we are aware of) are the papers [12, 4] which prove that the $(k, 2)$-subgraph problem (on general graphs) has $O(\log^2 n)$-approximation, even if we require

2-node-connectivity in the solution (instead of 2-edge-connectivity). Even for the special case of $\lambda = 3$, there is no known non-trivial approximation algorithm or lower bound (hardness result).

6. Acknowledgments. We thank anonymous referees whose comments improved the presentation of the paper. The second author also thanks Joseph Cheriyan for some initial discussions on the problem.

REFERENCES

[1] B. Awerbuch, Y. Azar, A. Blum and S. Vempala, *New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen*, SIAM J. Computing 28(1):254-262, 1999.

[2] A. Blum, S. Chawla, D. Karger, T. Lane, A. Meyerson, and M. Minkoff, *Approximation Algorithms for Orienteering and Discounted-Reward TSP*, SIAM J. on Computing 28(1):254-262, 1999. Earlier version in Proc of STOC 1995.

[3] A. Blum, R. Ravi, and S. Vempala, *A constant-factor approximation algorithm for the k-MST problem*, J. Comput. Syst. Sci. 58(1): 101-108, 1999. Earlier in Proceedings of the 28th Annual ACM Symposium on the Theory of Computing (STOC '96), pp. 442-448.

[4] C. Chekuri and N. Korula, *Prunning 2-connected graphs*, In Proceedings of the 28th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), 2008

[5] C. Chekuri, N. Korula, and M. Pál, *Improved Algorithms for Orienteering and Related Problems*, In Proc of ACM-SIAM SODA, 2008.

[6] J. Cheriyan and A Vetta, *Approximation algorithms for network design with metric costs*, SIAM J. Discr. Math. 21(3): 612–636, 2007. Earlier version in Proceedings of the thirty-seventh annual ACM symposium on Theory of computing (STOC) 2005, 167–175.

[7] F. Chudak, T. Roughgarden, and D. P. Williamson, *Approximate MSTs and Steiner trees via the primal-dual method and Lagrangean relaxation*, Math. Program. 100(2):411-421, 2004.

[8] A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan, *Detecting High Log-Densities - an $O(n^{1/4})$ Approximation for Densest k-Subgraph*, In Proceedings of the 42th ACM Symposium on Theory of Computing (STOC 2010).
U. Feige, G. Kortsarz and D. Peleg, *The dense k-subgraph problem*, Algorithmica, 29(3): 410-421, 2001. Preliminary version in the Proc. 34-th IEEE Symp. on Foundations of Computer Science (FOCS) pp 692-701, 1993.

[9] N. Garg, *A 3-Approximation for the minim tree spanning k vertices*, In Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS), 302-309, 1996.

[10] N. Garg, *Saving an epsilon: a 2-approximation for the k-MST problem in graphs*, In Proceedings of the thirty-seventh annual ACM symposium on Theory of computing (STOC), 396 - 402, 2005.

[11] K. Jain, *A factor 2 approximation algorithm for the generalized Steiner network problem*, Combinatorica, 21:39-60, 2001.

[12] L. Lau, S. Naor, M. Salavatipour, and M. Singh, *Survivable Network Design with Degree or Order Constraints*, Submitted to SIAM J. on Computing. Earlier version in Proceedings of the thirty-nineth annual ACM symposium on Theory of computing (STOC), 2007.

[13] S. Rajagopalan and V. Vazirani, *Logarithmic approximation of minimum weight k trees*, unpublished manuscript, 1995.

[14] R. Ravi, R. Sundaram, M.V. Marathe, D.J. Rosenkrants, and S.S. Ravi, *Spanning trees short or small*, SIAM Journal on Discrete Mathematics, 9(2):178-200, 1996.

[15] A. Schrijver, Combinatorial Optimization, Springer, 2003.