Improved Approximation Algorithms for the Min-max Tree Cover and Bounded Tree Cover Problems

M. Reza khani* Mohammad R. Salavatipour†

Abstract

In this paper we provide improved approximation algorithms for the Min-Max Tree Cover and Bounded Tree Cover problems. Given a graph $G = (V, E)$ with weights $w : E \to \mathbb{Z}^+$, a set $T_1, T_2, \ldots, T_k$ of subtrees of G is called a tree cover of G if $V = \bigcup_{i=1}^{k} V(T_i)$. In the Min-Max k-tree Cover problem we are given graph G and a positive integer k and the goal is to find a tree cover with k trees, such that the weight of the largest tree in the cover is minimized. We present a 3-approximation algorithm for this improving the two different approximation algorithms presented in [1, 5] with ratio 4. The problem is known to have an APX-hardness lower bound of $\frac{3}{2}$ [12]. In the Bounded Tree Cover problem we are given graph G and a bound λ and the goal is to find a tree cover with minimum number of trees such that each tree has weight at most λ. We present a 2.5-approximation algorithm for this, improving the 3-approximation bound in [1].

1 Introduction

The study of problems in which the vertices of a given graph are needed to be covered with special subgraphs, such as trees, paths, or cycles, with a bound on the number of subgraphs used or their weights has attracted a lot of attention in operations research and computer science community. Such problems arise naturally in many applications such as vehicle routing. As an example, in a vehicle routing problem with min-max objective, we are given a weighted graph $G = (V, E)$ in which each node represents a client. The goal is to dispatch a number of service vehicles to service the clients and the goal is to minimize the largest client waiting time, which is equivalent to minimizing the total distance traveled by the vehicle which has traveled the most. Observe that the subgraph traveled by each vehicle is a walk that can be approximated with a tree. This problem, under the name of "Nurse station location", was the main motivation in [5] to study these problems. In a different scenario, we may want to guarantee an upper bound for the service time; so we are given a bound on the distance traveled by each vehicle and the objective is to minimize the number of required vehicles needed to satisfy this guarantee. Min-max and bounded vehicle routing problems are part of an active body of research in the literature and have several application (see e.g.[2, 5, 1, 11, 12] and the references there).

In this paper we consider Min-Max k-Tree Cover Problem (MMkTC) and Bounded Tree Cover Problem (BTC) defined formally below. Suppose we are given an undirected graph $G = (V, E)$ and a weight function $w : E \to \mathbb{Z}^+$. For every subgraph H of G we use $V(H)$ and $E(H)$ to denote the

*Department of Computing Science,University of Alberta, Edmonton, Alberta T6G 2E8, Canada. e-mail: khani@ualberta.ca

†Toyota Tech. Inst. at Chicago, and Department of Computing Science, University of Alberta, Edmonton, Alberta T6G 2E8, Canada. e-mail: and mreza@cs.ualberta.ca. Supported by NSERC and an Alberta Ingenuity New Faculty award.

set of vertices and edges of H, respectively. A set $T_1, T_2, \ldots, T_k$ of subtrees of G is called a tree cover of G if every vertex of V appears in at least one T_i ($1 \leq i \leq k$), i.e. $V = \bigcup_{i=1}^{k} V(T_i)$. Note that the trees in a tree-cover are not necessarily edge-disjoint (thus may share vertices too). The weight of a tree T_i is $W(T_i) = \sum_{e \in T_i} w(e)$. In the Min-Max k-tree Cover problem (MMkTC) we are given the weighted graph G and a positive integer k and the goal is to find a tree cover with k trees, which we call a k-tree cover, such that the weight of the largest tree in the cover is minimized. In the Bounded Tree Cover problem (BTC), we are given the weight G and a parameter λ and the goal is to find a tree cover with minimum number of trees such that the weight of every tree in the cover is at most λ. There are other variations of these problems in which one wants to cover the vertices of a graph with paths or cycles (instead of trees), however the known algorithms for these variations (e.g. see [1]) solve the problem for tree versions first and then take a walk of the trees to obtain a path. So apart from their real world applications [5], covering graphs with trees have been the main step for covering graphs with paths and cycles.

Related Works: Even et al. [5] and Arkin et al. [1] gave two different 4-approximation algorithms for MMkTC. It is shown that MMkTC is APX-hard in [12], specifically a lower bound of $\frac{3}{2}$. The best approximation factor for BTC is due to Arkin et al. [1] which give a 3-approximation algorithm. It is easy to see that BTC is APX-hard even in the case when G is a weighted graph with height one, by an easy reduction from the bin packing problem.

Even et al. [5] give a 4-approximation algorithm for the rooted version of MMkTC in which k vertices are given in input and each tree of the trees in a k-tree cover has to be rooted at one of them. Nagamochi and Okada[10] give a $(3 - \frac{2}{p+1})$-approximation algorithm for MMkTC when all the trees have to be rooted at a given vertex r. They also give a $(2 - \frac{2}{k+1})$-approximation algorithm for MMkTC when the underlying metric is a tree and $(2+\epsilon)$-approximation algorithm for MMkTC when the underlying metric is a tree and each tree should be rooted at a certain vertex r.

In addition to trees, covering with other objects, such as tours, paths, and stars are studied in the literature. Frederickson et al. [6] gave an $(e + 1 - \frac{1}{k})$-approximation algorithm for covering a metric graph with k tours rooted at a given vertex (called k-traveling salesperson problem or k-TSP) where e is the best approximation ratio for the classic TSP problem. Other different variations of min-max and bounded vehicle routing problems are also studied in the literature (see e.g. [1, 11, 13, 9, 7]). Another related problem to k-TSP is called k-Traveling Repairman Problem (KTR) in which instead of minimizing the total lengths of the tour the objective function is to minimize the total latency of the nodes where the latency of each node is the distance traveled (time elapsed) before visiting that node for the first time. The case of $k = 1$ is known as the minimum latency problem. The best known approximation algorithm for $k = 1$ is 3.59 due to [3] and the best known approximation for KTR is $2(2+\alpha)$ [4] where α is the best approximation ratio for the problem of finding minimum tree spanning k vertices a.k.a k-MST (see also [8] and the references there).

Our Result: In this paper we improve the approximation ratios for both MMkTC and BTC problems.

Theorem 1 *There is a polynomial time* 3*-approximation algorithm for the MMkTC problem.*

This improves upon the 4-approximation algorithms of [1, 5]

Theorem 2 *There is a polynomial time* 2.5*-approximation algorithm for the BTC problem.*

This improves upon the 3-approximation algorithm of [1]. We prove Theorem 1 in Section 3 and Theorem 2 in Section 4.

2 Preliminaries

For a connected subgraph $H \subseteq G$ by tree weight of H we mean the weight of a minimum spanning tree (MST) of H and denote this value by $W_T(H)$. Note that this is different from the weight of H, i.e. $W(H)$ which is the sum of weights of *all* the edges of H. In every solution to either MMkTC or BTC problem, we can replace every edge uv of a tree in the cover with the shortest path between u, v in the graph without increasing the cost of the tree and the solution still remains feasible. Therefore, without loss of generality, if the input graph is G and $\tilde{G}$ is the shortest-path metric completion of G, we can assume that we are working with the complete graph $\tilde{G}$. Any solution to $\tilde{G}$ can be transformed into a feasible solution of G (for MMkTC or BTC) without increasing the cost. The following lemma will be useful in our algorithms for both the MMkTC and BTC problems.

Lemma 1 *Suppose $G = (V, E)$ is a graph which has a k-tree cover $\mathcal{T} = \{T_1, \ldots, T_k\}$, with maximum tree weight of λ and let $\lambda' \leq \lambda$ be a given parameter. Assume we delete all the edges e with $w(e) > \lambda'$ (call them heavy edges) and the resulting connected components be $C_1, \ldots, C_p$. Then $\Sigma_{i=1}^{p} W_T(C_i) \leq k\lambda + (k-p)\lambda'$.*

Proof. Let $G' = \bigcup_{i=1}^{p} C_i$ be the graph after deleting the heavy edges. Each tree in $\mathcal{T}$ might be broken into a number of subtrees (or parts) after deleting heavy edges; let $\mathcal{T}'$ denote the set of these broken subtrees, $|\mathcal{T}'| = k'$, and n_i be the number of trees of $\mathcal{T}'$ in component C_i. The total weight of the subtrees in $\mathcal{T}'$ is at most $k\lambda - (k'-k)\lambda'$, since the weight of each tree in $\mathcal{T}$ is at most λ and we have deleted at least $k'-k$ edges form the trees in $\mathcal{T}$ each having weight at least λ'. In each component C_i we use the cheapest $n_i - 1$ edges that connect all the trees of $\mathcal{T}'$ in C_i into one spanning tree of C_i. The weight of each of these added edges is no more than λ' and we have to add a total of $k' - p$ such edges (over all the components) in order to obtain a spanning tree for each component C_i. Thus, the total weight of spanning trees of the components C_i's is at most $k\lambda - (k'-k)\lambda' + (k'-p)\lambda' = k\lambda + (k-p)\lambda'$. ∎

Through our algorithms we may need to break a large tree into smaller trees that cover the (vertices of) original tree, are edge-disjoint, and such that the weight of each of the smaller trees is bounded by a given parameter. We use the following lemma which is implicitly proved in [5] (in a slightly weaker form) in the analysis of their algorithm.

Lemma 2 *Given a tree T with weight $W(T)$ and a parameter $\beta > 0$ such that all the edges of T have weight at most β, we can edge-decompose T into trees $T_1, \ldots, T_k$ with $k \leq \max(\lfloor \frac{W(T)}{\beta} \rfloor, 1)$ such that $W(T_i) \leq 2\beta$ for each $1 \leq i \leq k$.*

Proof. The idea is to "split away" (defined below) trees of weight in interval $[\beta, 2\beta)$ until we are left with a tree of size smaller than 2β. This process of "splitting away" is explained in [5]. We bring it here for the sake of completeness. Consider T being rooted at an arbitrary node $r \in T$. For every vertex $v \in T$ we use T_v to denote the subtree of T rooted at v; for every edge $e = (u, v)$ we use T_e to denote the subtree rooted at u which consist of T_v plus the edge e. Subtrees are called light, medium, or heavy depending on whether their weight is smaller than β, in the range $[\beta, 2\beta)$, or $\geq 2\beta$, respectively. For a vertex v whose children are connected to it using edges $e_1, e_2, \ldots, e_l$ splitting away subtree $T' = \bigcup_{i=a}^{b} T_{e_i}$ means removing all the edges of T' and vertices of T' (except v) from T and putting T' in our decomposition. Note that we can always split away a medium tree and put it in our decomposition and all the trees we place in our decomposition are edge-disjoint. So assume that all the subtrees of T are either heavy or light. Suppose T_v is a heavy subtree whose

children are connected to v by edges $e_1, e_2, \ldots$ such that all subtrees $T_{e_1}, T_{e_2}, \ldots$ are light (if any of them is heavy we take that subtree). Let i be the smallest index such that $T' = \bigcup_{a=1}^{i} T_{e_a}$ has weight at least β. Note that T' will be medium as all T_{e_j}'s are light. We split away T' from T and repeat the process until there is no heavy subtree of T (so at the end the left-over T is either medium or light).

If $W(T) \leq 2\beta$ then we do not split away any tree (since the entire tree T is medium) and the theorem holds trivially. Suppose the split trees are $T_1, T_2, \ldots, T_d$ with $d \geq 2$ with $W(T_i) \in [\beta, 2\beta)$ for $1 \leq i < d$. The only tree that may have weight less than β is T_d. Note that in the step when we split away T_{d-1} the total weight of the remaining tree was at least 2β, therefore we can assume that the average weight of T_{d-1} and T_d is not less than β. Thus, the average weight of all T_i's is not less than β which proves that d cannot be greater than $\lfloor \frac{W(T)}{\beta} \rfloor$. ■

3 3-approximation algorithm for MMkTC

In this section we present our 3-approximation algorithm for MMkTC. Before describing our algorithm we briefly explain the 4-approximation algorithm of [5]. Suppose that the value of the optimum solution to the given instance of MMkTC is OPT and let $\lambda \geq$ OPT be a value that we have guessed as an upper bound for OPT. The algorithm of [5] will either produce a k-tree cover whose largest tree has weight at most 4λ or will declare that OPT must be larger than λ, in which case we adjust our guess λ. So assume we have guessed λ with $\lambda \geq$ OPT.

For simplicity, let us assume that G is connected and does not have any edge e with $w(e) > \lambda$ as these clearly cannot be part of any optimum k-tree cover. Let T be a MST of G and $\mathcal{T} = \{T_1, \ldots, T_k\}$ be an optimum k-tree cover of G. We can obtain a spanning tree of G from $\mathcal{T}$ by adding at most $k-1$ edges between the trees of $\mathcal{T}$. This adds a total of at most $(k-1)\lambda$ since each edge has weight at most λ. Thus, $W(T) \leq \sum_{i=1}^{k} W(T_i) + (k-1)\lambda \leq (2k-1)\lambda$. Therefore, if we start from a MST of G, say T, and we split away trees of size in $[2\lambda, 4\lambda)$ then we obtain a total of at most $(2k-1)\lambda / 2\lambda k \leq k$ trees each of which has weight at most 4λ. In reality the input graph might have edges of weight larger than λ. First, we delete all such edges (called heavy edges) as clearly these edges cannot be part of an optimum solution. This might make the graph disconnected. Let $\{G_i\}_i$ be the connected components of the graph after deleting these heavy edges and let T_i be a MST of G_i. For each component G_i the algorithm of [5] splits away trees of weight in $[2\lambda, 4\lambda)$. Using Lemma 2 one can obtain a k_i-tree cover of each G_i with $k_i \leq \max(W_T(G_i)/2\lambda, 1)$ with each tree having weight at most 4λ. A similar argument as the one above shows (Lemma 3 in [5]) that $\sum_i (k_i + 1) \leq k$. One can do a binary search for the value λ which yields a polynomial 4-approximation.

Now we describe our algorithm. As said earlier, we work with the metric graph $\tilde{G}$. We use OPT to denote an optimal solution and OPT to denote the weight of the largest tree in OPT. Similar to [5] we assume we have a guessed value λ for OPT and present an algorithm which finds a k-tree cover with maximum tree weight at most 3λ if $\lambda \geq$ OPT. By doing a binary search for λ we obtain a 3-approximation algorithm that runs in time polynomial in input size. First, we delete all the edges e with $w(e) > \lambda/2$ to obtain graph G'. Let $C_1, \ldots, C_\ell$ be the components of G' whose tree weight (i.e. the weight of a MST of that component) is at most λ (we refer to them as *light components*), and let $C_{\ell+1}, \ldots, C_{\ell+h}$ be the components of G' with tree weight greater than λ (which we refer to as *heavy components*). The general idea of the algorithm is as follows: For every light component we do one of the following three: find a MST of it as one tree in our tree cover, or we decide to connect it to another light components with an edge of weight at most λ in which case we find a component with MST weight at most 3λ and put that MST as a tree in our solution, or

we decide to connect a light component to a heavy component. For heavy components (to which some light components might have been attached) we split away trees with weight in $[\frac{3}{2}\lambda, 3\lambda)$. We can show that if this is done carefully, the number of trees is not too big. We explain the details below.

For every light component C_i let $w_{min}(C_i)$ be the minimum edge weight (in graph $\tilde{G}$) between C_i and a heavy component if such an edge exists with weight at most λ, otherwise set $w_{min}(C_i)$ to be infinity. We might decide to combine C_i with a heavy component (one to which C_i has an edge of weight $w_{min}(C_i)$). In that case the tree weight of that heavy component will be increased by $A(C_i) = W_T(C_i) + w_{min}(C_i)$. The following lemma shows how we can cover the set of heavy components and some subset of light components with a small number of trees whose weight is not greater than 3λ.

Lemma 3 *Let $L_s = \{C_{l_1}, \ldots, C_{l_s}\}$ be a set of s light-components with bounded $A(C_i)$ values. If $\sum_{1\le i\le s} A(C_{l_i}) + \sum_{\ell+1\le i\le \ell+h} W_T(C_i) \le x - h\frac{\lambda}{2}$, then we can cover all the nodes in the heavy-components and in components of L_s with at most $\lfloor \frac{2x}{3\lambda} \rfloor$ trees with maximum tree weight no more than 3λ.*

Proof. First we find a MST in each heavy component and in each component of L_s, then we attach the MST of each C_{l_i} to the nearest spanning tree found for heavy components. As we have h heavy components, we get a total of h trees, call them $T_1, \ldots, T_h$. From the definition of $A(C_{l_j})$, the total weight of the constructed trees will be

$$\sum_{i=1}^{h} W(T_i) = \sum_{1\le j\le s} A(C_{l_j}) + \sum_{\ell+1\le i\le \ell+h} W_T(C_i) \le x - h\frac{\lambda}{2}, \tag{1}$$

where the last inequality is by the assumption of lemma. Now to each of the h constructed trees we will apply the procedure of Lemma 2 with $\beta = \frac{3}{2}\lambda$ to obtain trees of weight at most 3λ. This gives at most $\sum_{1\le i\le h} \max(\lfloor \frac{2W(T_i)}{3\lambda} \rfloor, 1)$ trees. To complete the proof of lemma it is sufficient to prove the following:

$$\sum_{1\le i\le h} \max(\lfloor \frac{2W(T_i)}{3\lambda} \rfloor, 1) \le \lfloor \frac{2x}{3\lambda} \rfloor. \tag{2}$$

Consider T_i for an arbitrary value of i. If T_i has been split into more than one tree, by Lemma 2 we know that the amortized weight of the split trees is not less than $\frac{3}{2}\lambda$. If T_i is not split, as T_i contains a spanning tree over a heavy component, $W(T_i) \ge \lambda$. Thus every split tree has weight at least $\frac{3}{2}\lambda$ excepts possibly h trees which have weight at least λ. Therefore, if the total number of split trees is r, they have a total weight of at least $r\frac{3}{2}\lambda - h\frac{\lambda}{2}$. Using Equation (1), it follows that r cannot be more than $\lfloor \frac{2x}{3\lambda} \rfloor$. ∎

Before presenting the algorithm we define a graph H formed according to the light components.

Definition 1 *For two given parameters a, b, graph H has $\ell + a + b$ nodes: ℓ (regular) nodes $v_1, \ldots, v_\ell$, where each v_i corresponds to a light component C_i, a dummy nodes called* null *nodes, and b dummy nodes called* heavy *nodes. We add an edge with weight zero between two regular nodes v_i and v_j in H if and only if $i \ne j$ and there is an edge in $\tilde{G}$ with length no more than λ connecting a vertex of C_i to a vertex of C_j. Every null node is adjacent to each regular node v_i ($1 \le i \le \ell$) with weight zero. Every regular node $v_i \in H$ whose corresponding light component C_i has finite value of $A(C_i)$ is connected to every heavy node in H with an edge of weight $A(C_i)$. There are no other edges in H.*

Inputs: $G(V, E)$, k, λ
Output: A set S which is a k-tree cover with maximum tree size 3λ.

1. Build $\tilde{G}$ which is the shortest-path metric completion of G and then delete all edges with weight more than $\frac{\lambda}{2}$; let $C_1, \ldots, C_{\ell+h}$ be the set of ℓ light and h heavy components created.
2. For $a : 0 \to \ell$
   (a) For $b : 0 \to \ell$
      i. $S \leftarrow \emptyset$
      ii. Construct H (as described above) with a null nodes and b heavy nodes.
      iii. Find a perfect matching with the minimum cost in H; if there is no such perfect matching continue from Step 2a ,
      iv. Attach each light-component C_i to its nearest heavy component (using the cheapest edge in $\tilde{G}$ between the two) if v_i is matched to a *heavy* node in the matching
      v. Decompose all the heavy components and the attached light components using Lemma 3 and add the trees obtained to S
      vi. If a vertex v_i is matched to a null node, add a MST of C_i to S.
      vii. For every matching edge between two regular nodes v_i and v_j join a MST of C_i and a MST of C_j using the cheapest edge among them (in G) and add it to S.
      viii. If $|S| \leq k$ then return S.
3. return failure

Figure 1: MMkTC Algorithm

Theorem 3 *Algorithm MMkTC (Figure 1) finds a k-tree cover with maximum tree weight at most* 3λ*, if* $\lambda \geq$ OPT.

Proof. Through out the whole proof we assume $\lambda \geq$ OPT. In order to bound the maximum weight of the cover with 3λ we need to use the optimal k-tree cover. Consider an optimal k-tree cover OPT; so each $T \in$ OPT has weight at most λ. First note that every tree $T \in$ OPT can have at most one edge of value larger than $\lambda/2$; therefore each $T \in$ OPT is either completely in one component C_i or has vertices in at most two components, in which case we say it is broken. If T is broken it consists of two subtrees that are in two components (we refer to the subtrees as *broken subtree or part* of T) plus an edge of weight $> \lambda/2$ connecting them; we call that edge the *bridge edge* of T. We characterize the optimal trees in the following way: a tree $T \in$ OPT is called light (heavy) if the entire tree or its broken subtrees (if it is broken) are in light (heavy) components only, otherwise if it is broken and has one part in a light component and one part in a heavy component then we call it a bad tree. We denote the number of light trees, heavy trees, and bad trees of OPT by k_ℓ, k_h, and k_b; therefore $k_\ell + k_h + k_b = k$. We say that a tree $T \in$ OPT is incident to a component if the component contains at least one vertex of T (see Figure 2).

We define multi-graph $H' = (V', E')$ similar to how we defined H except that edges of H' are defined based on the trees in OPT. V' consists of ℓ vertices, one vertex v'_i for each light component C_i. For each light tree $T \in$ OPT, if T is entirely in one component C_i we add a loop to v'_i and if T is broken and is incident to two light components C_i and C_j then we add an edge between v'_i and v'_j. So the total number of edges (including loops) is k_ℓ. There may be some isolated nodes (nodes

without any edges) in H', these are nodes whose corresponding light components are incident to only bad trees. Suppose M is a maximum matching in H' and let U be the set of vertices of H' that are not isolated and are not saturated by M. Because M is maximal, every edge in $E' \setminus M$ is either a loop or is an edge between a vertex in U and one saturated vertex. Therefore:

$$|M| + |U| \leq k_\ell. \tag{3}$$

Note that for every node v'_i (corresponding to a light component C_i) which is incident to a bad tree, that bad tree has a bridge edge (of weight at most λ) between its broken subtree in the light component (i.e. C_i) and its broken subtree in a heavy component. Therefore:

Lemma 4 *For every light component C_i which is incident to a bad tree, and in particular if v'_i is isolated, $A(C_i)$ is finite.*

We define the excess weight of each bad tree as the weight of its broken subtree in the light component plus the bridge edge. Let W_{excess} be the total excess weights of all bad trees of OPT. Note that W_{excess} contains $\sum_{v_i \text{ is isolated}} A(C_i)$, but it also contains the excess weight of some bad trees that are incident to a light component C_i for which v_i is not isolated. Thus:

$$W_{excess} \geq \sum_{v_i \text{ is isolated}} A(C_i). \tag{4}$$

Only at Steps 2(a)v, 2(a)vi, and 2(a)vii the algorithm adds trees to S. First we will show that each tree added to S has weight at most 3λ. At step 2(a)v, according to Lemma 3, all the trees will have weight at most 3λ. At Step 2(a)vi, as C_i is a light components its MST will have weight at most λ. At Step 2(a)vii, the MST of C_i and C_j are both at most λ, and as v_i and v_j are connected in H there is an edge with length no more than λ connecting C_i and C_j; thus the total weight of the tree obtained is at most 3λ. Hence, every tree in S has weight no more than 3λ. The only thing remained is to show that the algorithm will eventually finds a set S that has no more than k trees. We show that in the iteration at which $a = |U|$ and b is equal to the number of isolated nodes in H': $|S| \leq k$.

Lemma 5 *The cost of the minimum perfect matching computed in step 2(a)iii is no more than W_{excess}.*

Proof. Consider the iteration at which $a = |U|$ and b is the number of isolated nodes in H'. In this case, we can find a perfect matching in the following way: for every vertex $v'_i \in U$, $v_i \in H$ can be matched to a null node in H, for every isolated node $v'_i \in H'$, $v_i \in H$ can be matched to a heavy node in H (note that $A(C_i)$ is finite by Lemma 4), for all other vertices $v'_i \in H'$, v'_i is saturated by M, so the corresponding $v_i \in H$ can be matched according to the matching M. Note that the cost of this matching is $\sum_{v_i \text{ is isolated}} A(C_i)$ which is no more than W_{excess} by Equation (3). Since we find a minimum perfect matching in step (2(a)iii). ■

Note that the number of trees added to S at step (2(a)vii) is $|M|$ and the number of trees added at step (2(a)vi) is $|U|$. Thus the total number of trees added to S at these two steps is at most $|M| + |U| \leq k_\ell$ by Equation (3). The weight of the minimum perfect matching found in (2(a)iii) represents the total weight we add to the heavy components in step (2(a)iv). By Lemma 5, we know that the added weight is at most W_{excess}. In Lemma 6 we bound the total weight of heavy components and the added extra weight of matching by $(k_h + k_t) * \frac{3}{2}\lambda - h\frac{\lambda}{2}$. Using Lemma 3 we know that we can cover them by at most $k_h + k_b$ trees. Thus the total number of trees added to S is at most $k_\ell + k_h + k_b = k$. ■

The following lemma will bound the weight of the heavy components and W_{excess}.

Lemma 6 $\sum_{\ell+1\le i\le \ell+h} W_T(C_i) + W_{excess} \le (k_h + k_b) * \frac{3}{2}\lambda - h\frac{\lambda}{2}$, *if* $\lambda \ge$ OPT.

Proof. Again, we assume that $\lambda \ge$ OPT. We show a possible way to form a spanning tree for each heavy component plus the light components attached to it. Then we bound the total weight of these spanning trees.

We can make a spanning tree over a heavy component C_i by connecting all the trees and broken subtrees of the optimum solution that are in that component by adding edges of weight at most $\lambda/2$ between them since each edge in C_i has weight at most $\lambda/2$ (see Figure 2). Therefore, the tree weight of a heavy component can be bounded by the weight of optimal trees or broken subtrees inside it plus some edges to connect them. Suppose p trees of the heavy trees are broken and q of them are completely inside a heavy component; note that $p + q = k_h$. The rest of broken subtrees in heavy components are from bad trees. So overall we have $2p + q + k_b$ trees or broken subtrees in all the heavy components. Each of the q heavy trees that are not broken contribute at most $q\lambda$ to the left hand side. Those p heavy trees that are broken contribute at most $p\lambda/2$ to the left hand side since each of them has an edge of weight more than $\lambda/2$ that is deleted and is between heavy components. By definition of W_{excess}, we can assume the contribution of all bad trees to the left hand side is at most $k_b\lambda$. Thus, the total weight of edges e such that e belongs to an optimal tree and also belongs to a heavy component or is part of W_{excess} (i.e. the broken part of a bad tree plus its bridge edge) is at most $(p + q + k_b)\lambda - p\frac{\lambda}{2}$.

Overall we have $2p + q + k_b$ trees or broken subtrees in all the heavy components. In order to form a spanning tree in each heavy component we need at most $2p+q+k_b-h$ edges connecting the optimal trees and broken subtrees in the heavy components, since we have h heavy components. Since each edge in a component has weight at most $\frac{\lambda}{2}$, the total weight of these edges will be at most $(2p + q + k_b - h)\frac{\lambda}{2}$. Therefore, the total weight of spanning trees over all heavy components plus W_{excess} will be at most $(p + q + k_b)\lambda - p\frac{\lambda}{2} + (2p + q + k_b - h)\frac{\lambda}{2} = (k_h + k_b) * \frac{3}{2}\lambda - h\frac{\lambda}{2}$. ∎

We know that OPT can be at most $\sum_{e\in E} w(e)$. By Theorem 3 we know that if $\lambda \ge$ OPT, Algorithm 1 will find a k-tree cover with maximum tree weight at most 3λ. If $\lambda <$ OPT the algorithm may fail or may provide a k-tree cover with maximum weight at most 3λ which is also a true 3-approximation. Now by a binary search in the interval $[0, \sum_{e\in E} w(e)]$, we can find a λ for which our algorithm will give a k-tree cover with bound 3λ and for $\lambda - 1$ the algorithm will fails. Thus, $\lambda =$ OPT which gives us a 3-approximation factor. This completes the proof of Theorem 1.

4 2.5-approximation algorithm for BTC

Given an instance of BTC consisting of a graph G and bound λ on the tree sizes we use OPT to denote an optimum solution and $k =$ OPT denote the number of trees in OPT. As before, we can assume we are working with the shortest-path metric completion graph $\tilde{G} = (V, E)$. Our algorithm for this problem is similar to the algorithm for MMkTC, although the analysis is different. The over all structure of the algorithm is as follows. We delete all the edges with weight greater than $\lambda/4$ in $\tilde{G}$ to obtain graph G'. Let $C_1, \ldots, C_\ell$ be the components of G' whose weight is at most $\lambda/4$, called *light components*, and $C_{\ell+1}, \ldots, C_{\ell+h}$ be the components with weight greater than $\lambda/4$ which we refer to as *heavy components*. We define $A(C_i)$, the tree of a light component C_i plus the weight of attaching it to a heavy component as before: it is the weight of minimum spanning tree of C_i, denoted by $W_T(C_i)$, plus the minimum edge weight that connects a node of C_i to a heavy node if such an edge e exists (in $\tilde{G}$) such that $W_T(C_i) + w(e) \le \lambda$; otherwise $A(C_i)$ is set to infinity. The proof of the following lemma is identical to that of Lemma 3 with $\frac{3}{2}\lambda$ replaced with $\frac{1}{2}\lambda$.

Lemma 7 *Let $L_s = \{C_{l_1}, \ldots, C_{l_s}\}$ be a set of s light-components with bounded $A(C_i)$ values. If $\sum_{1\le i\le s} A(C_{l_i}) + \sum_{\ell+1\le i\le \ell+h} W_T(C_i) \le x - h\frac{\lambda}{4}$, then we can cover all the nodes in the heavy components and in components of L_s with at most $\lfloor \frac{2x}{\lambda} \rfloor$ trees with maximum tree weight no more than λ.*

Before presenting the algorithm we define a graph $H = (L, F)$ formed according to the light components similar to the way we defined it in the MMkTC problem.

Definition 2 *For two given parameters a, b, graph H has $\ell + a + b$ nodes: ℓ (regular) nodes $v_1, \ldots, v_\ell$, where each v_i corresponds to a light component C_i, a dummy nodes called* null *nodes, and b dummy nodes called* heavy *nodes. We add an edge with weight zero between two regular v_i and v_j in H if and only if $i \ne j$ and there is an edge e between C_i and C_j in $\tilde{G}$ such that $W_T(C_i) + W_T(C_j) + w(e) \le \lambda$. Every null node is adjacent to each regular node v_i ($1 \le i \le \ell$) with weight zero. Every regular node $v_i \in H$ whose corresponding light component C_i has finite value of $A(C_i)$ is connected to every heavy node in H with an edge of weight $A(C_i)$. There are no other edges in H.*

Theorem 2 follows from the following theorem.

Theorem 4 *Algorithm BTC (Figure 3) finds a k'-tree cover with maximum tree cost bounded by λ, such that $k' \le 2.5$OPT.*

Proof. It is easy to check that in all the three steps 2(a)v, 2(a)vi, and 2(a)vii the trees found have weight at most λ: since each is either found using Lemma 7 (Step 2(a)v), or is a MST of a light component (Step 2(a)vi), or is the MST of two light components whose total weight plus the shortest edge connecting them is at most λ (Step 2(a)vii). So it remains to show that for some values of a, b, the total number of trees found is at most 2.5OPT.

First note that if matching M found in Step 2(a)iii assigns nodes $v_{l_1}, \ldots, v_{l_b}$ to heavy nodes and has weight W_M then $\sum_{1\le i\le b} A(C_{l_i}) = W_M$. Let W_h denote the total tree weight of heavy components, i.e. $W_h = \sum_{\ell+1\le i\le \ell+h} W_T(C_i)$. Then the number of trees generated using Lemma 7 in Step 2(a)v is at most $\lfloor \frac{2(W_M+W_h+h\lambda/4)}{\lambda} \rfloor$, and the number of trees generated in Steps 2(a)vi and 2(a)iii is exactly $(\ell - b + a)/2$; so we obtain a total of at most $\lfloor \frac{2(W_M+W_h+h\lambda/4)}{\lambda} \rfloor + (\ell - b + a)/2$ trees. We prove the following lemma (see Appendix A for proof):

Lemma 8 *There exist $0 \le a' \le n$ and $0 \le b' \le n$ such that if H is built with a' null nodes and b' heavy nodes then H has a matching M' such that if Algorithm BTC uses M' then each tree generated has weight at most λ and the total number of trees generated will be at most* 2.5OPT.

This lemma is sufficient to complete the proof of theorem as follows. Consider an iteration of the algorithm in which $a = a'$ and $b = b'$. Suppose that the minimum perfect matching that the algorithm finds in this iteration is M with weight W_M. Since $W_M \le W_{M'}$, the total number of trees generated in Step 2(a)v is at most $\lfloor \frac{2(W_M+W_h+h\lambda/4)}{\lambda} \rfloor \le \lfloor \frac{2(W_{M'}+W_h+h\lambda/4)}{\lambda} \rfloor$. Furthermore, the number of trees generated in Steps 2(a)vi and 2(a)vii is exactly $(\ell - b' + a')/2$, so we obtain a total of at most $\lfloor \frac{2(W_M+W_h+h\lambda/4)}{\lambda} \rfloor + (\ell - b + a)/2$ trees. This together with the fact that $W_M \le W_{M'}$ and Lemma 8 shows that we get at most 2.5OPT trees using M. ■

References

[1] Esther M. Arkin, Refael Hassin, and Asaf Levin. Approximations for minimum and min-max vehicle routing problems. *Journal of Algorithms*, 59:1–18, 2006.

[2] Ann Melissa Campbell, Dieter Vandenbussche, and William Hermann. Routing for relief efforts. *Transportation Science*, 42:127–145, May 2008.

[3] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar. Paths, trees, and minimum latency tours. In *Proceedings of the 44th Annual IEEE Symposium on the Foundations of Computer Science*, pages 36– 45, 2003.

[4] C. Chekuri and A. Kumar. maximum coverage problem with group budget constraints. In *Proceedings of APPROX*, 2004.

[5] G. Even, N. Garg, J. Konemann, R. Ravi, and A. Sinha. Covering graphs using trees and stars. *Operations Research Letters*, 32:309–315, 2004. Earlier version in Proceedings of APPROX 2003.

[6] Greg N. Frederickson, Matthew S. Hecht, and Chul E. Kim. Approximation algorithms for some routing problems. *SIAM J. on Computing*, 7:178–193, 1978.

[7] Nili Guttmann-Beck and Refael Hassin. Approximation algorithms for min-max tree partition. *J. Algorithms*, 24:266–286, August 1997.

[8] R. Jothi and B. Raghavachari. Approximating k-traveling repairman problem with repairtimes. *J. of Discrete Algorithms*, 5:293–303, 2007.

[9] Chung-Lun Li, David Simchi-Levi, and Martin Desrochers. On the distance constrained vehicle routing problem. *Oper. Res.*, 40:790–799, July 1992.

[10] Hiroshi Nagamochi. Approximating the minmax rooted-subtree cover problem. *IEICE Transactions on Fundamentals of Electronics*, E88-A:1335–1338, 2005.

[11] Viswanath Nagarajan and R. Ravi. Approximation algorithms for distance constrained vehicle routing problems, 2008.

[12] Zhou Xu and Qi Wen. Approximation hardness of min-max tree covers. *Operations Research Letters*, 38:169–173, 2010.

[13] Zhou Xu and Liang Xu. Approximation algorithms for min-max path cover problems with service handling time. In *Proceedings of the 20th International Symposium on Algorithms and Computation*, ISAAC '09, pages 383–392, Berlin, Heidelberg, 2009. Springer-Verlag.

A Proof of Lemma 8

The rest of this section is to prove this lemma. We use the structure of OPT in order to determine a', b' as well as the matching M'. We do not give an explicit value for a', b', instead we start with $a' = b' = 0$ and we define the edges we add to M' instead. For every two vertices of H that we pair (i.e. every edge we place in M') if that edge involves a null node (or a heavy node) we increase a' (or b') accordingly. In other words, we add a new null node (or heavy node) to H whenever we

need to use a new copy of a null node (or heavy node). At the end, a' will be the total number of null nodes we used in our matching M' and b' will be the number of heavy nodes we used.

We call every tree in OPT an optimum tree. We say an optimum tree T is incident to a component C_i if C_i contains at least one node of T. Note that each optimum tree can be incident to at most 4 components as each edge deleted had weight more than $\lambda/4$. Let F be the set of light components which are incident to only one optimum tree. So each such component contains only one tree or broken subtree of a tree. We add matching edges to M' in 5 steps (described below) and also characterize the optimum trees into types. Initially $M' = \emptyset$, and we start with the optimum trees of first type and match some pairs of nodes in H based on the definition of Type 1 and add them to M'; then in Step 2 we define optimum trees of Type 2 and add all the matching edges that they define into M', and so on. Whenever we need to match a node v_i to a node v_j where v_j is already matched to another node in M' (in one of the previous steps) we use a new null node and match v_i to the null node (instead of v_j).

Step 1: Type 1 trees

An optimum tree is Type 1 if it is incident to only light components, say $C_{x_1}, \ldots, C_{x_p}$ (with $p \leq 4$) which satisfy at least one of the following: i) $p \leq 2$, in which case we match each of v_{x_1} and v_{x_2} (if it is not already matched) to a new null node and add these (at most) two edges to M', or ii)$p = 3$ and at least two of $v_{x_1}, v_{x_2}, v_{x_3}$ are adjacent in H, say v_{x_1}, v_{x_2}, then we add the edge $v_{x_1}v_{x_2}$ to M' and match v_{x_3} with a null node, or iii) $p = 4$ and there are two independent edges among these four vertices in H, say v_{x_1}, v_{x_2} are adjacent and v_{x_3}, v_{x_4} are adjacent, then we add these two edges to M'. So, for each Type 1 optimum tree, we generate at most a total of two trees in Steps 2(a)vi and 2(a)vii (each corresponding to a matching edge described above) that together cover all the vertices of the components $C_{x_1}, \ldots, C_{x_p}$. Note that each of the trees generated this way has weight at most λ by definition of edges of H. In Step 1 we add all possible matching edges to M' by considering all Type 1 optimum trees before going to the next step.

Step 2: Type 2 trees

Every optimum tree T that is not Type 1 and is incident to an even number (specifically 2 or 4) of the light components in F is Type 2. We claim that the vertices of H corresponding to these light components are all adjacent (with edges of weight zero). So we can match them arbitrarily with at most two edges, we add these (at most) two edges to M'. The reason is each of these components contains only the vertices of T (because they are in F, so cannot be incident with any other optimum tree). Therefore all these components belong to the same optimum tree T; so for any two such components, say C_i and C_j, there is a path P connecting two vertices of them in T such that $W_T(C_i) + W_T(C_j) + W(P) \leq \lambda$. Since we are working with the complete graph $\tilde{G}$, there is an edge $e \in \tilde{G}$ between C_i and C_j with $W_T(C_i) + W_T(C_j) + w(e) \leq \lambda$, so v_i, v_j are adjacent in H. In Step 2 we place all the matching edges generated by Type 2 trees into M' before going to the next step.

Step 3: Type 3 trees

Every optimum tree T that is not Type 1 or 2 and has following properties is Type 3: T is incident to an odd number of light components $C_{x_1}, \ldots, C_{x_p}$ (p is specifically 1 or 3) in F and at least one light component C_y not in F such that the broken subtree of T in C_y is connected to the broken subtree of one of C_{x_i}'s, say C_{x_1}, with an edge e_T of T (which is now deleted).

Suppose that T is Type 3, and $p = 3$ (the case that T is incident with only one light component in F is easier and is dealt with below). First note that since each of $C_{x_1}, C_{x_2}, C_{x_3}$ belongs to the same optimum tree (namely T), similar arguments as in case of Type 2, shows that vertices $v_{x_1}, v_{x_2}, v_{x_3}$ are all adjacent in H. Without loss of generality assume e_T connects the broken subtree of T in C_y to the one in C_{x_1}. We claim in that $v_{x_1}v_y$ is an edge in H as well. More specifically

$W_T(C_{x_1}) + W_T(C_y) + w(e_T) \leq \lambda$. The following claim implies this:

Claim 1 $W_T(C_{x_1}) + W_T(C_y) + w(e_T) \leq \lambda$.

Proof. Since T is not Type 1, C_{x_1} and C_y cannot be the only components to which T is incident (otherwise, each of C_{x_1} and C_y would be matched to null nodes as in Type 1). Therefore, there is at least one other component that has a broken subtree of T, and there is at least one other edge of T, call e', (which is deleted now) connecting that broken subtree to C_{x_1} or to C_y in $\tilde{G}$. Note that $w(e') > \lambda/4$ and $W_T(C_y) \leq \lambda/4$. Therefore, $W_T(C_{x_1}) + W_T(C_y) + w(e_T) \leq W_T(C_{x_1}) + w(e') + w(e_T) \leq \lambda$ since all these are parts of T. ■

Hence we can pair v_{x_1} with v_y and pair v_{x_2} with v_{x_3} and add these two edges to M'. Note that again the tree generated by each of these pairs has weight at most λ. If T is Type 3 and is incident to only one light component in F, say C_{x_1} then we pair v_{x_1} with v_y as above and add only one edge to M'. We consider all Type 3 optimum trees and add the corresponding matching edges to M' before going to consider the next step..

Step 4: Type 4 trees

Every optimum tree T that is not Type 1, 2, or 3 and has following properties is Type 4: T is incident to an odd number of light components $C_{x_1}, \ldots, C_{x_p}$ (p is specifically 1 or 3) in F and at least one heavy component C_y such that the broken subtree of T in C_y is connected to the broken subtree in one of C_{x_i}s with an edge e_T of T (which is now deleted).

Suppose that optimum tree T is Type 4 and $p = 3$ (again the case that T is incident with only one light component in F is easier). Arguments similar to the case of Type 3 show that vertices $v_{x_1}, v_{x_2}, v_{x_3}$ (corresponding to $C_{x_1}, C_{x_2}, C_{x_3}$) are all adjacent in H. Without loss of generality let assume e_T connects the broken subtree of T in C_y to the one in C_{x_1}. In this case, the weight of the broken subtree of T in C_{x_1}, plus the weight of e_T is no more than λ (as they are all part of T); in particular $W_T(C_{x_1}) + w(e_T) \leq \lambda$. This implies $A(C_{x_1}) \leq \lambda$ and so v_{x_1} is adjacent to heavy nodes in H. In this case we pair v_{x_1} with a (not already matched) heavy node in H and pair v_{x_2} with v_{x_3} and add these two edges to M'. Note that as argued before, the tree generated by the matching edge v_{x_2}, v_{x_3} has weight at most λ. Also, since we use Lemma 7 for each heavy component together with the light components attached to it, the weight of each tree generated from the heavy components (and their assigned light components) is at most λ.

Step 5: The rest of light components

This step completes the description of M'. Before starting this step, we explain why all the vertices in H corresponding to components in F are saturated by M' before this step. We show that if T has at least one broken subtree in a component in F then T is either Type 1, 2, 3, or 4, therefore all the components in F containing a broken subtree of T are matched in M'. We consider the following three cases for T: (1) If T is incident at only components in F then it is Type 1, (2) If T is incident at even number of components in F then it is Type 2, (3) If T is incident at odd number of components in F then, as it is not Type 1, T is incident at some other components not in F. As T is connected, there is an edge e_T which connects a broken subtree of T in a component in F to a broken subtree of T to a component C_y not in F. If C_y is a light component then T is Type 3 and if C_y is a heavy component then T is Type 4. So, all vertices corresponding to light components in F are already matched. In Step 5, if there is any light component that is not matched so far, each of them is incident with at least two optimum trees. In this step we match the corresponding vertex (in H) of each of these light components to a null node and these edges are added to M'.

Now we prove that the total number of trees generated by matching M' is at most 2.5OPT. Let N_1 denote the number of matching edges added to M' in Step 1, Y denote the number of matching edges added to M' in Steps 2 to 5 that does not involve a heavy node, and N_4 denote

the number of trees generated by applying Lemma 7 to the matching edges added to M' in Step 4 that involves a heavy node. Note that the total number of trees generated in the final solution based on matching M' is $N_1 + Y + N_4$. We use OPT_1 to denote the number of optimum trees of Type 1, and $\text{OPT}_{rest} = \text{OPT} - \text{OPT}_1$ to denote the number of other optimum trees. Our goal is to show $N_1 + Y + N_4 \leq 2.5\text{OPT}$, more specifically we show: $N_1 + Y + N_4 \leq 2\text{OPT}_1 + 2.5\text{OPT}_{rest}$. It is easy to see that for every optimum tree of Type 1, we add at most 2 edges to M' in Step 1 (and therefore at most 2 trees in the final solution). Therefore, $N_1 \leq 2\text{OPT}_1$. In the rest we show that $Y + N_4 \leq 2.5\text{OPT}_{rest}$. We prove the following claim.

Claim 2 *Suppose we add Y edges to M' in Steps 2 to 5 that do not involve a heavy node and have matched light components $C_{l_1}, \ldots, C_{l_s}$ to heavy nodes in Step 4. Then:*

(i) The union of the Y trees that are generated in the final solution based on the matching edges added to M' in Steps 2 to 5 that do not involve a heavy node contains at least $2Y$ broken subtrees of the optimum trees that are not Type 1.

(ii) $\sum_{i=1}^{s} A(C_{l_i}) + \sum_{\ell+1 \leq i \leq \ell+h} W_T(C_i) \leq \frac{5}{4}\lambda \cdot \text{OPT}_{rest} - \frac{\lambda}{2} \cdot Y - h\frac{\lambda}{4}$.

First, let us show how we can complete the proof of lemma using this claim. Using Lemma 7 and part (ii) of Claim 2, the total number of trees generated based on matching edges added to M' at Step 4 that involves a heavy node is bounded by: $N_4 \leq \lfloor 2.5\text{OPT}_{rest} - Y \rfloor$. Also, the total number of matching edges that do not involve a heavy node (and therefore the corresponding number of trees in the final solution) generated at Steps 2 to 5 is Y. So we get $Y + N_4 \leq \lfloor 2.5\text{OPT}_{rest} - Y \rfloor + Y \leq 2.5\text{OPT}_{rest}$, as wanted. Now it only remains to prove Claim 2.

Proof of Claim 2:

Part (i): We show that for every tree generated in the final solution based on matching edges added to M' in Steps 2 to 5 that do not involve a heavy node, there are two distinct broken subtrees of the optimum trees. To show this, we assign two broken subtrees for every such tree generated in the final solution such that each broken subtree is assigned to at most one tree of final solution.

For every optimum tree T of Type 2, each edge e added to matching M' in Step 2 is between two components in F each of which contains exactly one broken subtree of T; therefore the corresponding tree in the final solution generated based on e contains the broken parts of T in those two components of F. We assign those two broken subtrees to the tree generated. Also, every light component that is considered in Step 5 is not in F, i.e. it is incident with at least two optimum trees and so has at least two broken subtrees of two different optimum trees (that are not Type 1). Therefore, the tree generated at the final solution for each light component in Step 5 contains at least two broken subtrees of optimum trees that are not Type 1, we assign those two broken subtrees to the tree generated. Now we consider the matching edges added in Step 3 and 4 that do not involve a heavy node. If the matching edge $e \in M'$ corresponds to two components in F then similar to the case of Step 2, the tree generated in the final solution based on e contains the broken subtrees defined by the two components in F; we assign those two parts to the tree generated based on e. So the only remaining case is when we have a Type 3 tree T and it has a broken subtree in a component in F, say C_{x_1}, and another broken subtree in a light component not in F, say C_y (in Step 3). Note that C_{x_1} (since is in F) by definition has only one broken subtree and that is of tree T. Also, C_y has at least one broken subtree of T even if vertex $v_y \in H$ (corresponding to C_y) was matched to a different vertex (because C_y also had a broken subtree for a different optimum tree T'). So regardless of whether v_{x_1} is matched to v_y or to a null node, we can assign the two broken subtrees of T (one in C_{x_1} and one in C_y) to the tree generated based this matching edge in M'.

part (ii): To prove this part, suppose T is a Type 4 optimum tree as in Step 4 which has two broken subtrees, one subtree in the light component $C_{x_1} \in F$, denote it by T_{x_1} (note that there is no other subtree in C_{x_1}), and one subtree in the heavy component C_y, denote it by T_y, and there is an edge $e_T \in T$ that connects a node of T_{x_1} to a node of T_y. Recall that e_T was deleted as $W(e_T) > \lambda/4$. For the purpose of analysis, we merge component C_{x_1} (which consists of the vertices of T_{x_1}) with the heavy component C_y by adding the edge e_T back; this will merge the two broken subtrees T_{x_1} and T_y into one subtree. Also, by doing this, the weight of a MST in the new heavy component increases by $A(C_{x_1})$ only. If there are multiple optimum trees of Type 4 which have a broken subtree in C_y we merge them all with C_y. More generally, we do this merge operation for all the light trees $C_{l_1}, \ldots, C_{l_s}$ that are matched with heavy nodes in Step 4 and we let $C'_{\ell+1}, \ldots, C'_{\ell+h}$ be the set of new modified heavy components after these merge operations. Note that:

$$\sum_{i=1}^{s} A(C_{l_i}) + \sum_{\ell+1 \le i \le \ell+h} W_T(C_i) = \sum_{\ell+1 \le i \le \ell+h} W_T(C'_i).$$

Now we prove that $\sum_{\ell+1 \le i \le \ell+h} W_T(C'_i) \le \frac{5}{4}\lambda \cdot \text{OPT}_{rest} - \frac{\lambda}{2} \cdot Y - h\frac{\lambda}{4}$. Let p denote the number of (new) broken subtrees of the OPT_{rest} optimum trees that are not Type 1; Using part (i), at least $2Y$ of these parts are covered by (i.e. are contained in) the Y trees generated using the matching edges of Steps 2 to 5 that do not involve a heavy node. Therefore, the remaining at most $p - 2Y$ broken subtrees are in the modified heavy components. The total weight of optimum trees that are not Type 1 is at most $\lambda \cdot \text{OPT}_{rest}$. Out of these trees at least $p - \text{OPT}_{rest}$ edges are deleted even after merge operations that built modified heavy components, since we have a total of p broken subtrees. Therefore, the total weight of these p broken subtrees is at most $\lambda \cdot \text{OPT}_{rest} - \frac{\lambda}{4} \cdot (p - \text{OPT}_{rest})$ and all of these are inside the modified heavy components. By an argument similar to that of proof of Lemma 1, to make a spanning tree in each modified heavy component, the total weight of edges that need to be added between the broken subtrees inside the heavy components is at most $(p - 2Y - h)\frac{\lambda}{4}$. Therefore, the total weight of (MST's of) the modified heavy components is bounded by:

$$\lambda \cdot \text{OPT}_{rest} - \frac{\lambda}{4} \cdot (p - \text{OPT}_{rest}) + (p - 2Y - h)\frac{\lambda}{4} \le \frac{5}{4}\lambda \cdot \text{OPT}_{rest} - \frac{\lambda}{2} \cdot \lambda - h \cdot \frac{\lambda}{4}.$$

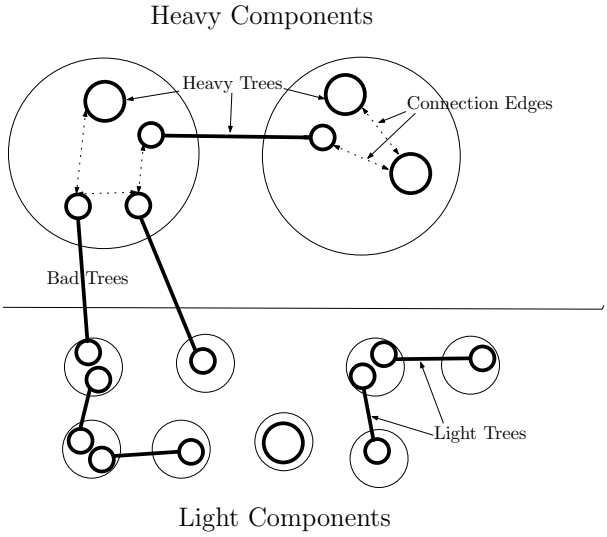

Figure 2: Structure of G after deleting edges with length greater than $\frac{\lambda}{2}$. Each thin circle corresponds to a component and each solid circle corresponds to an optimum tree or a broken subtree (part) of an optimum tree.

Inputs $G(V, E)$, λ
Output: A set s containing k'-tree cover with maximum tree cost λ in which $k' \leq 2.5\text{OPT}$.

1. Take $\tilde{G}$ to be the metric completion of G and delete edges with length more than $\frac{\lambda}{4}$ to form graph G' with components $C_1, \ldots, C_{\ell+h}$
2. For $a : 0 \to \ell$
   (a) For $b : 0 \to \ell$
      i. $S_{a,b} \leftarrow \emptyset$
      ii. Build graph H according to Definition 2 with a null nodes and b heavy nodes.
      iii. Find a perfect matching with the minimum cost in H, if there is no such perfect matching continue from Step 2a
      iv. Attach each light component C_i to its nearest heavy component if v_i is matched to a heavy node
      v. Decompose all the heavy components and the attached light components as explained in Lemma 7 and add the trees obtained to $S_{a,b}$
      vi. If a vertex v_i is matched to a null node, add MST of C_i to $S_{a,b}$.
      vii. For every matching edge between v_i and v_j consider the cheapest edge e between C_i and C_j (in $\tilde{G}$) and add a minimum spanning trees of $C_i \cup C_j \cup \{e\}$ to $S_{a,b}$.
3. return set $S_{a,b}$ with the minimum number of trees.

Figure 3: BTC Algorithm

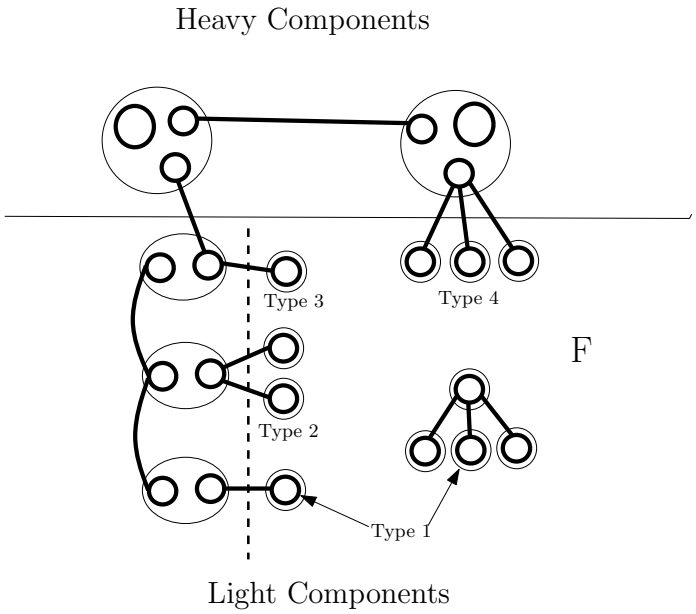

Figure 4: Each thin circle shows a component and each solide circle shows a broken subtree of an optimum tree; the solid lines show bridge edges that are deleted and were connecting broken subtrees of optimum trees