Approximate Minimum Tree Cover in All Symmetric Monotone Norms Simultaneously

Matthias Kaul
Hamburg University of Technology, Institute for Algorithms and Complexity, Hamburg, Germany
University of Bonn, Germany

Kelin Luo
University of Bonn, Germany
University at Buffalo, NY, USA

Matthias Mnich
Hamburg University of Technology, Institute for Algorithms and Complexity, Hamburg, Germany

Heiko Röglin
Universität Bonn, Germany

Abstract

We study the problem of partitioning a set of n objects in a metric space into k clusters $V_1, \ldots, V_k$. The quality of the clustering is measured by considering the vector of cluster costs and then minimizing some monotone symmetric norm of that vector (in particular, this includes the ℓ_p-norms). For the costs of the clusters we take the weight of a minimum-weight spanning tree on the objects in V_i, which may serve as a proxy for the cost of traversing all objects in the cluster, for example in the context of Multirobot Coverage as studied by Zheng, Koenig, Kempe, Jain (IROS 2005), but also as a shape-invariant measure of cluster density similar to Single-Linkage Clustering.

This problem has been studied by Even, Garg, Könemann, Ravi, Sinha (*Oper. Res. Lett.*, 2004) for the setting of minimizing the weight of the largest cluster (i.e., using ℓ_∞) as Min-Max Tree Cover, for which they gave a constant-factor approximation algorithm. We provide a careful adaptation of their algorithm to compute solutions which are approximately optimal with respect to *all* monotone symmetric norms *simultaneously*, and show how to find them in polynomial time. In fact, our algorithm is purely combinatorial and can process metric spaces with 10,000 points in less than a second.

As an extension, we also consider the case where instead of a target number of clusters we are provided with a set of *depots* in the space such that every cluster should contain at least one such depot. One can consider these as the fixed starting points of some agents that will traverse all points of a cluster. For this setting also we are able to give a polynomial-time algorithm computing a constant-factor approximation with respect to all monotone symmetric norms simultaneously.

To show that the algorithmic results are tight up to the precise constant of approximation attainable, we also prove that such clustering problems are already APX-hard when considering only one single ℓ_p norm for the objective.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Clustering, spanning trees, all-norm approximation

Digital Object Identifier 10.4230/LIPIcs.STACS.2025.57

Related Version *Full Version*: https://arxiv.org/abs/2501.05048 [18]

Funding *Matthias Kaul*: Most work done while at the Hamburg University of Technology.
Kelin Luo: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 390685813.
Heiko Röglin: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 459420781 and by the Lamarr Institute for Machine Learning and Artificial Intelligence lamarr-institute.org.

42nd International Symposium on Theoretical Aspects of Computer Science (STACS 2025).
Editors: Olaf Beyersdorff, Michał Pilipczuk, Elaine Pimentel, and Nguyễn Kim Thắng;
Article No. 57; pp. 57:1–57:18

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

A typical clustering problem takes as input a set of n objects in a metric space (V, d), and seeks a partition of these objects into k *clusters* $V_1, \ldots, V_k$, so as to optimize some objective function. For example, we might try to place k facilities onto the nodes of an edge-weighted graph with node set V, and then assign each remaining node to some facility. To model the cost of serving these nodes from their facility, one can use the cost of a minimum-weight spanning tree T_i on the subgraph induced by them. For $i = 1, \ldots, k$, let $w(T_i) = \sum_{e \in E(T_i)} w(e)$ be the weight of tree T_i. Historically, these kinds of problems have been studied for two different objectives. On the one hand, in the Min-Sum Tree Cover problem one might want to find k trees $T_1, \ldots, T_k$ to cover all nodes in V, while minimizing the total length of the service network, i.e., the sum $\sum_{i \in [k]} w(T_i)$ of the weights of the spanning trees. On the other hand, it is desirable that each facility does not serve too large of a network, so we can instead minimize the weight $\max_{i \in [k]} w(T_i)$ of the heaviest tree, which leads to the Min-Max Tree Cover problem [10, 19].

Many fundamental clustering problems were studied under this min-sum objective and min-max objective. These objectives can equivalently be considered as that of minimizing the 1-norm, or the ∞-norm, of the clustering. Examples include the k-median problem and the minimum-load k-facility location problem. These two problems are siblings in that they both deal with the assignment of points (or clients) to k centers (or facilities), with the costs of connecting those points to respective centers. Each point v is then assigned to one of these open centers, denoted as $c(v)$. The cost cost_c for each center c, which also reflects the cost associated with each facility, is calculated as the sum of distances between the center and all the points allocated to it, i.e, $\text{cost}_c = \sum_{v \in V : c(v) = c} d(v, c)$. In the k-median problem [14, 7, 21, 2], the objective is to minimize the ℓ_1-norm of $\{\text{cost}_c\}_c$, i.e., $\sum_c \text{cost}_c$, which represents the total cost of all clusters; whereas the minimum-load k-facility location problem [1] seeks to minimize the maximum load of an open facility, symbolised by the ℓ_∞-norm function of $\{\text{cost}_c\}_c$, i.e., $\max_c \text{cost}_c$.

There is a diverse array of cost functions that could be applicable to a variety of problem domains, and often, efficient algorithms are crafted to suit each specific objective. However, it is crucial to note that an optimal solution for one objective may not perform well for another. For instance, a solution for an instance of the Tree Cover that minimizes the ℓ_1-norm might be particularly inefficient when it comes to minimizing the ℓ_∞-norm, and vice versa (see examples in Figure 1a and Figure 1b). Therefore, one may wonder what the "generally optimal" solution would be for a given problem. Finding such a generally optimal solution is the task of the following problem:

▶ **Definition 1** (All-norm clustering problem). *An* all-norm clustering problem *takes as input a metric space (V, d), a cost function $w : \mathcal{P}(V) \to \mathbb{R}_{\geq 0}$, and a positive integer k. The goal is to partition V into clusters $V_1, \ldots, V_k$ which minimize*

$$\alpha = \max_{p \in \mathbb{R}_{\geq 1} \cup \{\infty\}} \frac{\left(\sum_i \left(w(V_i)\right)^p\right)^{1/p}}{OPT_p},$$

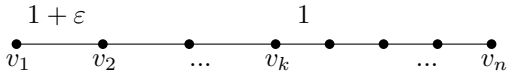
where OPT_p denotes the value of an optimal solution for the k-clustering problem under the ℓ_p-norm objective. Here, α is referred to as the all-norm approximation factor.

Our focus in this paper is the all-norm clustering problem where the cost $w(V_i)$ of each cluster is the cost of a minimum-weight spanning tree on V_i with respect to the metric d. We call this problem the All-Norm Tree Cover problem in line with the naming convention

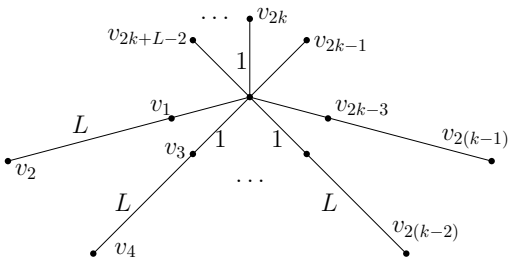
in the literature, and denote its instances by (V, d, w, k). (Note that, for simplicity, we only consider ℓ_p-norms here. However, the proofs turn out to work for any norm which is monotone and symmetric, cf. Ibrahimpur and Swamy [16] for a detailed introduction to such norms.) Observe that the number k of clusters is part of the input, i.e., it is *not* fixed.

The choice of the cost of a minimum-weight spanning tree might appear to be somewhat arbitrary here, but spanning trees are the key connectivity primitive in network design problems. Any constant-factor approximation for All-Norm Tree Cover will, for example, transfer to a constant-factor approximation for the all-norm version of the Multiple Traveling Salesperson Problem [5] where we ask to cover a metric space by k cycles instead of trees. This use of spanning trees as a proxy for the traversal times of the clusters was a key ingredient for by Zheng et al. [24] to partition a floorplan into similar-size areas to be served by different robots.

Let us observe that solving the All-Norm Tree Cover problem is non-trivial, as a solution which is good for one objective (i.e., for one particular norm ℓ_p) may be bad for another objective (i.e., for another norm $\ell_{p'}$), and vice versa. For examples of this phenomenon, see Figure 1.



(a) The path forms a metric where any two neighboring nodes in the set $\{v_1, \ldots, v_k\}$ have distance $1+\varepsilon$, while the distance of all other edges is 1. In this case, the optimal solution under the ℓ_1-norm involves removing all edges with distance $1+\varepsilon$, resulting in an ℓ_∞-norm objective cost $n-k$. However, the optimal solution cost for the ℓ_∞-norm is less than $n(1+\varepsilon)/k$.



(b) The spider graph includes $2k+L-1$ nodes. All edges connecting a node v_{2i-3} (where $i = 1, \ldots, k$) and v_i (where $i = 2k-1, \ldots, 2k+L-2$) to the center have distance 1, and all edges $\{v_{2i-1}, v_{2i}\}_{i=1,\ldots,k-1}$ have distance a large integer L. In this case, the optimal solution under the ℓ_∞-norm involves removing all edges connected to the nodes $\{v_{2i-1}\}_{i=1,\ldots,k-1}$, resulting in an ℓ_1-norm objective cost of kL. However, the optimal solution for the ℓ_1 norm has cost $k+L-1$.

Figure 1 Two instances of the All-Norm Tree Cover problem. Figure 1a is an instance where an optimal ℓ_1-norm solution does not return a good approximation in the ℓ_∞-norm; Figure 1b is an instance where an optimal ℓ_∞-norm solution does not return a good approximation in the ℓ_1-norm.

The All-Norm Tree Cover problem, alongside the related path cover and cycle cover problems, involves covering a specified set of nodes of a(n edge-weighted) graph with a limited number of subgraphs. These problems have attracted significant attention from the operations research and computer science communities due to their practical relevance in fields like logistics, network design, and data clustering. For instance, these problems are naturally applicable in scenarios such as vehicle routing, where the task involves designing optimal routes to service a set of customers with a finite number of vehicles. Different optimization objectives could be considered depending on the specific requirements. One could aim to minimize the maximum waiting time for any customer, an objective that is equivalent to the ℓ_∞-norm of the cost function associated with each vehicle. This ensures fairness, as it attempts to prevent any single customer from waiting excessively long. Alternatively, one could aim to minimize the total travel time or cost, which corresponds to the ℓ_1-norm of

the cost function across all vehicles [6, 4]. This objective seeks overall efficiency, making it beneficial from an operational perspective as it reduces fuel consumption and allows for more customers to be serviced within the same time frame [11, 4, 10, 19]. Understanding these problems in an all-norm setting thus enables the development of routing strategies that are adaptable to different priorities, including customer satisfaction, operational efficiency, or a balance between the two. Given that minimum spanning trees provide constant-factor approximations to traveling salesperson tours [8], we explore the possibility of covering the nodes of a graph with k trees.

In a scenario where each vehicle already has an assigned station, and the task is limited to the assignment of nodes to the vehicles, we are confronted with a variation of the All-Norm Tree Cover problem known as the All-Norm Tree Cover with Depots problem (also commonly referred to as the Rooted All-Norm Tree Cover problem), denoted by (V, d, w, D) where $D \subseteq V$ is a set of depots [10]. These problems under all norms will be formally defined and addressed in the subsequent sections of this paper.

1.1 Our contributions

Computing optimal tree covers is NP-hard even for the *single* ℓ_∞-norm; for this norm, it admits a constant-factor approximation in polynomial time. Our goal in this paper is thus to design constant-factor approximations for All-Norm Tree Cover, that is, for *all* monotone symmetric norms *simultaneously*. Building upon the example in Figure 1, it becomes clear that a constant-factor approximation for one norm objective does not necessarily guarantee a constant approximation for another norm. Meanwhile, achieving an all-norm approximation factor better than 3/2 within polynomial time is infeasible, as a lower bound of 3/2 on the approximability of the Tree Cover problem under the ℓ_∞-norm has been demonstrated by Xu and Wen [23] (assuming $\mathsf{P} \neq \mathsf{NP}$). In previous work, Even et al. [10] proposed a 4-approximation algorithm for the Min-Max Tree Cover problem, which is representative of the ℓ_∞-norm. That algorithm was subsequently refined by Khani and Salavatipour [19], who devised a 3-approximation. However, these existing algorithms fail to guarantee a constant approximation factor for optimal solutions under other norms, proving to be unbounded specifically, this holds even for the ℓ_1-norm (for an example, see Figure 3 in Section 3).

Our first main contribution is a polynomial-time constant-factor approximation algorithm for the All-Norm Tree Cover problem. Our algorithm amalgamates the strategies of suitable algorithms for both the ℓ_1-norm and the ℓ_∞-norm. It is well-understood that an optimal solution for the ℓ_1-norm objective involves successively eliminating the heaviest edges until precisely k components remain. Conversely, the algorithm for ℓ_∞-norm objective concludes when the number of trees generated by the edge-decomposition with respect to a guessed optimal value R is about to exceed k [10]. Our proposed method continually approximates the decomposed trees towards an ideal state. So in that sense the algorithm proposed by Even et al. [10] is effectively refined to solve other norm problems. Notably, our algorithm produces a feasible solution that is at most double the optimal solution for the ℓ_1-norm and four times the optimal solution for the ℓ_∞-norm simultaneously.

▶ **Theorem 2.** *There exists a polynomial-time $O(1)$-approximation algorithm for the* All-Norm Tree Cover *problem.*

We next consider the more involved All-Norm Tree Cover problem with depots. For All-Norm Tree Cover with depots, each tree in a tree cover solution is rooted at a specific depot. The special problem case of the single ℓ_∞-norm is the Min-Max Tree

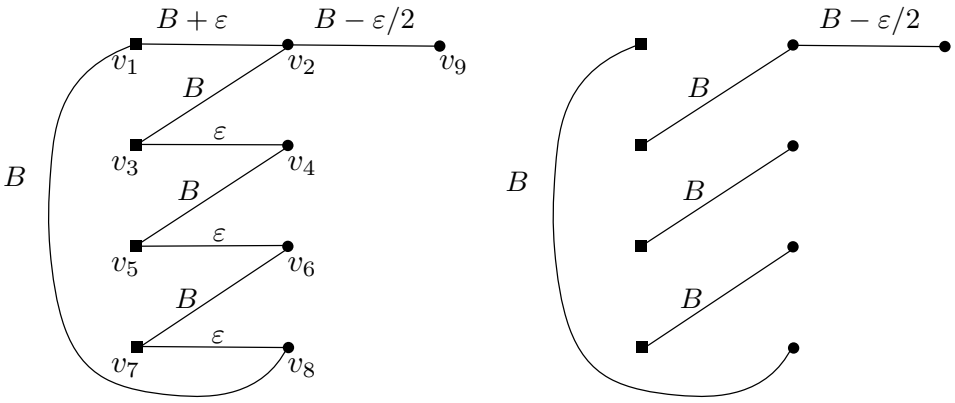


Figure 2 Example instance where the algorithm of Even et al. [10] does not return a good approximation in the ℓ_1-objective. The left figure is an example of the graph metric when $k = 4$ and the right figure is the final solution obtained by the rooted-tree-cover algorithm by Even et al. [10].

Cover problem with depots, for which Even et al. [10] devised a 4-approximation algorithm. Subsequently, Nagamochi [22] enhanced this framework by offering a $(3-\frac{2}{k+1})$-approximation algorithm under the restriction that all depots are located at the same node. However, neither of those algorithms can assure any constant approximation factor for the ℓ_1-norm objective (refer to the example in Figure 2).

Our key contribution extends the $O(1)$-approximation algorithm for All-Norm Tree Cover from Theorem 2 to the problem with depots:

▶ **Theorem 3.** *There exists a polynomial-time $O(1)$-approximation algorithm for the* All-Norm Tree Cover with Depots *problem.*

Our methodology comprises three stages:

1. Initially, we partition the nodes according to their distances to the depots, where partition class i contains all nodes at distance between 2^{i-1} and 2^i.
2. We then apply a version of the algorithm of Even et al. [10] in each class separately to find a good "pre-clustering" of the nodes. This is simplified by the previous partitioning since it allows us to assume that all nodes are at the same distance to the depots, up to a factor of 2.
3. Finally, we assign the clusters from the second step to the depots by iteratively computing matchings of clusters to depots, considering ever larger clusters, and allowing them to be matched to ever more distant depots. In this way, we essentially maintain a running estimate of the necessary tree weights, updating it when the matching process does not succeed in assigning all nodes to some depot. If it does succeed, we can show that (up to constant factors) no cluster is larger than the estimate, and that the estimate is correct, i.e., every solution has trees at least as large as predicted by the estimate.

Note that both the depot and non-depot algorithms will require only very simple algorithmic primitives such as minimum spanning trees and bipartite matchings. Thus, our algorithms can be implemented to run very quickly using purely combinatorial techniques, in particular without resorting to linear programming which can be impractically slow (see also Davies at al. [9] for recent work on combinatorial clustering algorithms). In fact, an implementation of our algorithm can process metric spaces with 10,000 points in less than a second, see Section 6 for details.

Our final contribution complements the algorithmic result from Theorem 3 by a complexity-theoretic lower bound: we show that one cannot expect polynomial-time approximation schemes to exist, even in the presumably easier setting ℓ_p-Tree Cover with Depots where we only want to approximate the optimum with respect to *one specific* ℓ_p-norm:

▶ **Theorem 4.** *For every $p \in (1, \infty]$ there exists a constant c such that ℓ_p-Tree Cover with Depots is NP-hard to approximate within a factor c. The NP-hardness holds under randomized reductions.*

Specifically, we show that ℓ_p-Tree Cover is NP-hard to approximate to a factor

$$\left[\frac{(106 - \frac{1}{4} + \frac{\varepsilon}{2})3^p + (\frac{1}{8} - \frac{\varepsilon}{4})(2^p + 4^p)}{(106 - 2\varepsilon)3^p + \varepsilon(2^p + 4^p)}\right]^{1/p} > 1,$$

for any choice of $\varepsilon > 0$.

2 Preliminaries

We set up some formal definitions. We will identify metric spaces and metrically edge-weighted graphs to allow for an easier treatment of connectedness, trees, and similar graph-theoretic objects. For any such graph G, we always keep in mind the underlying metric space V induced by the shortest-path metric on G. We also assume that the metrics used are integral, for ease of presentation. Our results extend to rational metrics by rescaling the metric.

We here concern ourselves with tree covers of graphs, which are to be defined as follows:

▶ **Definition 5** (Tree cover). *For a graph G , a* tree cover *is a collection $\{T_1, \ldots, T_k\}$ of trees for which $\bigcup_i V(T_i) = V$. We call k the* size *of such a tree cover.*

▶ **Definition 6** (Tree cover with depots). *For a graph G and a depot set $D \subseteq V$, a* tree cover with depots *is a tree cover $\{T_1, \ldots, T_{|D|}\}$ of G, where each T_i contains a unique depot from D.*

Notice in particular that there is no expectation of disjointness for the trees. If disjointness is required, we may assign every node to exactly one of the trees currently containing it and recompute minimum-weight spanning trees for each of the resulting clusters. This increases the cluster weights by at most a factor of 2 due to the gap between Steiner trees and minimum-weight spanning trees (see Lemma 9). For any connected subgraph H of a graph G, we use $w(H)$ to denote the weight of a minimum-weight spanning tree in the subgraph H. It is important to note that this weight can differ from the sum of the weight of the edges of H. We then use the p-norm ($p \geq 1$) as a measure of the quality of a tree cover. Specifically, the cost of a tree cover $\{T_1, \ldots, T_k\}$ is defined as the p-norm of the corresponding tree weights, that is, $w_p = \left(\sum_{i \in [k]} (w(T_i))^p\right)^{1/p}$.

There are two natural optimization questions for tree covers (and for tree covers with depots) which have been considered in the literature: The ℓ_1-Tree Cover problem, where the aim is to minimize the sum of the weights of the k trees, and the ℓ_∞-Tree Cover problem in which the objective is to minimize the weight of the heaviest tree within the cover. The ℓ_1-Tree Cover problem admits a polynomial-time algorithm, regardless of the presence of depots; in contrast, the ℓ_∞-Tree Cover problem is APX-hard, again regardless of whether depots are present or not.

We consider the interpolation between these two problems by allowing arbitrary ℓ_p norms:

▶ **Definition 7** (ℓ_p-Tree Cover (resp. ℓ_p-Tree Cover with Depots)). *Given a graph $G = (V, d)$ and some integer k (resp. depots $D \subseteq V$) and a $p \in [1, \infty)$, find a tree cover $\{T_1, \ldots, T_k\}$ (resp. with depots) which minimizes the expression $OPT_{p,k} := \left(\sum_{i=1}^{k} (w(T_i))^p\right)^{1/p}$.*

We will usually omit k if it is clear from context. All of the ℓ_p-variants (except $p = 1$) are NP-hard, as the proof of hardness for the ℓ_∞-case works for all values of $p > 1$ [23].

▶ **Definition 8** (All-Norm Tree Cover Problem (resp. All-Norm Tree Cover Problem with Depots)). *Given a graph $G = (V, d)$ and some integer k (resp. depots $D \subseteq V$), find a tree cover $\{T_1, \ldots, T_k\}$ (resp. with depots) which minimizes*

$$\max_{p \in [1,\infty)} \frac{\left(\sum_{i=1}^{k} (w(T_i))^p\right)^{1/p}}{OPT_p} .$$

To distinguish the problem versions for a *fixed* norm p from those for *all p simultaneously*, we denote by (ℓ_p, k) (or (ℓ_p, D) for problems involving depots) the tree cover problem for a specific value of p. Meanwhile, we use $(\ell_{p\in[1,\infty)}, k)$ (or $(\ell_{p\in[1,\infty)}, D)$ when depots are involved) to represent the All-Norm Tree Cover problem that encompasses all values of $p \in [1, \infty)$. We omit naming the underlying metric space (V, d), unless explicitly stated otherwise.

We will state a result about the relationship between the cost of minimum-weight Steiner trees and minimum-weight spanning trees, as we will harness this argument repeatedly:

▶ **Lemma 9** (Kou et al. [20]). *Let (V, d) be a metric space with some terminal set $F \subseteq V$, let $\hat{T}$ be a minimum-weight (Steiner) tree containing all nodes from F, and let T be a minimum-weight spanning tree of $(F, d|_F)$. Then $w(T) \leq (2 - 2/|F|) \cdot w(\hat{T})$.*

Proof. An easy way to obtain the result is to take $\hat{T}$ and use the tree-doubling heuristic to compute a traveling salesperson tour $\mathcal{T}$ of F in (V, d), costing at most $2 \cdot w(\hat{T})$. Removing the heaviest edge of $\mathcal{T}$ yields a (perhaps not yet minimal) spanning tree T' in $(F, d|_F)$ of weight at most $(2 - 2/|F|)w(\hat{T})$, allowing us to conclude that $w(T) \leq w(T') \leq (2 - 2/|F|)w(\hat{T})$. ◀

Lemma 9 will prove useful as we occasionally want to forget about some nodes of our instance. This may actually increase overall costs, since certain Steiner trees become unavailable. However, the lemma ensures that the increase in cost is bounded.

3 Simultaneous Approximations for the All-Norm Tree Cover Problem

First, to provide some some good intuition of the key techniques for the general case, we show in detail that the Tree Cover algorithm of Even et al. [10] gives a constant-factor approximation to the ℓ_p-tree cover problem without depots provided that it returns k trees. Indeed, we show that the argument will work for all monotone symmetric norms. The original algorithm works as follows, for a given graph $G = (V, d)$:

1. Guess an upper bound R on the optimum value of the ℓ_∞-norm tree cover problem for instance G, where initially $R = 1$.
2. Remove all edges with weight larger than R from G, and compute a minimum-weight spanning tree T_i for each connected component of the resulting graph.
3. Check that $\sum \lfloor \frac{w(T_i)+2R}{2R} \rfloor = k$; otherwise, reject R and go to step 1 with $R := 2R$.
4. Call a spanning tree T_i *small* if $w(T_i) < 2R$.

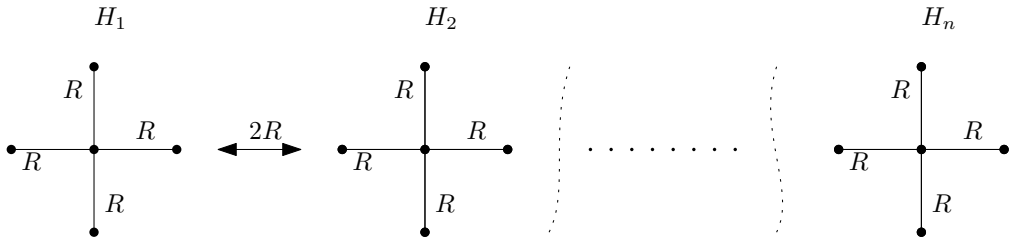


Figure 3 Example instance where the algorithm of Even et al. does not return a good approximation in the ℓ_1-objective. The instance consists of n identical copies H_i of the 4-star where all edges have length R and the copies are pairwise at distance $2R$. For $k = 5n - 1$, the instance has $OPT_\infty = OPT_1 = R$, but the algorithm will return $2n$ trees, each of weight $2R$. Observe that the partition requirement in Step 5 would also be fulfilled if the trees are not further partitioned and kept at size $4R$, however the algorithm of Even et al. produces the former solution. Both solutions do not achieve a good approximation in the ℓ_1 objective.

5. Call a spanning tree T_i *large* if $w(T_i) \geq 2R$. Decompose each such tree into edge-disjoint subtrees $\tilde{T}_j$ such that $2R \leq w(\tilde{T}_j) \leq 4R$ for all but one residual $\tilde{T}_j$ in each component[1].
6. Output the family of all small spanning trees, as well as all subtrees $\tilde{T}_j$ created in Step 5.

Even at al. show that this procedure will output *at most* k trees if $OPT_\infty \leq R$, although they relax the condition in Step 3 to $\sum \lfloor \frac{w(T_i)+2R}{2R} \rfloor \leq k$. For technical reasons, however, we need the equality in this spot. Since every tree has weight most $4R$, the algorithm evidently gives a 4-approximation in the ℓ_∞-norm if the correct value of $R = OPT_{\infty,k}$ is guessed. Otherwise, one can use binary search to find, in polynomial time, the smallest R for which one gets at most k trees.

In general, however, we notice that this algorithm will sometimes compute fewer than k trees, causing it to not return a good approximation for the other ℓ_p-norms, not even for the ℓ_1-norm (see Figure 3).

For this reason, we need the strengthened property in Step 3. Then, however, notice that such a value R does not necessarily exist; for example, for the instance in Figure 3 this is the case. We will describe later how to avoid this issue of non-existence; for now, we assume that such a value R exists and can be computed in polynomial time.

So suppose that the algorithm has returned k trees $T_1, \ldots, T_k$, where we simply remove some edges should the algorithm return fewer than k trees. This removal only improves the objective value, so we will assume – without loss of generality – that the algorithm already returned k trees. As a warm-up, we will start by considering only the case that $p = 1$, i.e., we show that this algorithm computes a solution that is a good approximation for (ℓ_1, k) tree cover.

▶ **Lemma 10.** *Let $\{T_1, \ldots, T_k\}$ be the set of trees returned by the algorithm for some fixed value of R. Then we have $\sum w(T_i) \leq 2OPT_{1,k}$.*

Proof. We observe first that an optimal solution for (ℓ_1, k) tree cover can be computed by removing iteratively the heaviest edge as long as this does not cause the graph to decompose into more than k components, as this procedure is equivalent to running Kruskal's algorithm. As the optimal solution contains no edges of weight greater than R, we consider the graph $G' := G - \{e \mid d(e) > R\}$. Let k_s be the number of small spanning trees S_i computed by

[1] This can be done with a simple greedy procedure, cutting of subtrees of this weight. For the details of this algorithm, we refer to Even et al. [10].

the algorithm, and let k_ℓ be the number of large spanning trees L_i. Then we certainly have $\sum w(T_i) \leq \sum w(S_i) + \sum w(L_i)$, since the trees computed by the algorithm are edge-disjoint subtrees of the initial spanning trees. Further, we have

$$OPT_{1,k} \geq \sum w(S_i) + \sum w(L_i) - (k - k_s - k_\ell)R,$$

because the optimal solution will remove $k - k_s - k_\ell$ edges from within the spanning trees computed by the algorithm, each of weight at most R. But notice that we can use Step 3 to obtain

$$k = \sum \left\lfloor \frac{w(S_i)}{2R} + 1 \right\rfloor + \sum \left\lfloor \frac{w(L_i)}{2R} + 1 \right\rfloor \leq k_s + k_l + \sum \frac{w(L_i)}{2R},$$

which implies that $(k - k_s - k_\ell)2R \leq \sum w(L_i)$. This inequality allows us to conclude that

$$\begin{aligned} OPT_{1,k} &\geq \sum w(S_i) + \sum w(L_i) - \frac{1}{2}\sum w(L_i) \\ &\geq \frac{1}{2}(\sum w(S_i) + \sum w(L_i)) \\ &\geq \frac{1}{2}\sum w(T_i) \ . \end{aligned}$$

◀

Notice that the strengthened version of Step 3 is indeed necessary here.

We can use a similar technique to show that the solution computed by the algorithm is a constant-factor approximation for (ℓ_p, k) tree cover for any choice of p, and with a constant of approximation independent of p. The key argument will be to show that an optimal solution to (ℓ_p, k) tree cover must, as in Lemma 10, have a total weight comparable to that of the solution computed by the algorithm. In a second step, one can then show that the algorithm distributes the total weight of its trees fairly evenly, because the large trees all have weight between $2R$ and $4R$.

Meanwhile, for the small trees we can demonstrate that choosing them differently from the algorithm will incur some cost of at least R. Thus, either the algorithms' solution has correctly chosen the partition on these small trees, or the optimal solution actually has a heavy tree not in the algorithms' solution, which we can use to pay for one of the large trees that the algorithm has, but the optimal solution does not. Notice that, if the algorithm achieves an equal distribution of some total weight that is comparable to the total weight of an optimal solution under the ℓ_p-norm, it also achieves a good approximation of OPT_p due to convexity of the ℓ_p-norms.

To start with, we will only give a rough analysis of the quality of the solution returned by the algorithm, although it already shows a constant factor of approximation. The full version on arXiv gives a more detailed proof that achieves a better constant [18].

Let us fix some $p \in [1, \infty)$ and any (ℓ_p, k) tree cover solution $\{\hat{T}_1, \ldots, \hat{T}_k\}$ (you may imagine that it is optimal). Then let k_s be the number of connected components computed by the algorithm that had a small spanning tree, and call those components $S_1, \ldots, S_{k_s}$. Similarly, for the large spanning trees, let there be k_ℓ components $L_1, \ldots, L_{k_\ell}$. Now we denote by E_ℓ the set of edges that are both contained in some $\hat{T}_i$ and in the cut induced by some S_j or L_j. In particular, all these edges have weight at least R. We count separately the small T_i incident to some edge from E_ℓ, say there are $k_{s,1}$ of them. We will also denote by $k_{s,2}$ the number of S_i for which one of the $\hat{T}_j$ is a minimum-weight spanning tree, and denote the set of these $\hat{T}_j$ as $T^=$. Note that any small component not counted by $k_{s,1}$ or $k_{s,2}$ is split into at least two components by the $\hat{T}_i$. For an illustration of this setting, we refer to Figure 4.

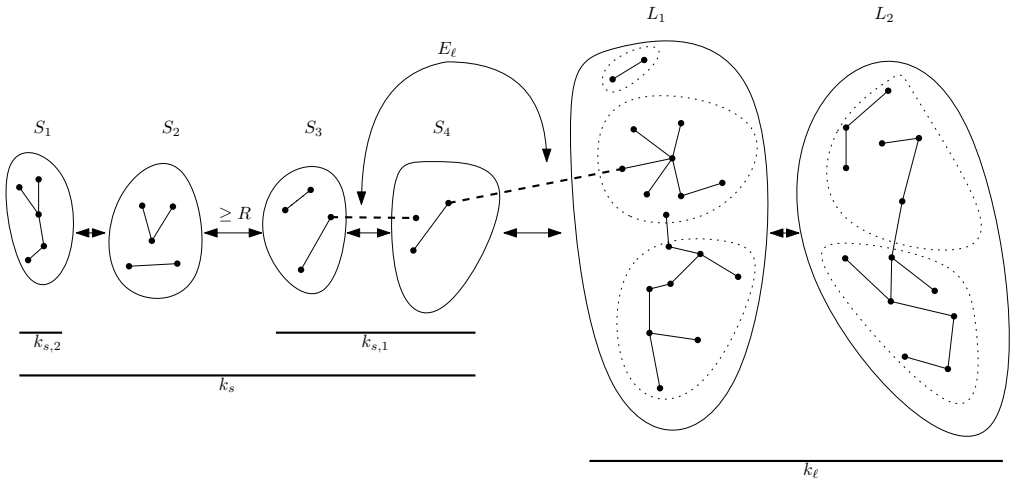


Figure 4 Illustration of how the algorithm's solution and some optimal solution can align against each other. The algorithm's partition of the graph into connected component is indicated by solid zones, the further subdivision of the large components by dotted zones. The trees of the optimal solution are drawn, and all edges crossing the boundary of a connected component are dashed.

We can now start to measure the total sum of edge weights in the $\hat{T}_j$, i.e., $\sum_j w(\hat{T}_j)$. We begin by relating it to the small trees of the algorithm's solution that do not agree with the optimal solution.

▶ **Observation 11.** *It holds that* $\sum_j w(\hat{T}_j) - \sum_{\hat{T}_i \in T^=} w(\hat{T}_i) \geq k_{s,1} \frac{R}{2}$.

Proof. We have $k_{s,1}$ pairwise disjoint node sets in G, each incident to at least one edge of weight at least R present in E_ℓ, so we get $|E_\ell| \geq k_{s,1}/2$ from the handshake lemma, and thus

$$\sum_j w(\hat{T}_j) - \sum_{\hat{T}_i \in T^=} w(\hat{T}_i) \geq |E_\ell| R \geq \frac{k_{s,1}}{2} R,$$

noting that the trees in $T^=$ do not contain any of the edges from E_ℓ. ◀

To compare the total weight against the number of large trees, we can use a similar argument as in Lemma 10. We again note that the initial spanning trees have weight at least $(k-k_\ell-k_s)2R$, and any tree cover cannot remove too many edges from them. The second part of this argument is no longer true though, since it is now possible for a solution to have many edges between the components of $G - \{e \mid d(e) \geq R\}$, allowing it to remove many edges within the components However, a careful analysis will show that this case does not pose a problem, since such inter-component edges are themselves heavy.

▶ **Observation 12.** *We have* $\sum_j w(\hat{T}_j) \geq (k - k_\ell - k_{s,1} - k_{s,2})R + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i)$.

Proof. Suppose we delete from the $\hat{T}_j$ all edges from E_ℓ. This will yield $k + |E_\ell|$ trees. We will count only those trees that lie in a large component L_i, of which there are at most $k+|E_\ell|-2k_s+k_{s,1}+k_{s,2}$. This is because every small component contains at least 2 of the $\hat{T}_j$, except for $k_{s,1}+k_{s,2}$ many. Now observe that to get a $(k+|E_\ell|-2k_s+k_{s,1}+k_{s,2})$-component spanning forest of *minimum* weight for the L_i, we start with the initial minimum spanning trees in each component (which have weight at least $(k - k_s - k_\ell)2R$) and remove at most $(k + |E_\ell| - k_\ell - 2k_s + k_{s,1} + k_{s,2})$ edges, each of weight at most R. The total remaining weight is $(k - k_s - k_\ell)2R - (k + |E_\ell| - k_\ell - 2k_s + k_{s,1} + k_{s,2}) = (k - k_\ell - k_{s,1} - k_{s,2} - |E_\ell|)R$.

Thus, we can measure the total weight of edges which are part of some $\hat{T}_i$ and lie in a large component as being at least the total weight the spanning trees of the L_i, minus the weight of the edges that may have been removed, and obtain

$$\begin{aligned} \sum_j w(\hat{T}_j) - \sum_{\hat{T}_i \in T^=} w(\hat{T}_i) - |E_\ell| R &\geq (k - k_\ell - k_{s,1} - k_{s,2} - |E_\ell|)R. \\ \implies \sum_j w(\hat{T}_j) - \sum_{\hat{T}_i \in T^=} w(\hat{T}_i) &\geq (k - k_\ell - k_{s,1} - k_{s,2})R \ . \end{aligned}$$ ◀

Notice that with these two observations, we can almost show some result along the lines of $OPT_{1,k} \geq c \cdot k \cdot R$ for some $c \in \mathbb{R}$, since either there are many large trees, in which case Observation 12 gives us the desired result, or there are many small trees in which case we can use Observation 11, unless $k_{s,2}$ is large. However, the case that $k_{s,2}$ is large, i.e., we have computed many of the trees that are present in the optimal solution, should also be beneficial, so we maintain a separate record of $k_{s,2}$ in the analysis to obtain:

▶ **Lemma 13.** *It holds that* $\sum_j w(\hat{T}_j) \geq \frac{R}{6}(k - k_{s,2}) + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i)$.

Proof. We combine Observation 12 and Observation 11 convexly with coefficients $1/3, 2/3$ to obtain

$$\begin{aligned} \sum_j w(\hat{T}_j) &\geq \frac{1}{3}\left[(k - k_\ell - k_{s,1} - k_{s,2})R\right] + \frac{2}{3}\left[k_{s,1}\frac{R}{2}\right] + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i) \\ \iff \sum_j w(\hat{T}_j) &\geq \frac{R}{3}(k - k_\ell - k_{s,2}) + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i) \\ \implies \sum_j w(\hat{T}_j) &\geq \frac{R}{6}(k - k_{s,2}) + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i), \end{aligned}$$

where the final inequality follows from the following reasoning:

$$k = \sum \left\lfloor \frac{w(S_i) + 2R}{2R} \right\rfloor + \sum \left\lfloor \frac{w(L_i) + 2R}{2R} \right\rfloor \geq k_s + 2k_\ell,$$

and thus $k - k_\ell - k_{s,2} \geq k - \frac{k-k_s}{2} - k_{s,2} = \frac{k}{2} + \frac{k_s}{2} - k_{s,2} \geq \frac{k-k_{s,2}}{2}$. ◀

The upshot of Lemma 13 is then that any tree cover using k trees with some $k_{s,2}$ trees in common with the algorithm's solution will have the property that the other $k - k_{s,2}$ trees have an average weight of $\Omega(R)$.

▶ **Theorem 14.** *If the algorithm of Even et al. returns k trees $T_1, \ldots, T_k$, then we have*

$$\left[\sum_{i=1}^{k}(w(T_i)^p)\right]^{1/p} \leq 24 \left[\sum_{i=1}^{k}(w(\hat{T}_i)^p)\right]^{1/p}$$

for any p and for any tree cover $\{\hat{T}_1, \ldots, \hat{T}_k\}$ of G with k trees.

Proof. From Lemma 13 we obtain

$$\sum_{i=1}^{k}(w(\hat{T}_i)^p) \geq (k - k_{s,2})\left(\frac{R}{6}\right)^p + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i)^p,$$

using convexity of $|x|^p$ for any $p \geq 1$. At the same time, from the algorithm it is clear that

$$\sum_{i=1}^{k}(w(T_i)^p) \leq (k - k_{s,2})(4R)^p + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i)^p,$$

because every tree is either identical to one of the $\hat{T}_j$, or has weight at most $4R$. Putting these inequalities together yields the statement of the theorem:

$$\frac{\sum_{i=1}^{k}(w(T_i)^p)}{\sum_{i=1}^{k}(w(\hat{T}_i)^p)} \leq \frac{(k-k_{s,2})(4R)^p + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i)^p}{(k-k_{s,2})\left(\frac{R}{6}\right)^p + \sum_{\hat{T}_i \in T^=} w(\hat{T}_i)^p} \leq \frac{(k-k_{s,2})(4R)^p}{(k-k_{s,2})\left(\frac{R}{6}\right)^p} \leq 24^p \ . \qquad \blacktriangleleft$$

The whole proof, in particular the result of Lemma 13 can be reinterpreted to give an even stronger result: namely, that the tree cover returned by the algorithm is approximately "strongly optimal", a concept introduced by Alon et al. [3] for analysing all-norm scheduling algorithms. The point is to show a strong version of lexicographic minimality of the constructed solution. Formally, let $\{T_1, \ldots, T_k\}$ be the solution computed by the algorithm for a given instance $(\ell_{p\in[1,\infty)}, k)$ and $\hat{T}_1, \ldots, \hat{T}_k$ any other tree cover for $(\ell_{p\in[1,\infty)}, k)$. Further, let the trees be sorted non-increasingly by weight as $w(T_1) \geq w(T_2) \geq \ldots$ and $w(\hat{T}_1) \geq w(\hat{T}_2) \geq \ldots$. Then $\{T_1, \ldots, T_k\}$ is *strongly optimal* if for any j we have

$$\sum_{i=1}^{j} w(T_i) \leq \sum_{i=1}^{j} w(\hat{T}_i) \ .$$

Similarly, one can speak of *c-approximate strongly optimality* if for some $c \in \mathbb{R}_{\geq 1}$ we have

$$\sum_{i=1}^{j} w(T_i) \leq c \sum_{i=1}^{j} w(\hat{T}_i) \ .$$

This property was also considered as "global c-balance" by Goel and Meyerson [12].

Approximate strong optimality suffices for our purposes, since a solution that is c-approximate strongly optimal is also a c-approximation with respect to any p-norm; in fact, we obtain a stronger result here, since c-approximate strongly optimality implies a c-approximation with respect to *any* convex symmetric function, including all monotone symmetric norms. This fact is the backbone of many all-norm approximation algorithms, for example those by Golovin, Gupta, Kumar and Tangwongsan [13] for set cover variants. Thus, we will obtain a 24-approximation from this analysis not only for all p-norms, but for all monotone symmetric norms.

▶ **Lemma 15.** *Let $\{T_1, \ldots, T_k\}$ be the solution computed by the algorithm for a given instance $(\ell_{p\in[1,\infty)}, k)$, and let $\{\hat{T}_1, \ldots, \hat{T}_k\}$ be any other tree cover for $(\ell_{p\in[1,\infty)}, k)$, where $w(T_1) \geq w(T_2) \geq \ldots$ and $w(\hat{T}_1) \geq w(\hat{T}_2) \geq \ldots$. Then $\sum_{i=1}^{j} w(T_i) \leq 24 \sum_{i=1}^{j} w(\hat{T}_i)$ for $j = 1, \ldots, k$.*

Proof. We obtain an upper bound for the values $\sum_{i=1}^{j} w(T_i)$ by assuming that all trees which are not accounted for by the $k_{s,2}$ small component trees shared between the T_i and the $\hat{T}_i$ have weight exactly $4R$. This will ensure that the shared trees are $T_{k-k_{s,2}+1}, \ldots, T_k$. Similarly, we obtain a lower bound $\sum_{i=1}^{j} w(\hat{T}_i)$ by allowing a non-descending permutation of the $\hat{T}_i$. That is, we reorder the $\hat{T}_i$ such that the first $k - k_{s,2}$ trees are not the shared small component trees with the T_i.

It then follows directly that

$$\sum_{i=1}^{j} w(T_i) \leq j \cdot 4R = 24(j \cdot \frac{R}{6}) \leq 24 \sum_{i=1}^{j} w(\hat{T}_i) \text{ for } j \leq k - k_{s,2},$$

as well as

$$\sum_{i=1}^{k-k_{s,2}} w(T_i) + \sum_{i=1+k-k_{s,2}}^{j} w(T_i) \leq 24 \sum_{i=1}^{k-k_{s,2}} w(\hat{T}_i) + \sum_{i=1+k-k_{s,2}}^{j} w(\hat{T}_i)$$
$$\implies \sum_{i=1}^{j} w(T_i) \leq 24 \sum_{i=1}^{j} w(\hat{T}_i) \text{ for } j > k - k_{s,2} \ . \quad \blacktriangleleft$$

3.1 Ensuring k trees

Recall that the previous analysis of the approximation factor relies on the algorithm being able to find some R such that Step 3 holds, which is not generally true as per Figure 3. We now demonstrate how to modify the algorithm in such a way that this is avoided. Consider a list $e_1, \dots, e_m$ of the edges of G, such that $d(e_i) \leq d(e_j)$ if $i \leq j$, i.e., they are sorted by length with ties broken arbitrarily. Then we consider separately the graphs $G_j := (V, \{e_i \mid i \leq j\})$ for all $j = 0, \dots, m$ and try to find for each G_j an $R \in [d(e_{j-1}), d(e_j)]$ that is accepted by the algorithm, where we set $d(e_0) := 0$, and allow the larger interval $[d(e_m), \sum_i d(e_i)]$ for G_m. Note that the intervals can potentially only contain a single node, but they are not empty.

Now observe that for G_0 with $R = 0$, the algorithm would only accept this choice of R if $k = |V|$. Similarly, for G_m and $R = \sum_i d(e_i)$, the algorithm would require $k = 1$. We can then show that the k changes by at most one as we change R or keep R and move from G_j to G_{j+1}. Let $k(G_j, R)$ denote the value of k that would be accepted by the algorithm. Thus

▶ **Observation 16.** *It holds that* $k(G_{j-1}, d(e_j)) - k(G_j, d(e_j)) \leq 1$.

Proof. Between G_{j-1} and G_j, only the presence of e_j changes. If this does not change the connected components, we have $k(G_{j-1}, d(e_j)) = k(G_j, d(e_j))$. Otherwise, there is exactly one connected component C in G_j that is split into two parts C_1, C_2 by removing e_j. We then see that

$$\left\lfloor \frac{w(C_1) + 2d(e_j)}{2d(e_j)} \right\rfloor + \left\lfloor \frac{w(C_2) + 2d(e_j)}{2d(e_j)} \right\rfloor \leq 2 + \left\lfloor \frac{w(C_1)}{2d(e_j)} + \frac{w(C_2)}{2d(e_j)} \right\rfloor$$
$$\leq 1 + \left\lfloor \frac{w(C) + 2d(e_j)}{2d(e_j)} \right\rfloor \ . \quad \blacktriangleleft$$

To see that changing R by a sufficiently small amount also only changes $k(G_j, R)$ by at most one, consider that there are only finitely many critical nodes where $k(G_j, R)$ changes at all, and they can be computed in polynomial time. They are all of the form $R = w(C_i)/2\ell$ for some connected component C_i of G_j and $\ell \in 1, \dots, n$. At these nodes, $w(C_i)/2R$ will be an integer, so $\lfloor w(C_i)/2R \rfloor = 1 + \lfloor (w(C_i) - \varepsilon)/2R \rfloor$. If all critical nodes are pairwise different, we can just iterate over them to find some G_j and R with $k(G_j, R) = k$.

If on the other hand a node is critical for multiple different components, assign to each component a distinct weight which can be taken to be arbitrarily small, ensuring that all critical nodes are now different. In effect, this is equivalent to taking all components where $\frac{w(C_i)+2R}{2R}$ is an integer and allowing the expression $\lfloor \frac{w(T_i)+2R}{2R} \rfloor$ to also take the value $\frac{w(T_i)}{2R}$. One may check that this does not impact the analysis, since in the analysis we only need that each component has a minimum spanning tree of weight at least $\lfloor \frac{w(T_i)}{2R} \rfloor \cdot 2R$. However, for legibility reasons we will suppress this and generally assume that some R can be found with $k(G_j, R) = k$. Combining with Theorem 14, this completes the proof of Theorem 2.

4 All-norm tree cover problem with depots

The algorithm for the All-Norm Tree Cover problem *with* depots is considerably more involved, in particular because the simple lower bound for the depot-less algorithm of assuming an even distribution of the total weight can no longer work: it might be necessary to have unbalanced cluster sizes, for instance if some depots are extremely far from all nodes.

Instead our algorithm constructs a c-approximately (here, c is a constant) strongly optimal solution by also maintaining some (implicit) evidence that the optimum solution contains large trees if it decides to create a large tree itself. More concretely we do the following:

1. First, we partition the node set V into layers L_i such that all nodes in L_i have distance between 2^{i-1} and 2^i to the depots.
2. Then we consider separately the nodes in the odd and even layers; this implies that nodes in different layers have a large distance to each other. Computing separate solutions for these two subinstances loses at most a factor 2 in the approximation factor due to Lemma 9.
3. Next, we partition the nodes in each layer L_i using the non-depot algorithm with $R = 2^i$. This yields a collection of subtrees in L_i such that the cost of connecting such a subtree to its nearest depot is in $\Theta(2^i)$. This allows us to treat them basically identically, loosing only the constant in the Θ. Indeed, we can show explicitly that this prepartitioning into subtrees can be assumed to be present in an optimal solution, up to a constant factor increase in the weight of each tree. For the full reasoning refer to the full version on arXiv [18].
4. To assign these trees to the depots, we iteratively maintain an estimate of the largest tree necessary in any solution as 2^i. We then collect all trees of weight $\Theta(2^i)$ and compute a maximum matching between them and the depots at distance $\Theta(2^i)$ to them. All unmatched trees are then combined to form trees of weight $\Theta(2^{i+1})$, and we update our estimate to 2^{i+1}.

To analyse the output of this algorithm, it will suffice to show that the estimate was correct up to a constant factor. That is, if in round i some trees (suppose k_i trees) of weight 2^i were assigned, we will be able to prove that any tree-cover solution must also have some family of at most k_i trees with total weight at least $k_i \cdot 2^i$.

From this we are then able to conclude c-approximate strong optimality for some value of $c < 10^6$. This is a rather large upper bound on the approximation factor, however, we conjecture that the actual constant achieved by the algorithm is much smaller. For the purposes of a clean presentation, we did not try to optimize the constant. The complete proof can be found in the full version on arXiv [18].

5 Computational Hardness

To complement our algorithmic results, we establish hardness results for all-norm tree cover problems and discuss on what kind of algorithmic improvements (to our algorithms) are potentially possible in light of these complexity results. Specifically, we show:

1. For any $p \in (1, \infty]$, problem ℓ_p-Tree Cover with Depots is weakly NP-hard, even with only 2 trees.
2. For any $p \in (1, \infty]$, problem ℓ_p-Tree Cover with Depots with k trees is (strongly) W[1]-hard parameterized by depot size $|D|$.
3. For any $p \in (1, \infty]$ there exists some $\varepsilon > 0$ such that ℓ_p-Tree Cover with Depots is NP-hard to approximate within a factor $< 1 + \varepsilon$ under randomized reductions.

Results 1 and 2 were already essentially presented by Even et al. [10], but we restate them for completeness. Note that, given these complexity results, numerous otherwise desirable algorithmic outcomes become unattainable. For instance, there cannot be polynomial-time approximation schemes for ℓ_p-Tree Cover with Depots, nor can we expect fixed-parameter algorithms (parameterized by the number of depots) finding optimal solutions, unless established hardness hypotheses (P≠NP, FPT $\neq$ W[1]) fail. Further, the reduction of Item 1 yields bipartite graphs of tree-depth 3, so parameterization by the structure of the graph supporting the input metric also appears out of reach.

The result in Item 3 follows by a direct gadget reduction from Max-Sat for 3-ary linear equations modulo 2 (MAX-E3LIN2), which was shown to be APX-hard by Håstad [15]. We show that one can transform such equations into an instance of ℓ_p-Tree Cover with Depots where every unsatisfied equation will correspond roughly to a tree of above average weight. This allows us to recover approximately the maximum number of simultaneously satisfiable constraints of such systems of equations.

Formally, we reduce from $3R\{2,3\}L2$, a modification of MAX-E3LIN2 where every variable occurs in exactly 3 equations, and for which hardness of approximation was shown by Karpinski et al. [17]. From an instance of 3R{2,3}L2 we construct an instance of ℓ_p-Tree Cover with Depots by introducing gadgets for the variables and clauses:

- For every variable x, introduce three nodes $\hat{x}, x_0$, and x_1, as well as edges $\hat{x}, x_i$ for $i = 0, 1$ with weight 3. Add the x_i's as depots.
- For every ternary clause C, introduce nodes $\hat{C}, C_{000}, C_{110}, C_{101}$, and C_{011} with edges $\{\hat{C}, C_i\}$ of weight 3. Add the C_i's as depots.
- For every binary clause $C = x \oplus y = 0$, introduce nodes $\hat{C}, C_{00}$, and C_{11} with edges $\{\hat{C}, C_i\}$ of weight 2. Add the C_i's as depots. Further add nodes $\hat{C_{00}}$ and $\hat{C_{11}}$ where $\hat{C_i}$ is connected only to C_i by an edge of weight 1.
- For every binary clause $C = x \oplus y = 1$, introduce nodes $\hat{C}, C_{01}$, and C_{10} with edges $\{\hat{C}, C_i\}$ of weight 2. Add the C_i's as depots. Further add nodes $\hat{C_{01}}$ and $\hat{C_{10}}$ where $\hat{C_i}$ is connected only to C_i by an edge of weight 1.

We connect the gadgets as follows:

- For every clause $C = x \oplus y \oplus z = 0$ we connect $C_{b_1 b_2 b_3}$ to x_{b_1}, y_{b_2}, and z_{b_3} with a path of length 2 where both edges have weight 1.
- For every clause $C = x \oplus y = b$ we connect $C_{b_1 b_2}$ to x_{b_1}, y_{b_2} with a path of length 2 where both edges have weight 1.

Intuitively, there is a depot for every way a clause could be satisfied, and the depots for a clause have a joint neighbour that is expensive to connect. The depot absorbing this neighbour should correspond to the way in which the clause is satisfied. Similarly there are two depots for each variable representing the two possible assignments to the variable. In this case the depot that does not get assigned the joint neighbour corresponds to the chosen assignment.

There are then additional vertices between the clause and vertex depots which can be assigned to the clause depots, unless the adjacent clause depot corresponds to the satisfying assignment of that clause. In that case they need to be assigned to an adjacent variable depot, which is to say the variables of the clause will have to be assigned in such a way that the clause is satisfied.

One can quickly check that a satisfying assignment to the clauses will allow us to compute a tree cover where every tree has size 3. Meanwhile every non-satisfied clause will push at least one unit of excess weight to a tree of size at least 3. Quantifying this relationship then permits us to compute approximately the maximum number of satisfiable clauses in such a system of equations.

6 Computational Experiments

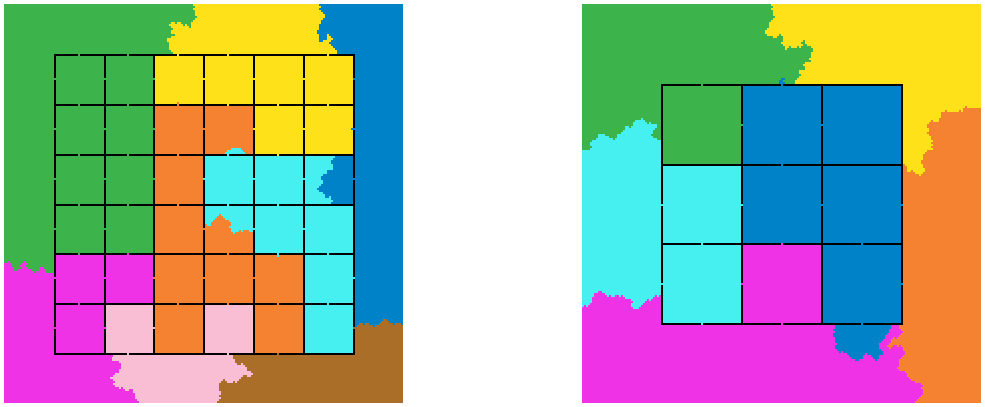
To illustrate the practical performance achievable by our clustering algorithms, we implemented the algorithm from Section 3 for the setting without depots in C++ and tested it on instances proposed by Zheng et al. [24]. Those instances model real-world terrain, which is to be partitioned evenly so that a fixed number robots can jointly traverse it. They consist of a grid where either random cells are set to be obstacles, i.e., inaccessible, or a grid-like arrangement of rooms is superimposed with doors closed at random. A metric is induced on the accessible cells of the grid by setting the distance of neighbouring cells to be 1.

For all instances on grids of size 200 by 200 (ca. 40,000 nodes), our implementation was able to compute a clustering in less than 200ms on an Intel i5-10600K under Windows with 48GB of available memory (although actual memory usage was negligible). The resulting partitions are illustrated in Figure 5. Note that we make two small heuristic changes to the original algorithm, which are, however, not amenable to be analyzed formally:

First, we adapt the partitioning of the large trees in Step 5 to try to cut the trees into subtrees of weight at least $2R$ rather than $4R$. Note that the choice of $4R$ captures a worst-case scenario where the instance contains edges of size almost R that are to be included in a solution. If the heaviest edges are considerably smaller than R, the necessary cutoff approaches $2R$ rather than $4R$. Thus, we run the partitioning algorithm with $2R$ rather than $4R$, and resort to the higher cutoff only in the case that this fails. Yet, for the considered instances this was not necessary.

Second, we post-process the computed solution to ensure that it has exactly k components. It is in principle possible that the algorithm computes a solution with fewer than k trees; in this case, we iteratively select the largest tree and split it into two parts of as similar a size as possible until we obtain exactly k components.

The clusterings obtained in this way in Figure 5 are all at least 3-approximately strongly optimal when compared to a hypothetical solution that distributes the total weight perfectly evenly on the k clusters.



(a) Output of the algorithm on a 200×200 grid with walls partitioned into 8 clusters. The cluster sizes are $2433, 5516, 9528, 4550, 5271, 3985, 2482$ and 4240; consequently the solution is at least 2.01-approximately strongly optimal.

(b) Output of the algorithm on a 200×200 grid with walls partitioned into 6 clusters. The cluster sizes are $7993, 4774, 6390, 8004, 5267$, and 6638; consequently, the solution is at least 1.61-approximately strongly optimal.

Figure 5 Visualizations of the results of our implementation of the non-depot tree cover algorithm. Inaccessible sections of the grid are marked in black; other colors represent the computed clusters.

References

1 Sara Ahmadian, Babak Behsaz, Zachary Friggstad, Amin Jorati, Mohammad R Salavatipour, and Chaitanya Swamy. Approximation algorithms for minimum-load k-facility location. *ACM Trans. Algorithms*, 14(2):1–29, 2018. doi:10.1145/3173047.

2 Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k-means and Euclidean k-median by primal-dual algorithms. *SIAM J. Comput.*, 49(4):FOCS17–97, 2019. doi:10.1137/18M1171321.

3 Noga Alon, Yossi Azar, Gerhard J. Woeginger, and Tal Yadid. Approximation schemes for scheduling on parallel machines. *J. Sched.*, 1(1):55–66, 1998. doi:10.1002/(SICI)1099-1425(199806)1:1<55::AID-JOS2>3.0.CO;2-J.

4 Cristina Bazgan, Refael Hassin, and Jérôme Monnot. Approximation algorithms for some vehicle routing problems. *Discrete Appl. Math.*, 146(1):27–42, 2005. doi:10.1016/J.DAM.2004.07.003.

5 Tolga Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209–219, 2006. doi:10.1016/j.omega.2004.10.004.

6 Mandell Bellmore and Saman Hong. Transformation of multisalesman problem to the standard traveling salesman problem. *J. ACM*, 21(3):500–504, 1974. doi:10.1145/321832.321847.

7 Jarosław Byrka, Thomas Pensyl, Bartosz Rybicki, Aravind Srinivasan, and Khoa Trinh. An improved approximation for k-median and positive correlation in budgeted optimization. *ACM Trans. Algorithms*, 13(2):1–31, 2017. doi:10.1145/2981561.

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

9 Sami Davies, Benjamin Moseley, and Heather Newman. Fast combinatorial algorithms for min max correlation clustering. In *Proc. ICML 2023*, pages 7205–7230, 2023.

10 Guy Even, Naveen Garg, Jochen Könemann, Ramamoorthi Ravi, and Amitabh Sinha. Min–max tree covers of graphs. *Oper. Res. Lett.*, 32(4):309–315, 2004. doi:10.1016/J.ORL.2003.11.010.

11 Greg N. Frederickson, Matthew S. Hecht, and Chul E. Kim. Approximation algorithms for some routing problems. In *Proc. SFCS 1976*, pages 216–227, 1976. doi:10.1109/SFCS.1976.6.

12 Ashish Goel and Adam Meyerson. Simultaneous optimization via approximate majorization for concave profits or convex costs. *Algorithmica*, 44:301–323, 2006. doi:10.1007/S00453-005-1177-7.

13 Daniel Golovin, Anupam Gupta, Amit Kumar, and Kanat Tangwongsan. All-norms and all-ℓ_p-norms approximation algorithms. In *Proc. FSTTCS 2008*, volume 2 of *Leibniz Int. Proc. Informatics*, pages 199–210, 2008. doi:10.4230/LIPICS.FSTTCS.2008.1753.

14 Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. *J. Algorithms*, 31(1):228–248, 1999. doi:10.1006/JAGM.1998.0993.

15 Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001. doi:10.1145/502090.502098.

16 Sharat Ibrahimpur and Chaitanya Swamy. Approximation algorithms for stochastic minimum-norm combinatorial optimization. In *Proc. FOCS 2020*, pages 966–977, 2020. doi:10.1109/FOCS46700.2020.00094.

17 Marek Karpinski, Michael Lampis, and Richard Schmied. New inapproximability bounds for TSP. *J. Comput. Syst. Sci.*, 81(8):1665–1677, 2015. doi:10.1016/J.JCSS.2015.06.003.

18 Matthias Kaul, Kelin Luo, Matthias Mnich, and Heiko Röglin. Approximate minimum tree cover in all symmetric monotone norms simultaneously, 2025. doi:10.48550/arXiv.2501.05048.

19 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.

20 L. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta Informatica*, 15(2):141–145, 1981. doi:10.1007/BF00288961.

21 Shi Li. A 1.488 approximation algorithm for the uncapacitated facility location problem. *Inf. Comput.*, 222:45–58, 2013. doi:10.1016/J.IC.2012.01.007.

22 Hiroshi Nagamochi. Approximating the minmax rooted-subtree cover problem. *IEICE Trans. Fund. Electr., Comm. Comp. Sci.*, 88(5):1335–1338, 2005. doi:10.1093/IETFEC/E88-A.5.1335.

23 Zhou Xu and Qi Wen. Approximation hardness of min–max tree covers. *Oper. Res. Lett.*, 38(3):169–173, 2010. doi:10.1016/J.ORL.2010.02.004.

24 Xiaoming Zheng and Sven Koenig. Robot coverage of terrain with non-uniform traversability. In *Proc. IEEE/RSJ Intl. Conf. Intell. Robots Syst.2007*, pages 3757–3764, 2007. doi:10.1109/IROS.2007.4399423.