A Fast 3-Approximation for the Capacitated Tree Cover Problem with Edge Loads

Benjamin Rockel-Wolff
Research Institute for Discrete Mathematics, University of Bonn, Germany

Abstract

The capacitated tree cover problem with edge loads is a variant of the tree cover problem, where we are given facility opening costs, edge costs and loads, as well as vertex loads. We try to find a tree cover of minimum cost such that the total edge and vertex load of each tree does not exceed a given bound. We present an $\mathcal{O}(m \log n)$ time 3-approximation algorithm for this problem.

This is achieved by starting with a certain LP formulation. We give a combinatorial algorithm that solves the LP optimally in time $\mathcal{O}(m \log n)$. Then, we show that a linear time rounding and splitting technique leads to an integral solution that costs at most 3 times as much as the LP solution. Finally, we prove that the integrality gap of the LP is 3, which shows that we can not improve the rounding step in general.

2012 ACM Subject Classification Mathematics of computing → Approximation algorithms; Mathematics of computing → Trees

Keywords and phrases Approximation Algorithms, Tree Cover, LP

Digital Object Identifier 10.4230/LIPIcs.SWAT.2024.39

Related Version *Full Version*: `https://arxiv.org/abs/2404.10638` [11]

1 Introduction

Graph cover problems deal with the following base problem. Given a graph G, the task is to find a set of (connected) subgraphs of G, the cover, such that each vertex of G is contained in at least one of the subgraphs. Usually, the subgraphs are restricted to some class of graphs, like paths, cycles or trees. Different restrictions can be imposed on the subgraphs, like a maximum number of edges, or a total weight of the nodes for some given node weights. Recently, Schwartz [12] published an overview of the literature on different covering and partitioning problems.

We consider the capacitated tree cover problem with edge loads. It is a variation of the tree cover problem that has not been studied so far to the best of our knowledge.

In the capacitated tree cover problem with edge loads, we are given a complete graph $G = (V, E)$, metric edge costs $c : E \to \mathbb{R}_+$, vertex loads $b : V \to [0, 1)$, metric edge loads $u : E \to \mathbb{R}_{\geq 0}$ with $u(e) < u(f) \Rightarrow c(e) \leq c(f)$, and a facility opening cost $\gamma \geq 0$. The task is to find a number of components $k \in \mathbb{N}_{\geq 1}$ and a forest F in G consisting of k trees minimizing

$$\sum_{e \in E(F)} c(e) + \gamma k,$$

such that each tree T_i has total load

$$u(T_i) := \sum_{e \in E(T_i)} u(e) + \sum_{v \in V(T_i)} b(v) \leq 1.$$

For simplicity of presentation, we additionally require that $u > 0$. This is not necessary in general and the extended proofs for $u \geq 0$ are covered in the full paper [11].

The capacitated tree cover problem with edge loads is closely related to the facility location problem with service capacities discussed by Maßberg and Vygen in [10]. Their problem uses Steiner trees to connect the nodes, not spanning trees. Furthermore, in their case edge cost and edge load are the same. They make use of this fact to prove a lower bound on the value of an optimum solution. Both problems have important practical applications in chip design. In [4] they are called the sink clustering problem and used for clock tree construction. In [2] they are used for repeater tree construction. In these applications terminals and edges have an electrical capacitance. A source can drive only a limited capacitance. Edge cost and capacitance usually are proportional to the length of an edge. As the edge length is given by the l_1-distance between its endpoints, this naturally matches our problem.

Our problem is also related to other facility location and clustering problems, like the (capacitated) k-center problem ([5, 8]) or the k-means problem ([6, 9]).

Other tree cover problems include the k-min-max tree cover problem and the bounded tree cover problem ([1, 3, 7]). In the k-min-max tree cover problem, we are given edge weights and want to find k trees such that the maximum of the total weights of the trees is minimized. In the bounded tree cover problem, we are given a bound on the maximum weight of a tree in the cover and try to minimize the number of trees that are required. For these problems Khani and Salavatipour [7] gave a 3- and 2.5-approximation respectively. They improve over the previously best known results by Arkin et al. [1], who presented a 4-approximation algorithm for the min-max tree cover problem and a 3-approximation algorithm for the bounded tree cover problem. Even et al. [3] independently gave a 4-approximation algorithm for the min-max tree cover problem. Furthermore, a rooted version of these problems has been studied. The best known approximation ratio for the capacitated tree case is 7 and was developed by Yu and Liu [15].

Many algorithms for cycle cover problems are also based on tree cover algorithms ([3, 13, 14]). An example is the capacitated cycle covering problem, where the cover consists of cycles (and singletons) and are given an upper bound on the total nodeweight of the cycles. The task is to minimize the total weight of the cycles plus the facility opening costs. Traub and Tröbst [13] presented a $2+\frac{2}{7}$-approximation for this problem. They use an algorithm for the capacitated tree cover problem as a basis for their $2+\frac{2}{7}$-approximation. In particular, they present a 2-approximation for the capacitated tree cover problem without edge loads.

2 Our contribution

In Section 3, we present an LP formulation of the capacitated tree cover problem with edge loads that is based on the formulation in [13].

Then, we will present a combinatorial algorithm that can optimally solve the LP in time $\mathcal{O}(m \log n)$ in Section 4, where n is the number of vertices and m is the number of edges of the graph.

Next, we show how to round the solution to an integral solution in Section 5, employing a splitting technique that runs in linear time from [10], and show that the resulting integral solution costs at most 3 times as much as the LP-solution. This proves our main theorem:

▶ **Theorem 1.** *There is a 3-approximation algorithm for the capacitated tree cover problem with edge loads that runs in time $\mathcal{O}(m \log n)$.*

While the overall approach is similar to the one used in [13], edges with load require a different algorithm for solving the LP. Furthermore, we need to be more careful in the analysis of our rounding step.

Finally, in Section 6, we will give an example proving that the integrality gap of our LP is at least 3.

3 The LP-formulation

We may assume that $\gamma \geq c(e)$ for all $e \in E$, as an edge with $c(e) > \gamma$ will never be used in an optimum solution (and could be removed from the solution of the algorithm without increasing the cost).

For simplicity, we will introduce some notation here: For any function $f : A \to B \subseteq \mathbb{R}$ from a finite set A into a set $B \subseteq \mathbb{R}$ and $X \subseteq A$ we write $f(X) := \sum_{x \in X} f(x)$.

Given a solution F to our problem with k components $\{T_1, \dots, T_k\}$, we know that each tree T_i contains exactly $|V(T_i)| - 1$ edges and hence $k = |V| - |E(F)|$. Each induced subgraph of F is a forest. So we know

$$|E(F[A])| \leq |A| - 1 \text{ for each } A \subseteq V.$$

Let us now consider the load on the subgraph of F, induced by $A \subseteq V$. Each connected component in $F[A]$ can have load at most 1. So there must be at least $b(A) + u(E(F[A]))$ components in $F[A]$. As each of the components is a tree, the inequality

$$|E(F[A])| \leq |A| - (b(A) + u(E(F[A])))$$

must be fulfilled. Using these properties, we can formulate the following LP relaxation of this problem:

$$\begin{aligned}
\min \quad & c^t x + \gamma(|V| - x(E)) && && (1)\\
\text{s.t.} \quad & x(E(G[A])) \leq |A| - 1 && \text{for each } A \subseteq V && (2)\\
& \sum_{e \in E(G[A])} (1 + u(e))x(e) \leq |A| - b(A) && \text{for each } A \subseteq V && (3)\\
& 0 \leq x(e) \leq 1 && \text{for each } e \in E && (4)
\end{aligned}$$

Here $x(e)$ denotes the fractional usage of the edge e. We will call an edge e *active* if $x(e) > 0$. The LP can be reformulated by using variables $y(e) := x(e)(1 + u(e))$:

$$\begin{aligned}
\min \quad & \sum_{e \in E} \frac{c(e)}{1 + u(e)} y(e) + \gamma \left(|V| - \sum_{e \in E} \frac{y(e)}{1 + u(e)} \right) && && (5)\\
\text{s.t.} \quad & \sum_{e \in E(A)} \frac{y(e)}{1 + u(e)} \leq |A| - 1 && \text{for each } A \subseteq V && (6)\\
& y(E(G[A])) \leq |A| - b(A) && \text{for each } A \subseteq V && (7)\\
& 0 \leq y(e) \leq 1 + u(e) && \text{for each } e \in E && (8)
\end{aligned}$$

For simplicity, we will always consider solutions x, y of both LPs at once. In the following, we will denote by $u_x(e) := x(e) \cdot u(e)$ the fractional load of edge e.

▶ **Definition 2.** *For a solution x, y to the LP, we define the* support graph $G_x := (V, \{e \in E \mid x(e) > 0\})$*, i.e. the graph consisting of the vertices V and all active edges.*

We call an edge tight *if $y(e) = 1 + u(e)$, and we call a set $A \subseteq V$ of vertices* tight *if inequality (7) is tight.*

Our goal will be to solve the LP exactly and then round to a forest that may violate the capacity constraints. This increases the edge cost by at most a factor of 2. In a final step each tree T in the forest with a load $b(V(T)) + u(E(T)) > 1$ can be split into at most $2 \cdot (b(V(T)) + u(E(T)))$ trees. This may decrease the edge cost, but loses a factor of 3 in the number of components, compared to the LP solution.

4 Solving the LP

Altough the LP has an exponential number of inequalities, we can solve it using a simple greedy algorithm, shown in Algorithm 1. We will focus on solving the second LP (5) – (8).

As a first step, we sort the edges $\{e_1, \dots, e_m\} = E(G)$ such that

$$\frac{c(e_1) - \gamma}{1 + u(e_1)} \leq \dots \leq \frac{c(e_m) - \gamma}{1 + u(e_m)}.$$

In each iteration, we compute a partition $\mathcal{A}_i \subset 2^{V(G)}$ of the vertices of G, based on the previous partition $\mathcal{A}_{i-1}$. We initialize y to 0 and start with $\mathcal{A}_0 := \{\{v\} | v \in V(G)\}$. Then we iterate through the edges from e_1 to e_m. For each edge e_i, we do the following:

If e_i has endpoints in two different sets of the partition $A_i^1, A_i^2 \in \mathcal{A}_{i-1}$, we increase $y(e_i)$ to the maximum possible value. This maximum value is the sum of the slacks of inequalities (7) for the sets A_i^1 and A_i^2: $|A_i^1| - b(A_i^1) - y(E(G[A_i^1])) + |A_i^2| - b(A_i^2) - y(E(G[A_i^2]))$. However, we assign at most $1 + u(e_i)$, such that we do not violate inequality (8). Finally, if we increased $y(e_i)$ by a positive amount, we create the new partition $\mathcal{A}_i$ that arises from $\mathcal{A}_{i-1}$ by removing A_i^1 and A_i^2 and adding their union.

We set $\mathcal{A} := \bigcup_{i=1,\dots,m} \mathcal{A}_i$. Observe that $\mathcal{A}$ is a laminar family. This guarantees that the support graph is always a forest and inequality (6) is automatically fulfilled.

Algorithm 1 Algorithm for solving the LP (5) – (8).

Input : G, c, u.
Output : y optimum solution of the LP (5) – (8).

```
Sort edges such that c(e_1)−γ/(1+u(e_1)) ≤ ... ≤ c(e_m)−γ/(1+u(e_m));
Set A_0 := {{v}|v ∈ V(G)} and y := 0;
for i = 1...m do
    if there are sets A_i^1, A_i^2 ∈ A_{i−1} with e_i ∩ A_i^1 ≠ ∅, e_i ∩ A_i^2 ≠ ∅ and A_i^1 ≠ A_i^2 then
        y(e_i) := min{1 + u(e_i), |A_i^1| − b(A_i^1) − y(E(A_i^1)) + |A_i^2| − b(A_i^2) − y(E(A_i^2))};
        if y(e_i) > 0 then
            A_i := (A_{i−1} \ {A_i^1, A_i^2}) ∪ {A_i^1 ∪ A_i^2};
        else
            A_i := A_{i−1}
```

▶ **Lemma 3.** *Let x, y be the solution computed by Algorithm 1. If a set $A \in \mathcal{A}$ from the algorithm is not tight, then all the edges in its induced subgraph $G_x[A]$ of the support graph are tight.*

Proof. Assume this were false. Take a minimal counterexample A. As the claim certainly holds for sets consisting only of one vertex ($G_x[A]$ has no edges if $|A| = 1$), we know that $|A| \geq 2$. We can write $A = A_i^1 \cup A_i^2$ with their associated edge e_i (for some i). We know

that e_i has to be tight by line 5, as A is not tight. Otherwise, the algorithm could have increased $y(e_i)$ further. At least one of the subsets A_i^1 and A_i^2 of A is not tight, otherwise, A were tight. W.l.o.g we may assume that A_i^1 is not tight. Then all of its edges are tight, by minimality of A. However, then we know that $x(E(G[A_i^1])) = |A_i^1| - 1$. Thus,

$$\begin{aligned}&|A_i^1| - b(A_i^1) - y(E(G[A_i^1])) = |A_i^1| - b(A_i^1) - x(E(G[A_i^1])) - u_x(E(G[A_i^1]))\\ &= |A_i^1| - b(A_i^1) - (|A_i^1| - 1) - u_x(E(G[A_i^1])) = 1 - (u_x(E(G[A_i^1]) + b(A_i^1)) < 1 + u(e_i).\end{aligned}$$

This implies that A_i^1 does not have enough slack to make e_i tight. Thus A_i^2 cannot be tight. As A contains an edge that is not tight in its support graph, this edge must be contained in A_i^2. We can conclude that A_i^2 is a smaller counterexample. This contradicts the minimality of A. ◀

▶ **Corollary 4.** *Let $e_i \in E$, A_i^1 and A_i^2 fulfill the conditions in line 4 of Algorithm 1. If e_i is tight then neither A_i^1, nor A_i^2 are tight.*

▶ **Theorem 5.** *Algorithm 1 works correctly and has running time $\mathcal{O}(m \log n)$.*

Proof. The running time is dominated by sorting. Due to space constraints, we will only give the ideas of the correctness proof. The details are contained in the full version [11] of this paper.

We first check that the algorithm outputs a solution to our LP. The minimum in line 5 guarantees that inequality (8) is fulfilled. We have already seen that the support graph of our solution is a forest, which means that inequality (6) is also satisfied. It remains to check that inequality (7) holds. Each $A \in \mathcal{A}$ fulfills the inequality, when it is introduced by line 5. Since $\mathcal{A}$ is a laminar family, we never change the value of $y(E(G[A]))$ after A has been introduced, so we already know that all $A \in \mathcal{A}$ satisfy inequality (7) when the algorithm is finished.

We define the slack of a set $A \subseteq V$ as the slack of inequality 7 for that set and denote it by $\sigma(A) := |A| - b(A) - y(E(G[A]))$.

Then we can prove that when the algorithm introduces a new set A, it has no more slack than each of the joined subsets.

▷ Claim 6. Let $A_i^1, A_i^2, A \in \mathcal{A}$ such that $A = A_i^1 \cup A_i^2$. We claim that $\sigma(A) \leq \sigma(A_i^j)$ for $j = 1, 2$.

The idea to prove this, is to show that $\sigma(A) = \sigma(A_i^1) + \sigma(A_i^2) - y(e_i)$. Then we use Corollary 4 and Lemma 3 to derive that $\max\{\sigma(A_i^1), \sigma(A_i^2)\} \leq y(e_i)$, which proves the claim. As a next step, we extend Claim 6 to all subsets of A.

▷ Claim 7. Let $A \in \mathcal{A}$. We claim that each subset $B \subseteq A$ has slack $\sigma(B) \geq \sigma(A)$.

We prove this by induction on the number of iterations. The main idea here is to first split B into subsets B_1, B_2 of the two sets A_i^1, A_i^2 that have been merged to form A. We show that $\sigma(B) \geq \sigma(B_1) + \sigma(B_2) - y(e_i)$. Then we apply our induction hypothesis to both sets and use the equality from Claim 6 to prove Claim 7.

Finally, we observe that for $B_1 \subseteq V$ and $B_2 \subseteq V$ from different connected components of G_x, we have $\sigma(B_1 \cup B_2) = \sigma(B_1) + \sigma(B_2)$. This means that we can split any subset of the vertices $B \subseteq V$ into subsets of the toplevel sets of $\mathcal{A}$, which are exactly the connected components of G_x. Then we can use Claim 7 on each of the subsets to see that they have nonnegative slack. The observation implies that also B has nonnegative slack. So inequality 7 is always satisfied.

Next we want to prove optimality. Assume that y were not optimum. Let y^* be an optimum solution that fulfills the following: It maximizes the index of the first edge in the order of the algorithm in which y and y^* differ. Among those it minimizes the difference in this edge. Let this index be denoted by k. As the algorithm always sets the values to the maximum that is possible without violating an inequality, we know that $y^*(e_k) < y(e_k)$.

By the ordering of the algorithm, we know that

$$\frac{c(e_k) - \gamma}{1 + u(e_k)} \leq \frac{c(e_i) - \gamma}{1 + u(e_i)}$$

for all $i > k$. Our goal will be, to find an edge e_i with $i > k$ such that we can increase $y^*(e_k)$ and avoid violating constraints or increasing the objective by decreasing $y^*(e_i)$ in x^*, y^*.

Let G_k be the connected component of e_k in the subgraph of G that contains only e_k and the active edges with index less than k. Define

$$\Gamma := \{e_i \in E(G_{x^*}) \mid i > k \text{ and } e_i \text{ incident to } v \in V(G_k)\}.$$

Note that $\Gamma \neq \emptyset$, because otherwise, we could increase $y^*(e_k)$ to $y(e_k)$ without violating any constraints. Since $c(e) \leq \gamma$, this would not increase the objective value.

We will prove that all tight sets containing the vertices of G_k must have a common edge in Γ.

▷ Claim 8. Let $\mathcal{T} := \{B \subseteq V \mid V(G_k) \subseteq B \text{ and } B \text{ tight}\}$ be the family of tight sets of x^*, y^* containing the vertices of G_k. We claim that

$$\Gamma \cap \bigcap_{B \in \mathcal{T}} E(G_{x^*}[B]) \neq \emptyset.$$

If there is an edge in Γ between vertices of G_k, then this certainly holds. Otherwise, we know that $V(G_k)$ is not tight, because the algorithm was able to set $y(e_k) > y^*(e_k)$.

Let $\mathcal{S} := \{S_1, \ldots, S_p\} \subseteq \mathcal{T}$ be a set of $p \geq 2$ tight sets containing the vertices of G_k and set $Z := \bigcup_{S_i \in \mathcal{S}} S_i$. First, we observe that Z is tight as well. Then, we will prove our claim by induction on p. The key idea is to use the tightness of Z and the S_i to decompose $y^*(E(G[Z]))$ into an alternating sum of y^* on intersections of the S_i and then reassemble $y^*(E(G[Z]))$ from the parts (see [11]). Then, we see that there must be slack on the cut defined by G_k in the intersection of all S_i. Thus there must be an edge in the cut, which proves our claim.

Finally, we can pick an edge $f \in \Gamma$ that is contained in all tight sets that contain the vertices of G_k. If $u(f) \leq u(e_k)$, we know that $\frac{1}{1+u(f)} \geq \frac{1}{1+u(e_k)}$. So we can decrease $y^*(f)$ and increase $y^*(e_k)$ by the same amount without violating any constraints. By the ordering of our algorithm, this can not increase the objective value. This would contradict our choice of y^*. Hence $u(f) > u(e_k)$. But then $c(f) \geq c(e_k)$ and we could decrease $x^*(f)$ and increase $x^*(e_k)$ without increasing the objective value. Furthermore, we also do not create a violation this way, because $\epsilon \cdot (1 + u(f)) > \epsilon \cdot (1 + u(e_k))$ for $\epsilon > 0$. This contradicts our choice of y^* and concludes the proof. ◀

The support graph of the LP solution computed by Algorithm 1 is always a forest. Thus, Theorem 5 implies the following:

▶ **Corollary 9.** *There is always a solution x, y to both LPs, such that the support graph G_x is a forest.*

5 The Rounding Strategy

Now we want to round the LP solution, computed by Algorithm 1, to get an integral solution. We do so by rounding up edges e with $x(e) \geq \alpha$, for some $0 \leq \alpha \leq 1$ to be determined later. All other edges are rounded down. The forest arising from this rounding step may contain components T with $b(V(T)) + u(E(T)) > 1$. These large components will be split into at most $2 \cdot (b(V(T)) + u(E(T)))$ legal components. We achieve this by using a splitting technique that is often used for these cases, for example in [10] and also in [13]. During splitting, we need to shortcut some paths. This is possible, because G is a complete graph and u and c are metric.

The splitting technique traverses the trees in a bottom up fashion (for an arbitrary root). At each node, it approximately solves a bin packing problem to split off components that are too heavy. However, for the analysis, we only require the result that it is possible to split the trees into $2 \cdot (b(V(T)) + u(E(T)))$ legal components.

▶ **Lemma 10** (Maßberg and Vygen 2008 [10]). *If G is a complete graph and u and c are metric, there is a linear time algorithm that splits a tree with total load $b(V(T))+u(E(T)) > 1$ into at most $2 \cdot (b(V(T)) + u(E(T)))$ legal trees and does not increase the total edge cost.*

In Section 5.1, we will study the LP solution, that we get from Algorithm 1. We will exploit the structure of this solution in our analysis. Then we will bound the number of components that we get after rounding and splitting in Section 5. We do this by providing an upper bound on the value of each edge after rounding and splitting. Finally, in Section 5.2.4, we show that $\alpha := \frac{2}{3}$ will lead to an approximation factor of 3 independent of the edge loads.

5.1 The general structure of the LP solution

Let x, y be a solution found by the algorithm. Recall that for edges $e \in E(G)$, $u_x(e) := x(e) \cdot u(e)$ was the fractional load of the edge e in our solution. Note that then it holds for each set $A \subseteq V(G)$ and edge $e \in E(G)$ that

$$y(E(G[A])) = x(E(G[A])) + u_x(E(G[A])) \text{ and } y(e) = x(e) + x(e)u(e).$$

Without loss of generality, we can assume that $0 < x(e) < 1$ for all edges and G_x does not contain singletons. We simply remove all edges with $x(e) = 0$. Then we contract all inclusionwise maximal sets $A \in \mathcal{A}$ such that all edges in their respective induced support graph are tight and set the load of the new vertex to $b(A) + u_x(E(G[A]))$. Corollary 4 implies that we contracted all tight edges this way. These operations only make the approximation guarantee worse, because these components will have the same value in the rounded solution as in the LP-solution. In the remaining graph the following assertions hold:

1. $|\{v\}| - b(\{v\}) - y(E(G[\{v\}])) = 1 - b(v) \leq 1$ for all $v \in V$.
2. All $A \in \mathcal{A}$ containing more than 1 vertex are tight, by Lemma 3.

Now, we will take a closer look at the sets A_i^j for $i = 1, \ldots, m$ and $j = 1, 2$. In the following analysis, we will assume without loss of generality that $|A_i^1| \leq |A_i^2|$. By the above assertions, we have for an edge e_i and the two associated sets A_i^1, A_i^2, either

(i) both A_i^1 and A_i^2 contain only one vertex and one of them is not tight, or
(ii) A_i^1 contains only one vertex and is not tight and A_i^2 contains more vertices and is tight

To make the following easier to read, we add the following definitions

▶ **Definition 11.** *Edges that fulfill condition (i) are called* seed edges *and edges that fulfill condition (ii) are called* extension edges. *For each edge e_i we denote by v_{e_i} the unique vertex in set A_i^1.*

Note that every edge e_i is either a seed edge or an extension edge, but this only holds after contracting the sets of tight edges as described above.

Thus, whenever e_i is a seed edge, the algorithm sets

$$y(e_i) := |A_i^1| - b(A_i^1) - y(E(G[A_i^1])) + |A_i^2| - b(A_i^2) - y(E(G[A_i^2])) = 1 - b(A_i^1) + 1 - b(A_i^2),$$

where we use the fact that $E(G[A_i^j]) = \emptyset$ for $j = 1, 2$. Since both A_i^1 and A_i^2 were singletons, we can conclude

$$x(e_i) + u(e_i)x(e_i) = y(e_i) = 2 - b(A_i^1 \cup A_i^2).$$

Similarly, for extension edges, we get

$$x(e_i) + u(e_i)x(e_i) = 1 - b(A_i^1).$$

In the analysis of the rounding step, we need some further observations:

▶ **Observation 12.** *Let T be a connected component in G_x. Then*

- *T is a tree.*
- *If $|V(T)| > 1$, then T contains exactly one seed edge and all other edges are extension edges.*
- *If T contains a seed edge e_i, then $i = \min_{e_j \in E(T)} j$ or in other words, e_i was the first edge of T considered in the algorithm.*

A proof of these observations is not difficult, but deferred to the full version of this paper for space reasons.

5.2 Analyzing the rounding step

First note that by our rounding procedure, the sum of the edge-weights can increase by at most $\frac{1}{\alpha}$. So for the edge-weights it is sufficient to make sure that $\alpha \geq \frac{1}{2}$ and the main difficulty is to bound the number of components.

Before we choose α, let us estimate how many components we get after rounding and splitting. To do this, we take a look at the connected components after rounding. Let T be such a component. We denote by $\mathrm{comp}(T)$ the number of connected components we need to split T into.

Let C^* be the set of components before splitting and C be the set of components after splitting. Our goal here is to estimate $|C|$ by a contribution $\mathrm{est}(e)$ of each edge $e \in E(G)$, such that the number of components after splitting is

$$|C| = \sum_{T \in C^*} \mathrm{comp}(T) \leq \sum_{T \in C^*} |V(T)| - \sum_{e \in E(G)} \mathrm{est}(e) = |V(G)| - \sum_{e \in E(G)} \mathrm{est}(e)$$

There are three cases:

1. *singletons*: T consists of only one vertex.
2. *good trees*: T consists of more than one vertex and $u(E(T)) + b(V(T)) \leq 1$
3. *large trees*: T consists of more than one vertex and $u(E(T)) + b(V(T)) > 1$

Case 1. T is a singleton. Its number of components is

$$\mathrm{comp}(T) := 1 = |V(T)| - 0.$$

Case 2. T is a good tree. So we can keep this component for a solution to the problem. The number of components is

$$\operatorname{comp}(T) := 1 = |V(T)| - (|V(T)| - 1) \leq |V(T)| - \sum_{e \in E(T)} [1 - 2b(v_e) - 2u(e)] .$$

For all $e \in E(T)$, we set $\operatorname{est}(e) := 1 - 2b(v_e) - 2u(e)$.

Case 3. T is a large tree. So we have to split this component to get a feasible solution. Denote by e' the edge in T with the lowest index according to the sorting of the algorithm. Let $\bar{v} \neq v_{e'}$ be incident to e'. Note that this does not have to be a seed edge, as the components after rounding do not necessarily contain a seed edge. We rewrite the number of components:

$$\begin{aligned}\operatorname{comp}(T) \leq 2 \cdot (u(E(T)) + b(V(T))) = \quad & |V(T)| - [2 - 2b(v_{e'}) - 2u(e') - 2b(\bar{v})] \\ & - \sum_{e' \neq e \in E(T)} [1 - 2b(v_e) - 2u(e)] .\end{aligned}$$

If T contains a seed edge, then this edge is e'. This means that the number of components can be estimated by edges in T. We set $\operatorname{est}(e') := 2 - 2b(v_{e'}) - 2u(e') - 2b(\bar{v})$ and $\operatorname{est}(e) := 1 - 2b(v_e) - 2u(e)$ for all $e \in E(T) \setminus \{e'\}$.

Otherwise T only consists of extension edges. In this case, we write

$$\begin{aligned}[2 - 2b(v_{e'}) - 2u(e') - 2b(\bar{v})] + \sum_{e' \neq e \in E(T)} & [1 - 2b(v_e) - 2u(e)] \\ & = [1 - 2b(\bar{v})] + \sum_{e \in E(T)} [1 - 2b(v_e) - 2u(e)] .\end{aligned}$$

Then, we set $\operatorname{est}(e) := 1 - 2b(v_e) - 2u(e)$ for all $e \in E(T)$. However, in this case we need to account for the additional $1 - 2b(\bar{v})$. To do so, we call the edge incident to $\bar{v}$ that is not contained in T a *filler edge*. For all filler edges $\{v, w\} = e \in E(G)$, we w.l.o.g. assume that e is a filler edge for the component that contains v and set

$$\operatorname{est}(e) := \begin{cases} 2 - (b(v) + b(w)), & \text{if } e \text{ is the filler edge of two components} \\ 1 - b(v), & \text{otherwise.} \end{cases}$$

For all edges not considered before, we set $\operatorname{est}(e) := 0$.

Now we have that

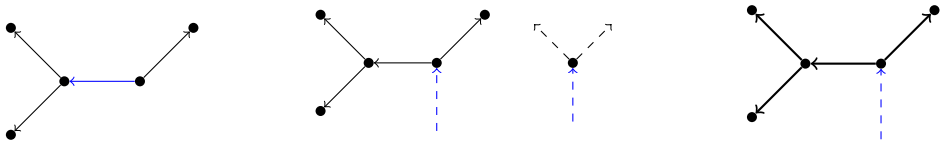
$$|C| \leq |V(G)| - \sum_{e \in E(G)} \operatorname{est}(e)$$

Our next goal is to find a lower bound on $\sum_{e \in E(G)} \operatorname{est}(e)$. Here we will leave out most of the computations, due to space restrictions, but they are contained in the full version [11] of this paper.

5.2.1 Lower bounds for the extension edges

We start with the simpler case of extension edges. An overview over the cases in which they can appear is shown in Figure 1. Let e be an extension edge. If it appears inside a good tree or a large tree. Then

$$\operatorname{est}(e) = 2x(e) - 1 - 2u(e)(1 - x(e)).$$



(a) Inside a (good or large) tree.

(b) Leading towards a good tree or a singleton.

(c) As a filler edge leading to a large tree.

Figure 1 The cases in which extension edges can occur. Dashed edges have been rounded down, while solid ones have been rounded up. Thick edges belong to a large tree. For each edge e the arrowhead points towards v_e.

If it is incident to a singleton or a good tree, we can estimate

$$\text{est}(e) = 0 \geq 2x(e) - 1 - (2x(e) - 1).$$

If it is a filler edge, we can estimate

$$\text{est}(e) = 1 - 2b(v_e) = 1 - 2(1 - x(e) - x(e)u(e)) = 2x(e) - 1 + x(e)u(e) \geq 2x(e) - 1.$$

5.2.2 Lower bounds for the seed edges

Next we consider seed edges. An overview over the cases in which they can appear is shown in Figure 2. Let e be a seed edge. $e = \{v_e, \bar{v}\}$ can not be contained in a singleton. It can also not be contained in a good tree, as we have

$$1 + u(e) > y(e) = 1 - b(v_e) + 1 - b(\bar{v}) \Leftrightarrow b(v_e) + b(\bar{v}) + u(e) > 1.$$

So if it is contained in a connected component, then this component is a large tree and it was the first edge considered in this component. We estimate

$$\text{est}(e) = 2x(e) - 2 - 2u(e)(1 - x(e)).$$

Otherwise both endpoints are incident to different components. This means that it was rounded down. If these components are singletons or good trees, we can estimate

$$\text{est}(e) = 0 \geq 2x(e) - 2.$$

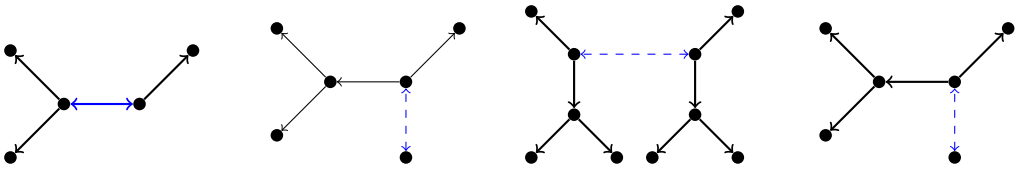
If both are large trees, then e is a filler edge for both and we have

$$\text{est}(e) = 2 - 2(b(v_e) + b(\bar{v})) = 2 - 2(2 - x(e) - x(e)u(e)) = 2x(e) - 2 + 2u(e)x(e) \geq 2x(e) - 2.$$

The last case is that e is incident to one large tree and a good tree or a singleton. This means it is a filler edge for only one endpoint. W.l.o.g. let this endpoint be v_e. We set $y_1 := (1 + u(e))$, $x_1 := 1 - b(v_e)$ and $y_2 := (1 + u(e))$, $x_2 := 1 - b(\bar{v})$. For a later estimate note that then $x_2 \leq \alpha$ as $x(e) \leq \alpha$. We can estimate

$$\text{est}(e) \geq 2x(e) - 2 - (2x_2 - 1).$$

Now almost all estimates are of the same form.



(a) Inside a large tree.

(b) Incident only to good trees or singletons.

(c) As a filler edge for two large trees.

(d) As a filler edge for one large tree and incident to a good tree or a singleton.

Figure 2 The cases in which seed edges can occur. Dashed edges have been rounded down, while solid ones have been rounded up. Thick edges belong to a large tree. For each extension edge e the arrowhead points towards v_e. Seed edges have arrowheads on both ends.

5.2.3 Summary of the estimates

Before we choose α and derive the approximation guarantee, let us summarize the derived estimates.

$\text{est}(e) \geq$

seed edges

$2x(e) - 2 - 0$	incident to good trees or singletons, filler for both ends
$2x(e) - 2 - (2x_2 - 1)$	filler for one end
$2x(e) - 2 - 2u(e)(1 - x(e))$	in a large tree

extension edges

$2x(e) - 1 - 0$	filler edge
$2x(e) - 1 - (2x(e) - 1)$	incident to good tree or singleton
$2x(e) - 1 - 2u(e)(1 - x(e))$	inside a component

The base part, which is left in black, now sums up to at most $2x(E(G)) - |V(G)|$, because there is exactly one seed edge for every component that is not a singleton. So it remains to estimate the parts marked in blue. Our goal will be to estimate this part in terms of $|V(G)| - x(E(G))$. That is, find a β, such that we have "sum of blue parts" $\leq \beta(|V(G)| - x(E(G))$. We will achieve this by first estimating for each $\{v_e, w\} \in E(G_{x^*})$ that

$$\text{"blue part"} \leq \begin{cases} \beta(b(v_e) + b(w) + x(e)u(e)) & \text{for seed edges} \\ \beta(b(v_e) + x(e)u(e)) & \text{for extension edges.} \end{cases}$$

Then, we can use that to sum up the estimates of the differences

$$\text{"sum of blue parts"} \leq \beta(b(V(G)) + u(x(E(G)))) \leq \beta(|V(G)| - x(E(G))),$$

where the last inequality follows directly from the LP-inequalities.

In total, we are left with

$$|C| \leq |V(G)| - \sum_{e \in E(G)} \text{est}(e)$$
$$\leq |V(G)| - (2x(E(G)) - |V(G)| - \beta(|V(G)| - x(E(G)))) = (2 + \beta)(|V(G)| - x(E(G)))$$

For some special instances, we can get approximation ratios that are better than 3, but in general we can not be better than a factor 3 with this technique. We will show this in the last section.

5.2.4 A general approximation guarantee

For a general approximation guarantee, we will choose $\alpha := \frac{2}{3}$ to achieve a 3-approximation (so $\beta = 1$).

We start with the edges $\{v_e, w\} = e$ with a "blue part" of 0: Seed edges that are incident to good trees or singletons, filler edges for both ends or extension edges that are filler edges. Here, we directly see that

$$0 \leq b(v_e) + b(w) + x(e)u(e) \text{ for seed or } 0 \leq b(v_e) + x(e)u(e) \text{ for extension edges.}$$

Next, we cover extension edges that are incident to good trees or singletons. They have been rounded down as well, so we have $x(e) \leq \frac{2}{3}$. This implies

$$2x(e) - 1 \leq \frac{1}{3} \leq 1 - x(e) = b(v_e) + x(e)u(e).$$

Analogously, for seed edges that are filler edges for one end, we get

$$2x_2 - 1 \leq b(v_e) + b(w) + x(e)u(e).$$

Finally, we cover the edges that have been rounded up. These are seed edges in a large tree or extension edges inside a component. Here we have $x(e) \geq \frac{2}{3} \geq 2(1 - x(e))$. So we can get

$$2u(e)(1 - x(e)) \leq x(e)u(e)$$

and again from this $2u(e)(1-x(e)) \leq b(v_e)+b(w)+x(e)u(e)$ for seed edges or $2u(e)(1-x(e)) \leq b(v_e) + x(e)u(e)$ for extension edges.

6 The integrality gap of the LP

We will now prove that the integrality gap of the LP is 3. This means that using the approach discussed here, we can not achieve a better approximation guarantee.

▶ **Theorem 13.** *The integrality gap of the LP-relaxation given in Section 3 is at least* 3.

Proof. For an instance I denote by $\mathrm{OPT}(I)$ the value of an optimum (integral) solution and by $\mathrm{OPT}_{\mathrm{LP}}(I)$ the value of an optimum LP-solution. We will provide a sequence I_k of instances, such that $\lim_{k\to\infty} \frac{\mathrm{OPT}(I_k)}{\mathrm{OPT}_{\mathrm{LP}}(I_k)} = 3$.

Let $0 < \epsilon < \frac{1}{2}$. For some $k \geq 3$, let G be a k-star. That is a graph with $k+1$ vertices $\{C\} \cup \{v_1, \ldots, v_k\}$ and edges $\{\{C, v_i\} \mid i = 1, \ldots, k\}$. We set $c \equiv 0$ and $\gamma := 1$. For all edges $e \in E(G)$, we set $u(e) := \frac{1}{2}$. Finally, we set $b(C) := 1 - \epsilon$ and $b(v_i) := \epsilon$ for $i = 1, \ldots, k$. In order to get to a complete graph, we extend G, by adding edges between all pairs v_i, v_j for $i < j$ and set $c(\{v_i, v_j\}) = 0$ and $u(\{v_i, v_j\}) := 1 - \epsilon$. Clearly, the resulting u and c are metric. We will denote this instance by $I_{k,\epsilon}$. A depiction of $I_{k,\epsilon}$ is shown in Figure 3.

In an optimum integral solution to this instance, no edge can be used. This means that $\mathrm{OPT}(I_{k,\epsilon}) = k + 1$. Now we solve the *LP* using the algorithm from section 2, showing that

$$\mathrm{OPT}_{\mathrm{LP}}(I_{k,\epsilon}) = |V| - \sum_{i=1,\ldots,k} x(e_i) = k + 1 - \frac{2}{3} - (k-1)\left(\frac{2}{3} - \frac{\epsilon}{3}\right) = \frac{k}{3} + \frac{(k-1)\epsilon}{3} + 1.$$

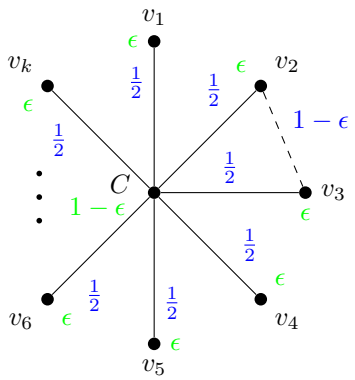


Figure 3 A picture showing the instance described in the proof of Theorem 13. The solid edges belong to the k-star. Edge loads are marked in blue and node loads are marked in green. The dashed edge is an example for the edges added to complete the graph.

Setting $I_k := I_{k,\frac{1}{k^2}}$, we get

$$\lim_{k\to\infty} \frac{\mathrm{OPT}(I_k)}{\mathrm{OPT}_{\mathrm{LP}}(I_k)} = \lim_{k\to\infty} \frac{k+1}{\frac{k}{3}+\frac{(k-1)}{3k^2}+1} = 3. \qquad \blacktriangleleft$$

Together with the upper bound of 3 given by the analysis of the algorithm, we can conclude:

▶ **Corollary 14.** *The integrality gap of the LP is* 3.

References

1. Esther M Arkin, Refael Hassin, and Asaf Levin. Approximations for minimum and min-max vehicle routing problems. *Journal of Algorithms*, 59(1):1–18, 2006. doi:10.1016/J.JALGOR.2005.01.007.
2. Christoph Bartoschek. *Fast Repeater Tree Construction*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, 2014.
3. Guy Even, Naveen Garg, Jochen Könemann, Ramamoorthi Ravi, and Amitabh Sinha. Min–max tree covers of graphs. *Operations Research Letters*, 32(4):309–315, 2004. doi:10.1016/J.ORL.2003.11.010.
4. Stephan Held, Bernhard Korte, Dieter Rautenbach, and Jens Vygen. Combinatorial optimization in vlsi design. *Combinatorial Optimization – Methods and Applications*, 31:33–96, 2011. doi:10.3233/978-1-60750-718-5-33.
5. Dorit S Hochbaum and David B Shmoys. A best possible heuristic for the k-center problem. *Mathematics of operations research*, 10(2):180–184, 1985. doi:10.1287/MOOR.10.2.180.
6. Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. A local search approximation algorithm for k-means clustering. In *Proceedings of the 18th annual symposium on Computational geometry*, pages 10–18. ACM, 2002. doi:10.1145/513400.513402.
7. M. Reza Khani and Mohammad R. Salavatipour. Improved approximation algorithms for the min-max tree cover and bounded tree cover problems. *Algorithmica*, 69(2):443–460, 2014. doi:10.1007/S00453-012-9740-5.
8. Samir Khuller and Yoram J Sussmann. The capacitated k-center problem. *SIAM Journal on Discrete Mathematics*, 13(3):403–418, 2000. doi:10.1137/S0895480197329776.
9. Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982. doi:10.1109/TIT.1982.1056489.

10 Jens Maßberg and Jens Vygen. Approximation algorithms for a facility location problem with service capacities. *ACM Transactions of Algorithms*, 4(4):50:1–50:15, 2008. doi:10.1145/1383369.1383381.

11 Benjamin Rockel-Wolff. A fast 3-approximation for the capacitated tree cover problem with edge loads, 2024. arXiv:2404.10638.

12 Stephan Schwartz. An overview of graph covering and partitioning. *Discrete Mathematics*, 345(8):112884–112900, 2022. doi:10.1016/J.DISC.2022.112884.

13 Vera Traub and Thorben Tröbst. A fast $(2 + \frac{2}{7})$-approximation algorithm for capacitated cycle covering. *Mathematical Programming*, 192(1):497–518, 2022. doi:10.1007/S10107-021-01678-3.

14 Zhou Xu, Dongsheng Xu, and Wenbin Zhu. Approximation results for a min–max location-routing problem. *Discrete Applied Mathematics*, 160(3):306–320, 2012. doi:10.1016/J.DAM.2011.09.014.

15 Wei Yu and Zhaohui Liu. Better approximability results for min–max tree/cycle/path cover problems. *Journal of Combinatorial Optimization*, 37(2):563–578, 2019. doi:10.1007/S10878-018-0268-8.