Balanced Partitions of Trees and Applications *

Andreas Emil Feldmann[1] and Luca Foschini[2]

1 **Institute of Theoretical Computer Science, ETH Zürich**
Zürich, Switzerland
feldmann@inf.ethz.ch
2 **Department of Computer Science, U.C. Santa Barbara**
Santa Barbara, USA
foschini@cs.ucsb.edu

Abstract

We study the k-BALANCED PARTITIONING problem in which the vertices of a graph are to be partitioned into k sets of size at most $\lceil n/k \rceil$ while minimising the *cut size*, which is the number of edges connecting vertices in different sets. The problem is well studied for general graphs, for which it cannot be approximated within any factor in polynomial time. However, little is known about restricted graph classes. We show that for trees k-BALANCED PARTITIONING remains surprisingly hard. In particular, approximating the cut size is APX-hard even if the maximum degree of the tree is constant. If instead the diameter of the tree is bounded by a constant, we show that it is NP-hard to approximate the cut size within n^c, for any constant $c < 1$.

In the face of the hardness results, we show that allowing *near-balanced* solutions, in which there are at most $(1+\varepsilon)\lceil n/k \rceil$ vertices in any of the k sets, admits a PTAS for trees. Remarkably, the computed cut size is no larger than that of an optimal balanced solution. In the final section of our paper, we harness results on embedding graph metrics into tree metrics to extend our PTAS for trees to general graphs. In addition to being conceptually simpler and easier to analyse, our scheme improves the best factor known on the cut size of near-balanced solutions from $O(\log^{1.5}(n)/\varepsilon^2)$ [Andreev and Räcke TCS 2006] to $O(\log n)$, for weighted graphs. This also settles a question posed by Andreev and Räcke of whether an algorithm with approximation guarantees on the cut size independent from ε exists.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases balanced partitioning, bicriteria approximation, hardness of approximation, tree embeddings

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.100

1 Introduction

In this paper we study the k-BALANCED PARTITIONING problem, which asks for a partition of the n vertices of a graph into k sets of size at most $\lceil n/k \rceil$ each, such that the number of edges connecting vertices in different sets, called the *cut size*, is minimised. The problem has numerous applications of which one of the most prominent is in the field of parallel computing. There, it is crucial to evenly distribute n tasks (vertices) among k processors (sets) while minimising the inter-processor communication (edges between different sets), which constitutes a bottleneck. Other applications can be found in the design of electronic circuits and sparse linear solvers. However, despite the broad applicability, k-BALANCED

* The first author is supported by the Swiss National Science Foundation under grant 200021_125201/1. The second author is supported by the National Science Foundation grant IIS 0904501.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 100–111

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


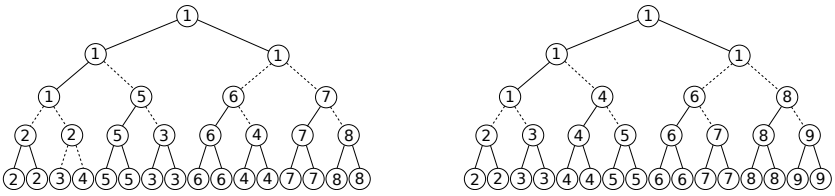

Figure 1 Two optimally partitioned binary trees. For the tree on the left $k = 8$ (with a cut size of 10) whereas $k = 9$ (with a cut size of 8) for the tree on the right. The numbers in the vertices indicate the set they belong to and the cut set is represented by the dashed edges.

PARTITIONING is a notoriously hard problem. The special case of $k = 2$, commonly known as the BISECTION problem, is already NP-hard [11]. For this reason, approximation algorithms that find a balanced partition with a cut size larger than optimal have been developed. We follow the convention of denoting the approximation ratio on the cut size by α.

Unfortunately, when k is not constant even finding an approximation of the minimum balanced cut still remains infeasible, as it is known that no finite approximation for the cut size can be computed in polynomial time, unless P=NP [1]. In order to overcome this obstacle, relaxing the balance constraints has proven beneficial. By that we mean that the sets of the partitions are allowed to be of size at most $(1 + \varepsilon)\lceil n/k \rceil$ for some factor $\varepsilon \geq 0$. Along these lines, *bicriteria approximation* algorithms have been proposed, which approximate both the balance and the cut size of the optimal solution. That is, the computed cut size is compared to the optimal cut size of a *perfectly balanced* partition in which $\varepsilon = 0$.

The k-BALANCED PARTITIONING problem has received some attention for the case $\varepsilon \geq 1$, that is when a perfect balance is approximated within a factor of two. For this case, the best result is by Krauthgamer *et al.* [15] who give an algorithm with approximation factor $\alpha \in \mathcal{O}(\sqrt{\log n \log k})$. However, it is not hard to imagine how the slack on the balance can be detrimental in practical applications. In parallel computing, for instance, a factor of two on the balance in the workload assigned to each machine can result in a factor of two slowdown, since the completion time is solely determined by the overloaded machines.

Therefore the case of $\varepsilon < 1$, that is when *near-balanced* partitions are allowed, is of greater practical interest. No progress has been made on near-balanced partitions since Andreev and Räcke [1] gave an algorithm with $\alpha \in \mathcal{O}(\log^{1.5}(n)/\varepsilon^2)$—a significantly worse bound than the one for $\varepsilon \geq 1$. This is not surprising since, as argued in [1], as ε approaches 0 and the constraint on the balance becomes more stringent the k-BALANCED PARTITIONING problem starts bearing more resemblance to a packing problem than to a partitioning problem. One direct side effect is that the spreading metric relaxations developed for $\varepsilon \geq 1$ [5, 15] do not extend to near balanced partitions. This is because such strategies aim at breaking the graph into components of size less than n/k while minimizing the cut, only to later rely on the fact that pieces of that size can be packed into k partitions such that no partition exceeds $2n/k$ vertices. However, when near-balanced solutions are required, the partition phase can no longer be oblivious of the packing. So it is necessary to combine the partition step with an algorithm to pack the pieces into near-balanced partitions.

Our Contribution. As argued above, the restriction to near-balanced partitions poses a major challenge in understanding the structure of k-BALANCED PARTITIONING. For this reason, we consider the simplest non-trivial instance class of the problem, namely connected trees. Figure 1 gives an example of how balanced partitions exhibit a counter-intuitive behaviour even on *perfect binary trees*, as increasing k does not necessarily entail a larger cut size. Our results confirm this intuition when a perfectly balanced solution is required: adapting an argument by Andreev and Räcke [1], we show that it is NP-hard to approximate

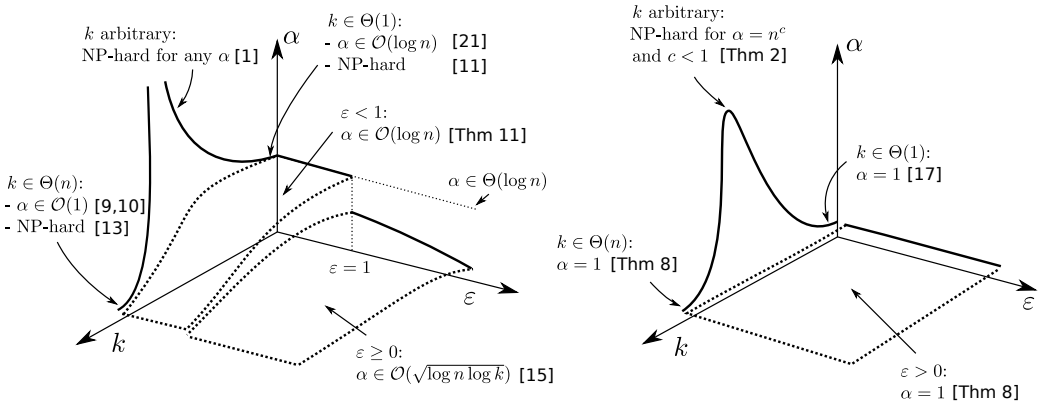

Figure 2 Illustrations of best approximation factor α known against k and ε, for general graphs (left) and trees (right). The plane (α, k) represents the case of perfectly balanced solutions ($\varepsilon = 0$) and shows that the restriction to trees does not significantly change the asymptotic behaviour. However, for $\varepsilon > 0$ much better approximations can be devised for trees. This remarkable behaviour can be partially adapted to general graphs via tree decompositions, allowing us to reduce the gap between the case $\varepsilon < 1$ and $\varepsilon \geq 1$ visible in the plot on the left.

the cut size within any factor better than $\alpha = n^c$ for any constant $c < 1$. (Figure 2 summarises the results presented here together with the related work.) This is asymptotically tight, since a trivial approximation algorithm can achieve a ratio of n by cutting all edges of the tree. Interestingly, the lower bound remains true even if the diameter of the tree (i.e. the length of the longest path between any two leaves) is restricted to be at most 4, while instances of diameter at most 3 are polynomially solvable.

By a substantially different argument, we show that a similar dichotomy arises when parametrizing the complexity with respect to the maximum degree Δ. For trees with $\Delta = 2$ (i.e. paths) k-`BALANCED PARTITIONING` is trivial. However, if $\Delta = 5$ the problem becomes NP-hard and with $\Delta = 7$ we show it is APX-hard. Finding where exactly the dichotomy arises, i.e. the $\Delta \in \{3, 4\}$ at which k-`BALANCED PARTITIONING` becomes hard, is an interesting open problem. These results should be contrasted with a greedy algorithm by MacGregor [17] to find tree bisections that can be extended to find perfectly balanced partitions for k sets with $\alpha \in \mathcal{O}(\log(n/k))$, for trees of constant degrees.

On the positive side, we show that when near-balanced solutions are allowed, trees behave substantially better than general graphs. We present an algorithm that computes a near-balanced partition for any constant $\varepsilon > 0$ in polynomial time, achieving a cut size no larger than the optimal for a perfectly balanced partition, i.e. $\alpha = 1$. In this sense the presented algorithm is a PTAS w.r.t. the balance of the computed solution. In addition, our PTAS can be shown to yield an optimal *perfectly balanced* solution for trees if $k \in \Theta(n)$, while on general graphs the problem is NP-hard for these values of k [13].

In the last section of our paper we capitalise on the PTAS for trees to tackle the k-`BALANCED PARTITIONING` problem on general, weighted graphs. By embedding a graph into a collection of trees with a cut distortion of $\mathcal{O}(\log n)$ we can use our PTAS for trees to get a solution for graphs. Since the PTAS has approximation factor $\alpha = 1$, the total approximation factor paid for the general graphs is due only to the distortion of the embedding, that is $\alpha \in \mathcal{O}(\log n)$. Note that since the graph is decomposed into trees as a preliminary step, the decomposition is oblivious of the balance constraints related to solving k-`BALANCED PARTITIONING` on the individual trees, hence the distortion does not depend on ε. This is sufficient to simultaneously improve on the previous best result known [1]

of $\alpha \in \mathcal{O}(\log^{1.5}(n)/\varepsilon^2)$, and answers an open question posed in the same paper whether an algorithm with no dependence on ε in the ratio α exists. In addition, our analysis is significantly simpler than the one in [1]: the ad-hoc approach of [1] must deal directly with the complications introduced by considering general graphs. We are able to minimise those by relying on the powerful tool of tree decompositions.

Related Work. We extend the results in [1], where it is shown that for general graphs approximating the cut size of the k-`BALANCED PARTITIONING` problem is NP-hard for any finite factor α, if perfectly balanced partitions are needed. In [1] the authors also give a bicriteria approximation algorithm with $\alpha \in \mathcal{O}(\log^{1.5}(n)/\varepsilon^2)$ when solutions are allowed to be near-balanced. If more unbalance is allowed, for $\varepsilon \geq 1$ Even *et al.* [5] present an algorithm with $\alpha \in \mathcal{O}(\log n)$ using spreading metrics techniques. Later Krauthgamer *et al.* [15] improved the result to $\alpha \in \mathcal{O}(\sqrt{\log n \log k})$ using a semidefinite relaxation which combines l_2 metrics with spreading metrics. To the best of our knowledge, the only result on restricted graph classes is for graphs with excluded minors (such as planar graphs). By applying a spreading metrics relaxation and the results in [14][1] it is possible to compute near-balanced solutions with $\alpha \in \mathcal{O}(1)$.

The special case when $k = 2$, commonly known as the `BISECTION` problem, is well studied. It is NP-hard in the general case [11] but approximation algorithms are known. For instance Räcke [21] gives an algorithm with approximation ratio $\alpha \in \mathcal{O}(\log n)$ for perfectly balanced partitions. For near-balanced partitions Leighton and Rao [16] show how to compute a solution using *min-ratio cuts*. In this solution the cut size is approximated within $\alpha \in \mathcal{O}(\gamma/\varepsilon)$, where γ is the approximation factor of computing a min-ratio cut. In [16] it was shown that $\gamma \in \mathcal{O}(\log n)$, and this result was improved [2] to $\gamma \in \mathcal{O}(\sqrt{\log n})$. For (unweighted) planar graphs it is possible to compute the optimum min ratio cut in polynomial time [19]. If a perfectly balanced solution to `BISECTION` is required for planar graphs, Dìaz *et al.* [4] show how to obtain a PTAS. Even though it is known that the `BISECTION` problem is weakly NP-hard on planar graphs with vertex weights [19], whether it is NP-hard on these graphs in the unweighted case is unknown. For other special graph classes the problem can be solved optimally in polynomial time [3]. For instance an $\mathcal{O}(n^4)$ algorithm for grid graphs without holes has been found [8], while for trees an $\mathcal{O}(n^2)$ algorithm [17] exists.

In addition to the case $k = 2$, some results are known for other extreme values of k. For trees the above mentioned bisection algorithm by MacGregor [17] is easily generalised to solve the k-`BALANCED PARTITIONING` problem for any constant k in polynomial time. At the other end of the spectrum, i.e. when $k \in \Theta(n)$, it is known that the problem is NP-hard [13] for any $k \leq n/3$ on general graphs. Feo and Khellaf [10] give a $\alpha = n/k$ approximation algorithm for the cut size which was improved [9] to $\alpha = 2$ in case k equals $n/3$ or $n/4$.

We complete the review of related work by discussing the literature on tree decompositions, which we leverage in our algorithm for general graphs. Informally a tree decomposition of a graph G is a set of trees for which the leaves correspond to the vertices of G, and for which the structure of their cuts approximate the cuts in G. Tree decompositions have been studied in the context of oblivious routing schemes (see [18] for a survey). In [21], Räcke introduces an optimal decomposition with factor $\mathcal{O}(\log n)$, which we employ in the present work. In a recent work, Madry [18] shows that it is possible to generalise Räcke's insights so that any *cut based* problem (see [18] for more details) is solvable on graphs by computing solutions on tree decompositions of the input graph.

[1] We thank an anonymous reviewer for pointing out this folklore result.

2 The Hardness of Computing Perfectly Balanced Partitions

We now consider the problem of finding a perfectly balanced partition of a tree with minimum cut size. We prove hardness results in the case where either the diameter or the degrees are restricted to be constant. All reductions are from MAX-3-PARTITION, defined as follows.

▶ **Definition 1** (MAX-3-PARTITION)**.** Given $3k$ integers $a_1, \ldots, a_{3k}$ and a threshold s, such that $s/4 < a_i < s/2$ and $\sum_{i=1}^{3k} a_i = ks$, find the maximum number of disjoint triples of a_1 to a_{3k} such that each triple sums up to exactly s.

The MAX-3-PARTITION problem is APX-hard, i.e. for some constant ρ it is NP-hard to decide whether all k integers or at most k/ρ of them can be partitioned into triples that sum up to exactly s. This is true even if all integers are polynomially bounded in k, which can be shown using the standard reduction for the corresponding decision problem in [11] and the results on the 3D-MATCHING problem by Petrank [20] [2].

We begin by showing that an approximation algorithm with factors $\alpha = n^c$ and $\varepsilon = 0$, for any constant $c < 1$, for k-BALANCED PARTITIONING on trees could be used to find the optimal solution to an instance of MAX-3-PARTITION. We do not rely on the APX-hardness of the latter problem for the proof, but only on the NP-hardness of the corresponding decision problem. The idea for the reduction is similar to the one used by Andreev and Räcke [1] for general graphs. The result holds even if the diameter of the tree is bounded by a constant[3].

▶ **Theorem 2.** *The k-BALANCED PARTITIONING problem on trees has no polynomial time approximation algorithm with approximation factors $\alpha = n^c$ and $\varepsilon = 0$, for any constant $c < 1$, even if the diameter is at most 4, unless P=NP.*

The reduction in the proof of the above theorem relies on the fact that the degree of the tree is unbounded. Hence a natural question arising is what the complexity of bounded degree trees is. As we will show next the problem remains surprisingly hard when bounding the degree. We are able to prove two hardness results for this case. First we show that the problem of finding a perfectly balanced partition of a tree is APX-hard even if the maximum degree of the tree is at most 7. To prove this result we use a gap-preserving reduction from MAX-3-PARTITION. In particular this means that a substantially different (and more involved) technique than the one used to prove Theorem 2 has to be employed: rather than relying on the fact that many edges have to be cut in a gadget consisting of a high degree star, we need gadgets with structural properties that guarantee a number of cut edges proportional to the number of integers that cannot be packed into triples in a MAX-3-PARTITION instance.

▶ **Theorem 3.** *Unless P=NP, there exists a constant ρ such that the minimum cut size of a perfectly balanced partition on trees with maximum degree Δ cannot be approximated in polynomial time within $\alpha = 1 + (1 - \rho^{-1})/24$ and $\varepsilon = 0$ even if Δ is at most 7.*

Proof. Given an instance of MAX-3-PARTITION with polynomially bounded integers a'_i, consider the instance I where $a_i = 12a'_i$. Obviously all hardness properties are preserved by this transformation. As a consequence all integers are divisible by 4 and $s > 20$, which will become important later in the proof. For each a_i in I, construct a gadget T_i composed by a path on a_i vertices (called a_i-path) connected to the root of a tree on s vertices (called

[2] We thank Nikhil Bansal for pointing out the connection between the reductions in [11] and the results by Petrank [20]

[3] All missing proofs of this paper can be found in the full version [7].

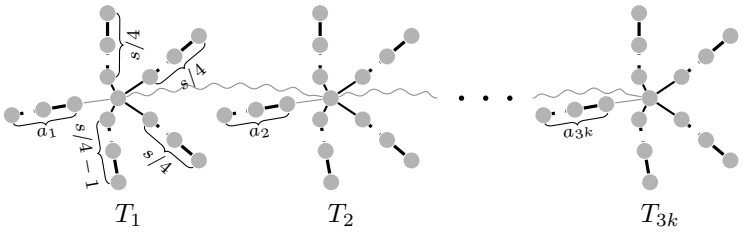

Figure 3 Construction for Theorem 3. Each gadget T_i is composed by an a_i-path connected to the root of an s-tree through an edge from A (straight grey). Each s-tree branches into four paths of (almost) the same length. Two adjacent gadgets in a path are connected through the roots of their s-trees with an edge from B (wiggled grey).

s-tree). The root of the s-tree branches into four paths, three of them with $s/4$ vertices each, and one with $s/4-1$ vertices. The construction is completed by connecting the roots of the s-trees in a path, as shown in Figure 3. We call B the set of edges connecting different T_is and A the set of edges connecting an a_i-path with the corresponding s-tree in each T_i.

At a high level, we set out to prove that if all k integers in I can be partitioned into triples that sum up to exactly s, then the constructed tree T can be split into a $4k$-balanced partition with cut size $6k-1$. If however at most a ρ fraction of the integers can be partitioned in this way, T requires at least $(1-\rho^{-1})k/4$ additional cut edges. This means that a polynomial time algorithm computing an approximate $4k$-balanced partition for T within factor $\frac{6k-1+(1-\rho^{-1})k/4}{6k-1} \geq 1+(1-\rho^{-1})/24$ of the optimum cut size could approximate the `MAX-3-PARTITION` problem within the ratio ρ in polynomial time. Hence the theorem follows.

It is easy to see that if all k integers of I can be partitioned into triples of size exactly s, cutting exactly the $6k-1$ edges in A and B suffices to create a valid $4k$-balanced partition. It remains to be shown that $(1-\rho^{-1})k/4$ additional edges are required in the other case. Let C^* be an optimal cut set of a $4k$-balanced partition in T when at most k/ρ many integers can be partitioned into triples of size exactly s in I. We argue that by incrementally repositioning cut edges from the set $C := C^* \setminus (A \cup B)$ to edges in $(A \cup B) \setminus C^*$, eventually all the edges in $A \cup B$ will be cut. However, the following lemma implies that a constant fraction of the edges initially in C will not be moved. We will then argue that the more triples of I can not be packed into triples of size s, the more edges are left in C. Thus the more edges must additionally have been in C compared to those in $A \cup B$. We rely on the following lemma.

▶ **Lemma 4.** *If $s > 20$ then $|C| \geq 2|(A \cup B) \setminus C^*|$.*

Consider the following algorithm $\mathcal{A}$ which repositions cut edges from a $4k$-balanced partition and thereby computes an approximation to `MAX-3-PARTITION`. As long as there is an uncut edge $e \in A \cup B$, $\mathcal{A}$ removes a cut edge in C and cuts e instead. At the end of the process, when all edges in $A \cup B$ are cut, $\mathcal{A}$ removes the set of cut edges left in C denoted by C'. Then $|C'|$ is the number of additional edges cut in the case at most a ρ fraction of the integers can be partitioned into triples of size exactly s. When repositioning a cut edge from C to $A \cup B$, or when removing a cut from C', $\mathcal{A}$ modifies the sizes of the sets in the partition induced by the cut set, and the balance might be lost. In particular, when a cut edge $e \in C$ is removed, the algorithm will join the two connected components induced by the cut set and incident to e to form a single component. The algorithm will then include it in an arbitrary one of the sets that contained the two components. This changes the sizes of at most two sets in the partition. When a new cut is introduced by $\mathcal{A}$, a component is split into two and the two newly created components are retained in the same set, thus no set size is changed.

By Lemma 4 there are at least as many edges in C', as there are edges that are repositioned from C to $A \cup B$. Since each edge from C repositioned by $\mathcal{A}$ causes at most two changes in set sizes, the total number of set size changes performed by $\mathcal{A}$ amortized on the removed edges in C' is therefore at most $4|C'|$.

When $\mathcal{A}$ terminates only edges from $A \cup B$ are cut. Therefore the remaining connected components correspond to the $3k$ integers a_i of I in addition to $3k$ integers equal to s. Since at most a ρ fraction of the k integers in I can be partitioned into triples of size exactly s, and the elements of size s can be ignored, at least $(1-\rho^{-1})k$ of the sets of the resulting partition do not have size exactly s. This means that $\mathcal{A}$ must have changed the size of at least $(1-\rho^{-1})k$ sets, since it converted a $4k$-balanced partition into a solution to `MAX-3-PARTITION` with at least $(1-\rho^{-1})k$ unbalanced sets. This finally implies that $4|C'| \geq (1-\rho^{-1})k$, which concludes the proof since $|C'|$ is the number of additional cuts required. ◀

Using a similar argument as in the above proof, if we restrict the degree to be at most 5 we can still show that the problem remains NP-hard. For this we use a slightly different construction than the one shown in Figure 3: instead of connecting the s-trees through their roots, the B edges connect the leaves of the shortest branches of the s-trees. It is then possible to show that exactly the $6k-1$ edges in A and B are cut if all k integers in I can be partitioned into triples of size exactly s, while otherwise at least $6k$ edges are cut. Since the `MAX-3-PARTITION` problem is NP-hard this suffices to establish the following result.

▶ **Theorem 5.** *The k-`BALANCED PARTITIONING` problem on trees has no polynomial time algorithm even if the maximum degree is at most 5, unless P=NP.*

3 Computing Near-Balanced Partitions

The previous section shows that approximating the cut size of k-`BALANCED PARTITIONING` is hard even on trees if perfectly balanced partitions are desired. We showed that for the general case when the degree is unbounded there is no hope for a polynomial time algorithm with non-trivial approximation. Therefore, in this section we study the complexity of the problem when allowing the partitions to deviate from being perfectly balanced. In contrast to the negative results presented so far, we prove the existence of a PTAS for k-`BALANCED PARTITIONING` on trees that computes near-balanced partitions but returns a cut size no larger than the optimum of a perfectly balanced partition.

Conceptually one could find a *perfectly balanced* partition of a tree T with minimum cut size in two steps. First all the possible ways of cutting T into connected components are grouped into equivalence classes based on the sizes of their components. That is, the sets of connected components $\mathcal{S}$ and $\mathcal{S}'$ belong to the same equivalence class if they contain the same number of components of size x for all $x \in \{1, \ldots, \lceil n/k \rceil\}$. In a first step the set of connected components that achieves the cut of minimum size for each class is computed and set to be the representative of the class. In a second stage only the equivalence classes whose elements can be packed into k sets of size at most $\lceil n/k \rceil$ are considered, and among those the representative of the class with minimum cut size is returned. Clearly such an algorithm finds the optimal solution to the k-`BALANCED PARTITIONING` problem, but the runtime is exponential in n as the total number of equivalence classes is exponential. The idea of our algorithm is instead to group sets of connected components into coarser equivalence classes, defined as follows. The coarser definition allows reducing the total number of classes at the expense of introducing an approximation error in the balance of the solution.

▶ **Definition 6.** Let $\mathcal{S}$ be a set of disjoint connected components of the vertices of T, and $\varepsilon > 0$. A vector $\vec{g} = (g_0, \ldots, g_t)$, where $t = \lceil \log_{1+\varepsilon}(1/\varepsilon) \rceil + 1$, is called the *signature* of $\mathcal{S}$ if in $\mathcal{S}$ there are g_0 components of size in $[1, \varepsilon\lceil n/k \rceil)$ and g_i components of size in $[(1+\varepsilon)^{i-1} \cdot \varepsilon\lceil n/k \rceil, (1+\varepsilon)^{i} \cdot \varepsilon\lceil n/k \rceil)$, for each $i \in \{1, \ldots, t\}$.

The first stage of our algorithm uses a dynamic programming scheme to find a set of connected components of minimum cut size among those with signature $\vec{g}$, for any possible $\vec{g}$. Let $\mathbb{S}$ denote the set containing each of these optimal sets that cover all vertices of the tree, as computed by the first stage. In the second stage the algorithm attempts to distribute the connected components in each set $\mathcal{S} \in \mathbb{S}$ into k bins, where each bin has a capacity of $(1+\varepsilon)\lceil n/k \rceil$ vertices. This is done using a scheme originally proposed by Hochbaum and Shmoys [12, 22] for the `BIN PACKING` problem. The final output of our algorithm is the partition of the vertices of the given tree that corresponds to a packing of a set $\widetilde{\mathcal{S}} \in \mathbb{S}$ that uses at most k bins and has minimum cut size. Both stages of the algorithm have a runtime exponential in t. Hence the runtime is polynomial if ε is a constant.

We now describe the dynamic programming scheme to compute the set of connected components of minimum cut size among those whose signature is $\vec{g}$, for every possible $\vec{g}$. We fix a root $r \in V$ among the vertices of T, and an ordering of the children of every vertex in V. We define the *leftmost* and the *rightmost* among the children of a vertex, the siblings *left of* a vertex, and the *predecessor* of a vertex among its siblings according to this order in the natural way. The idea is to recursively construct a set of disjoint connected components for every vertex $v \neq r$ by using the optimal solutions of the subtrees rooted at the children of v and the subtrees rooted at the siblings left of v. More formally, let for a vertex $v \neq r$ the set $L_v \subset V$ contain all the vertices of the subtrees rooted at those siblings of v that are left of v and at v itself (Figure 4). We refer to a set $\mathcal{F}$ of disjoint connected components as a *lower frontier of* L_v if all components in $\mathcal{F}$ are contained in L_v and the vertices in V not covered by $\mathcal{F}$ form a connected component containing the root r. For every vertex v and every signature $\vec{g}$, the algorithm recursively finds a lower frontier $\mathcal{F}$ of L_v with signature $\vec{g}$. Finally, a set of connected components with signature $\vec{g}$ covering all vertices of the tree can be computed using the solutions of the rightmost child of the root. The algorithm selects a set having minimum cut size in each recursion step. Let $C_v(\vec{g}, m)$, for any vertex $v \neq r$ and any integer m, denote the optimal cut size over those lower frontiers of L_v with signature $\vec{g}$ that cover a total of m vertices with their connected components. If no such set exists let $C_v(\vec{g}, m) = \infty$. Additionally, we define $\mu := (1+\varepsilon)\lceil n/k \rceil$, and $\vec{e}(x)$ for any integer $x < \mu$ to be the signature of a set containing only one connected component of size x. We now show that the function $C_v(\vec{g}, m)$ can be computed using a dynamic program.

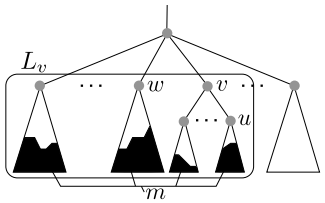

Figure 4 A part of a tree in which a vertex v, its rightmost child u, its predecessor w, the set of vertices L_v, and the m covered vertices by some lower frontier with signature $\vec{g}$ are indicated.

Let $\mathcal{F}^*$ denote an optimal lower frontier associated with $C_v(\vec{g}, m)$. First consider the case when v is a leaf and the leftmost among its siblings. Then $L_v = \{v\}$ and hence the set $\mathcal{F}^*$ either contains $\{v\}$ as a component or is empty. In the latter case the cut size is 0 and in the former it is 1 since the leaf has to be cut from the tree. Thus $C_v((0, \ldots, 0), 0) = 0$ and $C_v(\vec{e}(1), 1) = 1$ while all other function values equal infinity. Now consider the case when v is neither a leaf nor the leftmost among its siblings, and let w be the predecessor and u the rightmost child of v. The set L_v contains the vertices of the subtrees rooted at v's siblings that are left of v and at v itself. The lower frontier $\mathcal{F}^*$ can either be one in which the edge

from v to its parent is cut or not. In the latter case the m vertices that are covered by $\mathcal{F}^*$ do not contain v and hence are distributed among those in L_w and L_u since $L_v = L_w \cup L_u \cup \{v\}$. If x of the vertices in L_u are covered by $\mathcal{F}^*$ then $m - x$ must be covered by $\mathcal{F}^*$ in L_w. The vector $\vec{g}$ must be the sum of two signatures $\vec{g}_u$ and $\vec{g}_w$ such that the lower frontier of L_u (respectively L_w) has minimum cut size among those having signature $\vec{g}_u$ (respectively $\vec{g}_w$) and covering x (respectively $m - x$) vertices. If this were not the case the lower frontier in L_u (respectively L_w) could be exchanged with an according one having a smaller cut size—a contradiction to the optimality of $\mathcal{F}^*$. Hence in case v is a non-leftmost internal vertex and the edge to its parent is not cut, $C_v(\vec{g}, m)$ equals

$$\min\left\{C_w(\vec{g}_w, m - x) + C_u(\vec{g}_u, x) \mid 0 \leq x \leq m \wedge \vec{g}_w + \vec{g}_u = \vec{g}\right\}. \tag{1}$$

If the edge connecting v to its parent is cut in $\mathcal{F}^*$, then all n_v vertices in the subtree rooted at v are covered by $\mathcal{F}^*$. Hence the other $m - n_v$ vertices covered by $\mathcal{F}^*$ must be included in L_w. Let x be the size of the component $S \in \mathcal{F}^*$ that includes v. Analogous to the case before, the lower frontiers L_u and L_w with signatures $\vec{g}_u$ and $\vec{g}_w$ in $\mathcal{F}^* \setminus \{S\}$ must have minimum cut sizes. Hence the vector $\vec{g}$ must be the sum of $\vec{g}_u$, $\vec{g}_w$, and $\vec{e}(x)$. Therefore in case the edge to v's parent is cut, $C_v(\vec{g}, m)$ equals

$$1 + \min\left\{C_w(\vec{g}_w, m - n_v) + C_u(\vec{g}_u, n_v - x) \mid 1 \leq x < \mu \wedge \vec{g}_w + \vec{g}_u + \vec{e}(x) = \vec{g}\right\}. \tag{2}$$

Taking the minimum value of the formulas in (1) and (2) thus correctly computes the value for $C_v(\vec{g}, m)$ for the case in which v is neither a leaf nor the leftmost among its siblings. In the two remaining cases when v is either a leaf or a leftmost sibling, either the vertex u or w does not exist. Therefore for these cases the recursive definitions of C_v can easily be derived from formulas (1) and (2) by letting all function values $C_u(\vec{g}, x)$ and $C_w(\vec{g}, x)$ of a non-existent vertex u or w be 0 if $\vec{g} = (0, \ldots, 0)$ and $x = 0$, and ∞ otherwise.

The above recursive definitions for C_v give a framework for a dynamic program that computes the solution set $\mathbb{S}$ in polynomial time if ε is a constant, as the next theorem shows.

▶ **Theorem 7.** *For any tree and constant $\varepsilon > 0$ the set $\mathbb{S}$ is computable in polynomial time.*

The second stage of the algorithm attempts to pack each set of connected components $\mathcal{S} \in \mathbb{S}$ into k bins of capacity $(1+\varepsilon)\lceil n/k \rceil$. This means solving the well known `BIN PACKING` problem, which is NP-hard in general. However we are able to solve it in polynomial time for constant ε using a method developed by Hochbaum and Schmoys [12]. The description of the second phase has been deferred to the full version of the paper [7]. The main theorem resulting from the algorithms of the first and second phase is stated next.

▶ **Theorem 8.** *For any tree T, $k \in \{1, \ldots, n\}$, and $\varepsilon > 0$ there is an algorithm that computes a partition of T's vertices into k sets such that each set has size at most $(1+\varepsilon)\lceil n/k \rceil$ and its cut size is at most that of an optimal perfectly balanced partition of the tree. Furthermore the runtime is polynomial if ε is a constant and can be upper bounded by $\mathcal{O}(n^4(k/\varepsilon)^{1+3\lceil \frac{1}{\varepsilon}\log(\frac{1}{\varepsilon})\rceil})$.*

4 Extension to Unrestricted Weighted Graphs

In this section we present an algorithm that employs the PTAS given in Section 3 to find a near-balanced partition of a graph G with weighted edges. The (weighted) cut size computed has a capacity of at most $\alpha \in \mathcal{O}(\log n)$ times that of an optimal perfectly balanced partition of G. The algorithm relies on using our PTAS to compute near-balanced partitions of a set of decomposition trees that well approximate the cuts in G. This set can be found using

the results by Räcke [21]. The reason why this process yields an $\mathcal{O}(\log n)$ approximation of the cut size depends on the properties of the decomposition, which we now detail, after introducing a few definitions. A *cut* $W \subseteq V$ of the vertices of a graph $G = (V, E, u)$, with capacity function $u : E \to \mathbb{R}^+$, is to be computed. The quality of the cut is measured using the *cut size* $C(W)$, which is the sum of the capacities of the edges connecting vertices in W and $V \setminus W$. A *decomposition tree* of a graph $G = (V, E, u)$ is a tree $T = (V_T, E_T, u_T)$, with capacity function $u_T : E_T \to \mathbb{R}^+$, for which the leaves $L \subset V_T$ of T correspond to the vertices in G. More precisely there is a mapping $m_G : V_T \to V$ of all tree vertices to vertices in G such that m_G induces a bijection between L and V. Let $m_T : V \to L$ denote the inverse function of this bijection. Accordingly we also define a *leaf cut* $K \subseteq L$ of a tree. The *cut size* $C(K)$ of a leaf cut K is the minimum capacity of edges in E_T that have to be removed in order to disconnect K from $L \setminus K$. We map cuts W to leaf cuts K and vice versa using the notation $m_T(W)$ and $m_G(K)$ to denote the image of W and K according to m_T and m_G respectively. We make use of the following result which can be found in [18, 21].

▶ **Theorem 9.** *For any graph $G = (V, E, u)$ with n vertices, a family of decomposition trees $\{T_i\}_i$ of G and positive real numbers $\{\lambda_i\}_i$ with $\sum_i \lambda_i = 1$ can be found in polynomial time, such that for any cut W of G and corresponding leaf cuts $K_i = m_{T_i}(W)$, $C(K_i) \geq C(W)$ for each i (lower bound), and $\sum_i \lambda_i C(K_i) \leq \mathcal{O}(\log n) \cdot C(W)$ (upper bound).*

Since $\sum_i \lambda_i = 1$ the above theorem implies that for at least one tree T_i it holds that $C(K_i) \leq \mathcal{O}(\log n) \cdot C(W)$. As we will see, this allows for a fast approximation of balanced partitions in graphs using a modified version of the PTAS given in Section 3. We adapt the PTAS to compute near-balanced *leaf partitions* of each T_i. That is, it computes a partition $\mathcal{L} = \{L_1, \ldots, L_k\}$ of the l leaves L of a tree T into k sets of size at most $(1 + \varepsilon)\lceil l/k \rceil$ each. The *cut size* $C(\mathcal{L})$ in this case is the minimum capacity of edges that have to be removed in order to disconnect the sets in $\mathcal{L}$ from each other. It is easy to see that the PTAS given in Section 3 can be adapted to compute near-balanced leaf partitions: first signatures need to count leaves instead of vertices in Definition 6. Also edge counts are replaced with sum of edge capacities, in Equation 2. Moreover, we need to keep track of the number l_v of leaves at a subtree of a vertex v instead of the number n_v of vertices in Equations 1 and 2.

▶ **Corollary 10.** *For any weighted tree T, $\varepsilon > 0$, and $k \in \{1, \ldots, l\}$, there is an algorithm that computes a partition of the l leaves of T into k sets such that each set includes at most $(1 + \varepsilon)\lceil l/k \rceil$ leaves and its cut size is at most that of an optimal perfectly balanced leaf partition. The runtime is polynomial in k and the number of vertices of T if ε is constant.*

Using the above results we show that near-balanced partitions that deviate by only a logarithmic factor from the optimal cut size can be computed for graphs in polynomial time.

▶ **Theorem 11.** *Let $G = (V, E, u)$ be a graph with n vertices, $\varepsilon > 0$ be a constant, and $k \in \{0, \ldots, n\}$. There is an algorithm that computes in polynomial time a partition of V into k sets such that each set has size at most $(1 + \varepsilon)\lceil n/k \rceil$ and its cut size is at most $\mathcal{O}(\log n)$ times that of the optimal perfectly balanced solution.*

5 Conclusions

In this paper, the k-`BALANCED PARTITIONING` problem was studied on trees and some of the results were applied to weighted graphs. When a perfectly balanced solution is required, we showed that even when either the diameter or the degree of the tree is constant the

k-BALANCED PARTITIONING problem remains hard to approximate. In this sense, trees represent the simplest unit that capture the full complexity of the problem.

On the other hand, if one settles for near-balanced solutions, trees prove to be "easy" instances which admit a PTAS with approximation $\alpha = 1$, the best possible in the bicriteria sense. This crucial fact enables our PTAS for trees to be extended into an algorithm for general graphs with approximation factor $\alpha \in \mathcal{O}(\log n)$, improving on the best previous [1] bound of $\alpha \in \mathcal{O}(\log^{1.5}(n)/\varepsilon^2)$. Hence, remarkably, the same approximation guarantee can be attained on the cut size for the k-BALANCED PARTITIONING problem in case $k = 2$ (the BISECTION problem) and for unrestricted k, if we settle for near-balanced solutions in the latter case. This is in contrast to the strong inapproximability results for both general graphs and trees when the solutions are to be perfectly balanced.

Open Problems. For perfectly balanced partitions of trees it remains open to generalise our results to show a tighter dependency of the hardness on the degree (Figure 5). In addition, the possibility of an approximation algorithm for perfectly balanced partitions with a better ratio than $\alpha \in \mathcal{O}(\Delta \log_\Delta(n/k))$, as provided by the greedy scheme by MacGregor [17], remains open. In particular, Theorem 3 does not rule out an algorithm that approximates the cut size by the factor $\alpha = 25/24$ if $\Delta \leq 7$.

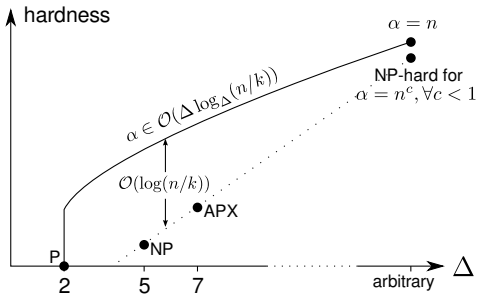

Figure 5 The hardness of trees versus their maximum degree Δ: the hardness results from Section 2 (large dots) indicate that the hardness grows with Δ (dotted line). A modification of MacGregor's [17] algorithm yields an approximation of $\mathcal{O}(\Delta \log_\Delta(n/k))$ (solid line). This means there is a gap of size $\mathcal{O}(\log(n/k))$ between the lower and upper complexity bounds in case Δ is constant.

For near-balanced partitions a question resulting from our work is whether faster algorithms can be found. We showed that we can achieve approximations on the cut size that do not depend on ε. Hence it suggests itself to try and find an algorithm that will compensate the cost of being able to compute a near-balanced solution for any $\varepsilon > 0$ not in the runtime but in the cut size. A recent result [6] however shows that there is no hope for a reasonable algorithm of this sort. More precisely it is shown that, unless P=NP, for trees there is no algorithm for which the runtime is polynomial in the input length and the inverse of ε, and has the following properties. The computed solution is near-balanced and the cut size may deviate from the optimum of a perfectly balanced solution by $\alpha = n^c/\varepsilon^d$, for any constants c and d where $c < 1$.

Another main challenge we see for general graphs is to resolve the discrepancy in complexity between the case $\varepsilon \geq 1$ and the case $\varepsilon < 1$, studied in this paper (recall Figure 2). For the case $\varepsilon \geq 1$ the algorithm by Krauthgamer *et al.* [15] achieves factor $\alpha \in \mathcal{O}(\sqrt{\log n \log k})$ and in the same paper it is shown that a dependency of α on k is unavoidable. Proving similar results for the case $\varepsilon < 1$ seems difficult to achieve, as the spreading metric techniques generally used for $\varepsilon \geq 1$ do not extend to $\varepsilon < 1$. Furthermore, the tree embedding results we used to achieve an $\mathcal{O}(\log n)$ approximation do not seem amenable to leading to algorithms with $o(\log n)$ approximation factor. Therefore it is likely that radically new techniques need to be developed to resolve the discrepancy.

References

1 K. Andreev and H. Räcke. Balanced graph partitioning. *Theor. Comp. Sys.*, 39(6):929–939, 2006.

2 S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings and graph partitioning. In *Proceedings of the 26th annual ACM symposium on Theory of computing*, pages 222–231, 2004.

3 J. Díaz, J. Petit, and M. J. Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34(3):313–356, 2002.

4 J. Díaz, M. J. Serna, and J. Torán. Parallel approximation schemes for problems on planar graphs. *Acta Informatica*, 33(4):387–408, 1996.

5 G. Even, J. Naor, S. Rao, and B. Schieber. Fast approximate graph partitioning algorithms. In *Proceedings of the 8th annual ACM-SIAM symposium on Discrete algorithms*, pages 639–648, 1997.

6 A. E. Feldmann. Fast balanced partitioning of grid graphs is hard. *ArXiv e-prints*, (arXiv:1111.6745v1), November 2011.

7 A. E. Feldmann and L. Foschini. Balanced partitions of trees and applications. Technical Report 733, Institute of Theoretical Computer Science, ETH Zürich, July 2011.

8 A. E. Feldmann and P. Widmayer. An $O(n^4)$ time algorithm to compute the bisection width of solid grid graphs. In *Proceedings of the 19th Annual European Symposium on Algorithms*, pages 143–154, 2011.

9 T. Feo, O. Goldschmidt, and M. Khellaf. One-half approximation algorithms for the k-partition problem. *Operations Research*, 40:170–173, 1992.

10 T.A. Feo and M. Khellaf. A class of bounded approximation algorithms for graph partitioning. *Networks*, 20(2):181–195, 1990.

11 M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the Theory of NP-completeness*. W.H. Freeman and Co., 1979.

12 D. S. Hochbaum and D. B. Shmoys. A polynomial approximation scheme for machine scheduling on uniform processors: using the dual approximation approach. *SIAM Journal on Computing*, 17(3):539–551, 1988.

13 D. G. Kirkpatrick and P. Hell. On the completeness of a generalized matching problem. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 240–245, 1978.

14 P. Klein, S.A. Plotkin, and S. Rao. Excluded minors, network decomposition, and multicommodity flow. In *Proceedings of the 25th annual ACM symposium on Theory of computing*, pages 682–690, 1993.

15 R. Krauthgamer, J. Naor, and R. Schwartz. Partitioning graphs into balanced components. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 942–949, 2009.

16 T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.

17 R. M. MacGregor. *On partitioning a graph: a theoretical and empirical study*. PhD thesis, University of California, Berkeley, 1978.

18 A. Madry. Fast approximation algorithms for cut-based problems in undirected graphs. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 245 –254, 2010.

19 J. K. Park and C. A. Phillips. Finding minimum-quotient cuts in planar graphs. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 766–775, 1993.

20 E. Petrank. The hardness of approximation: Gap location. *Computational Complexity*, 4(2):133–157, 1994.

21 H. Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, 2008.

22 V. V. Vazirani. *Approximation Algorithms*. Springer, 2003.