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 problem of finding the minimum number of edges that, when cut, form a partition of the vertices into k sets of equal size. This is called the k-BALANCED PARTITIONING problem. The problem is known to be inapproximable within any finite factor on general graphs, while little is known about restricted graph classes.

We show that the k-BALANCED PARTITIONING problem remains APX-hard even when restricted to unweighted tree instances with constant maximum degree. If instead the diameter of the tree is constant we prove that the problem is NP-hard to approximate within n^c, for any constant $c < 1$.

If vertex sets are allowed to deviate from being equal-sized by a factor of at most $1+\varepsilon$, we show that solutions can be computed on weighted trees with cut cost no worse than the minimum attainable when requiring equal-sized sets. This result is then extended to general graphs via decompositions into trees and improves the previously best approximation ratio from $O(\log^{1.5}(n)/\varepsilon^2)$ [Andreev and Räcke TCS 2006] to $O(\log n)$. This also settles the open problem of whether an algorithm exists for which the number of edges cut is independent of ε.

[*]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. An extended abstract [14] of this article appeared at the *29th International Symposium on Theoretical Aspects of Computer Science (STACS)*.

1 Introduction

In this article we study the k-BALANCED PARTITIONING problem. The problem asks for a partition of the n vertices of a graph into k sets of size at most $\lceil n/k \rceil$ each. At the same time the total number of edges connecting vertices in different sets, called the *cut cost*, needs to be minimised. The problem has numerous applications of which one of the most prominent is in the field of parallel-computing [2]. 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 VLSI circuit design [5], image processing [31, 37], computer vision [23], route planning [7], and divide-and-conquer algorithms [25, 32]. However, despite the broad applicability, k-BALANCED PARTITIONING is a notoriously hard problem. The special case of $k = 2$, commonly known as the BISECTION problem, is already NP-hard [18]. For this reason, approximation algorithms that find a balanced partition with a cut cost larger than optimal have been developed. We follow the convention of denoting the approximation ratio on the cut cost by α.

Unfortunately, when k is not constant even finding an approximation of the minimum balanced cut still remains infeasible. In fact, no finite approximation for the cut cost can be computed in polynomial time, unless P=NP [1]. In order to circumvent the hardness results, researchers have investigated algorithms that compute solutions with relaxed balance constraints. 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 $\varepsilon > 0$. In most cases the algorithms proposed are *bicriteria approximations*, which approximate both the balance and the cut cost of the optimal solution. The approximation ratio achieved by these algorithms is measured by comparing the cut cost of the computed solution in which the balance constraint is relaxed, with the optimal cut cost of a *perfectly balanced* partition. That is, one in which each set has size at most $\lceil n/k \rceil$.

The k-BALANCED PARTITIONING problem has received some attention for the case $\varepsilon = 1$. This means approximating the perfect balance within a factor of two. For this case, the best result is by Krauthgamer *et al.* [22] who give an algorithm with approximation factor $\alpha \in \mathcal{O}(\sqrt{\log n \log k})$ on the cut cost. However, it is not hard to imagine how the slack on the balance can be unattractive for 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. This is because the completion time is

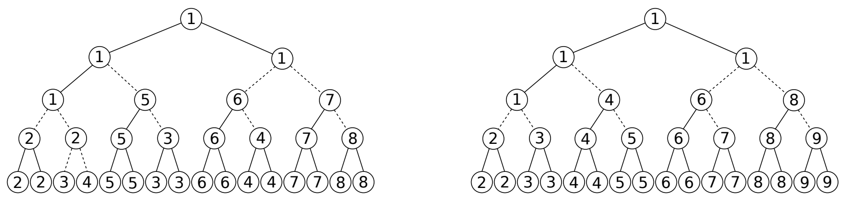

Figure 1: Two optimally partitioned binary trees. For the tree on the left $k = 8$ (with a cut cost of 10) whereas $k = 9$ (with a cut cost of 8) for the tree on the right. The numbers in the vertices indicate the set they belong to and the cut edges are dashed.

solely determined by the machines with the highest workload. Therefore the case when ε can be chosen arbitrarily between 0 and 1, is of greater practical interest. We call the resulting partitions *near-balanced*. 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 $\varepsilon = 1$.

As argued in [1], it is not surprising that the achieved ratio α is worse for near-balanced solutions than it is in the case $\varepsilon = 1$. This is because typically the algorithms for $\varepsilon = 1$ work in two phases. First the graph is partitioned into components of size at most $2\lceil n/k \rceil$ while minimising the cut cost. Then the components are packed into k bins. The fact that $\varepsilon = 1$ implies that any such returned components can be packed into k bins using a greedy algorithm. However this approach cannot be used to find near-balanced solutions. In fact, as ε approaches 0 and the constraint on the balance becomes more stringent, the cutting phase must be adapted to return components that can later be packed into bins of the required size. This fact significantly complicates the cutting phase.

1.1 Our Contribution

As argued above, the restriction to near-balanced partitions poses a major challenge in understanding the k-`BALANCED PARTITIONING` problem. For this reason, we consider the simplest non-trivial instance class of the problem, namely connected trees. Even in this simple case the structure of the solution is not easily understood. 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 cost. Our results confirm

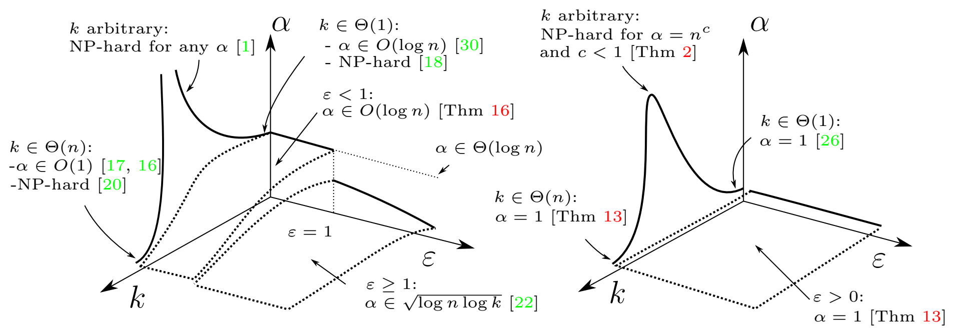

Figure 2: Illustrations of best approximation factor α known against k and ε, for general graphs (left) and trees (right). The plane (α, k) represents the case of perfectly balanced solutions ($\varepsilon = 0$) and shows that the restriction to trees does not significantly change the asymptotic behaviour. 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 $\varepsilon = 1$ visible in the plot on the left.

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 the cut cost within $\alpha = n^c$ for any constant $c < 1$. 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 equal to 4. Instances of diameter at most 3 on the other hand, are polynomially solvable.

By a substantially different argument, we show that a similar dichotomy arises when parametrising the complexity with 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 [26] that can be modified 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 present approximation algorithms that compute near-balanced partitions for edge-weighted graphs. In this case the cut cost is measured by the total weight of the respective edge set of the solution. We show that when near-balance is allowed, trees exhibit a substantially better behavior compared to general graphs. We present an algorithm for edge-weighted trees that computes a near-balanced partition for any constant $\varepsilon > 0$ in polynomial time. It achieves a cut cost 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, such a PTAS can be shown to yield an optimal *perfectly balanced* solution for trees if $k \in \Theta(n)$. On general graphs the problem is NP-hard for these values of k [20].

In the last section of our article we capitalise on the PTAS we presented for trees to tackle the k-`BALANCED PARTITIONING` problem on general edge-weighted graphs. By decomposing 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 decomposition, 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 the balance factor 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 answer an open question posed in the same paper whether an algorithm with no dependence on ε in the ratio α exists. Figure 2 summarises the related work and our contribution.

1.2 Related Work

This article extends the results in [1], where it is shown that approximating the cut cost of the k-`BALANCED PARTITIONING` problem on general graphs 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. When more unbalance is allowed, for $\varepsilon = 1$ Even *et al.* [10] present an algorithm with $\alpha \in \mathcal{O}(\log n)$ that uses spreading metrics techniques. For the same value of ε, Simon and Teng [33] gave a method that can achieve a factor of $\alpha \in \mathcal{O}(\log k\sqrt{\log n})$ by recursively applying edge separators. Later Krauthgamer *et al.* [22] improved these results to $\alpha \in \mathcal{O}(\sqrt{\log n \log k})$ using a semidefinite relaxation which combines l_2^2 metrics with spreading metrics. For graphs with excluded minors[1] (such as planar graphs) solutions with $\alpha \in \mathcal{O}(1)$ and $\varepsilon = 1$ can be computed in $\widetilde{\mathcal{O}}(n^3)$ deterministic, or $\widetilde{\mathcal{O}}(n^2)$ expected, time by applying a spreading metrics relaxation and the results in [21]. A faster algorithm can be obtained [11] for the special case of grid graphs without holes. This is achieved by combining some structural insights [13] on the cut shapes with the above mentioned techniques of Simon and Teng [33]. This yields an algorithm running in $\widetilde{\mathcal{O}}(n^{1.5})$ time. While the algorithm also computes partitions with $\varepsilon = 1$, it trades the faster runtime with a worse ratio of $\alpha \in \mathcal{O}(\log k)$. For the problem under consideration, to the best of our knowledge these are the only results for restricted graph classes. However for a related problem on scheduling of pipelined operator trees, Bodlaender *et al.* [6] gave a PTAS. In this problem the objective is also to find a partition of the vertices into k sets. However instead of fixing an upper bound on the set sizes, the optimisation function to be minimised depends on the cut cost and the maximum set size.

The special case when $k = 2$, commonly known as the `BISECTION` problem, is well studied. The `BISECTION` problem is NP-hard in the general case [18] but approximation algorithms are known. Räcke [30] gives an algorithm with approximation ratio $\alpha \in \mathcal{O}(\log n)$ for perfectly balanced partitions. For near-balanced partitions, Leighton and Rao [24] show how to compute a solution using min-ratio cuts. In this solution the cut cost is approximated within $\alpha \in \mathcal{O}(\gamma/\varepsilon)$, where γ is the approximation factor of computing a min-ratio cut. In [24] it was shown that $\gamma \in \mathcal{O}(\log n)$, and this result was improved [3] 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 [28]. If a perfectly balanced solution to `BISECTION` is required for planar graphs, Dìaz *et al.* [9] 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 [28], 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 [8]. For instance an $\mathcal{O}(n^4)$ time algorithm for grid graphs without holes has been found [15], while for trees an $\mathcal{O}(n^2)$ time algorithm [26] exists.

In addition to the case $k = 2$, some results are known for other extreme values of k. For trees the algorithm from [26] 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 [20] for any $k \leq n/3$ on general graphs. Feo and

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

Khellaf [17] give an $\alpha = n/k$ approximation algorithm for the cut cost which was improved to $\alpha = 2$ [16] in case k equals $n/3$ or $n/4$.

We complete the review of related work by discussing the literature on hierarchical graph decompositions, which we leverage to extend our PTAS for trees to general graphs. Informally a hierarchical 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. Graph decompositions have been studied in the context of oblivious routing schemes (see [27] for a survey). Räcke [30] introduced an optimal decomposition with factor $\mathcal{O}(\log n)$, which we employ in the present work. In a recent work, Madry [27] shows that it is possible to generalise Räcke's insights so that any *cut based* problem (see also [36]) is solvable on graphs by computing solutions on trees resulting from a decomposition of the input graph. This result directly translates to our scenario and hence we use his notation in the present work.

2 The Hardness of Computing Perfectly Balanced Partitions in Trees

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

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

The `3-PARTITION` problem is strongly NP-hard [18] which means that it remains so even if all integers are polynomially bounded in the size of the input. We call an instance to the `3-PARTITION` problem a *YES instance* if it is solvable and a *NO instance* otherwise.

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 decide an instance of `3-PARTITION`. The idea for the reduction is similar to the one used by Andreev and Räcke [1] for general graphs. The result remains valid even if the diameter of the tree is bounded by a constant.

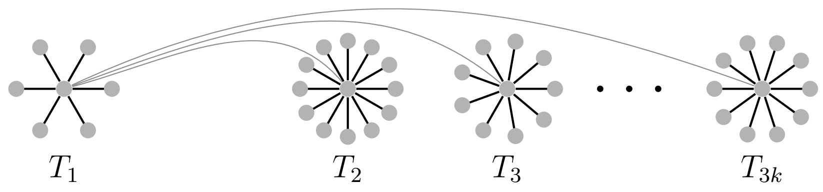

Figure 3: Construction for the reduction of Theorem 2. Thin grey edges link star centres, darkened edges connect centres to leaves.

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

Proof. The construction used in the proof is shown in Figure 3. Let $m = 3kn^c$ for some constant $c < 1$. For each a_i of a given instance I of the `3-PARTITION` problem, define a corresponding gadget T_i as a star on $a_i m$ vertices. Construct a tree T where the centres of all T_i for $i \geq 2$ are connected to the centre of T_1, as shown in Figure 3. The number of vertices of the resulting tree is $n = \sum_{i=1}^{3k} a_i m = 3k^2 sn^c$. Solving this equation for n gives $n = (3k^2 s)^{\frac{1}{1-c}}$. Since c is constant and s can be assumed to be polynomially bounded in k, the tree T can be constructed in polynomial time. The following lemma, which we prove below, gives the properties needed of such a tree T to prove Theorem 2.

Lemma 3. *In a tree T as constructed above for a `3-PARTITION` instance I, an optimal perfectly balanced partition has cut cost at most $3k - 1$ if and only if I is a YES instance. Otherwise it requires at least $m = 3kn^c$ edges.*

Given that this lemma is true, this would allow an approximation algorithm with approximation factor at most n^c to decide between a YES and a NO instance of the `3-PARTITION` problem. Since the latter is NP-hard this proves the theorem. □

Proof of Lemma 3. It is easy to see that if I is a YES instance then cutting at most the $3k - 1$ edges that connect the T_is suffices. Suppose now that I is a NO instance and that the cut set C^* of minimum size that partitions T into $\{V_1, \dots, V_k\}$, where $|V_i| = ms$, has size strictly less than m. The set C^* can be expressed as $A \cup B$, where A contains only edges that link T_i centres, and

B contains only edges separating a T_i's centre from one of its leaves. By the assumption on the cut cost it follows that $|A| < m$ and $|B| < m$. Also $|B| > 0$ as cutting only edges from A would separate complete T_is, thus implying a solution to I, and contradicting the fact that I is a NO instance.

Let V_l be a set of the partition induced by C^* that contains at least one isolated leaf v, which always exists since $|B| > 0$. Assume that V_l contains an incomplete T_i, that is, V_l contains the centre of such a T_i but not one of its leaves w. Then w must be contained in some other V_j, where $j \neq l$. Thus swapping v and w allows putting both w and the incomplete T_i in V_l, rendering the cut that separate them superfluous. This contradicts the assumption that C^* is a cut set of minimum size. Hence V_l can contain only a (possibly empty) set of complete T_is in addition to one or more isolated leaves.

The proof is completed by noticing that the number of vertices of a set of complete T_is is a multiple of m. Therefore at least m additional isolated leaves are required for $|V_l|$ (recall $|V_l| = ms$) to be a multiple of m, contradicting the assumption that $|B| < m$ and establishing the result. □

The reduction in the proof of the above theorem relies on the fact that the degree of the tree is unbounded. Therefore the natural next step is to investigate the complexity of bounded degree trees. As we will show next, the problem remains surprisingly hard for constant degree trees. 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. We prove this result by a reduction from the `GAP-3-PARTITION` problem. This is the `3-PARTITION` problem in which, for a given ρ, either all or at most a ρ fraction of the a_is can be partitioned into the desired triples. A formal definition follows.

Definition 4 (`GAP-3-PARTITION`)**.** Let $3k$ integers a_1 to a_{3k}, a threshold $s \in \mathbb{N}$, and $\rho > 1$ be given, such that $s/4 < a_i < s/2$, $\sum_{i=1}^{3k} a_i = ks$, and the integers can either all be partitioned into k triples that sum up to exactly s or at most k/ρ of them can. Decide whether k or at most k/ρ such triples can be found.

There is a constant $\rho > 1$ for which the `GAP-3-PARTITION` problem is NP-hard. This is true even if all integers are polynomially bounded in k. These results follow from the original NP-hardness proof of `3-PARTITION` by

Garey and Johnson [18] and the results of Petrank [29].[2] The latter result introduces a constant sized gap for the 3D-MATCHING problem. By considering the reductions given by Garey and Johnson from 3D-MATCHING to 3-PARTITION it can readily be seen that they are gap-preserving (cf. [4]). These reductions also establish the strong NP-hardness of 3-PARTITION.

The technique we use to prove the hardness for constant degree trees is substantially different, and more involved, than the method used to prove Theorem 2. 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 GAP-3-PARTITION instance.

Theorem 5. *Unless P=NP, there exists a constant $\rho > 1$ such that the k-BALANCED PARTITIONING problem on unweighted trees with maximum degree at most 7 has no polynomial time approximation algorithm with ratios $\varepsilon = 0$ and $\alpha = 1 + (1 - \rho^{-1})/24$.*

Proof. Consider an instance I of GAP-3-PARTITION with polynomially bounded integers that are divisible by 12. Obviously all hardness properties are preserved by this restriction since GAP-3-PARTITION is strongly NP-hard and we may multiply each integer and the threshold parameter of an arbitrary instance by 12. 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 an a_i-path) connected to the root of a tree on s vertices (referred to as an 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. Additionally the roots of the s-trees are connected in a path, as shown in Figure 4. We define B to be the set of edges connecting different T_is and A the set of edges connecting an a_i-path with the root of the corresponding s-tree in each T_i. Below, we will prove the following lemma, which states the structure needed of the constructed tree T in order to prove Theorem 5.

Lemma 6. *If all k integers in a given GAP-3-PARTITION instance I can be partitioned into triples that sum up to exactly s, then the constructed tree T can be split into $4k$ parts of a perfectly balanced partition with cut*

[2]We thank Nikhil Bansal for pointing out the connection between the reductions in [18] and the results by Petrank [29].

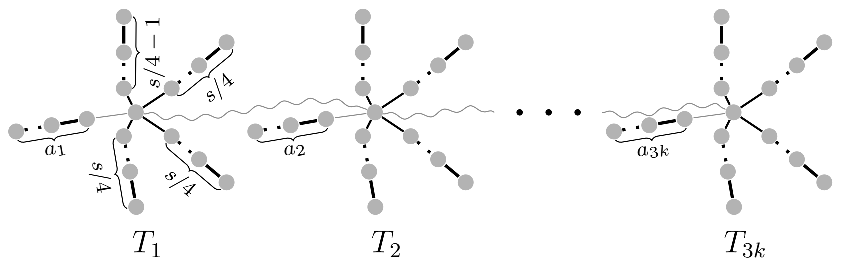

Figure 4: Construction for Theorem 5. Each gadget T_i is composed by an a_i-path connected to the root of an s-tree through an edge from A (straight grey). Each s-tree branches into four paths of (almost) the same length. Two adjacent gadgets in a path are connected through the roots of their s-trees with an edge from B (wiggled grey).

cost $6k-1$. *If however at most* k/ρ *such triples can be found,* T *requires at least* $(1-\rho^{-1})k/4$ *additional cut edges if* $s > 20$.

It immediately follows that an algorithm computing an approximation within a factor smaller than $\frac{6k-1+(1-\rho^{-1})k/4}{6k-1}$ of the optimum cut cost, can decide the `GAP-3-PARTITION` problem. Hence this proves the theorem. □

Proof of Lemma 6. 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 perfectly balanced partition into $4k$ parts.

It remains to be shown that $(1-\rho^{-1})k/4$ additional edges are required when the integers in I can be partitioned into at most k/ρ triples of size exactly s. Let in this case C^* be an optimal set of edges that cuts T into $4k$ parts of a perfectly balanced partition. 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 cannot 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 technical lemma which we will prove later.

Lemma 7. *If* $s > 20$ *then* $|C| \geq 2|(A \cup B) \setminus C^*|$.

Consider the following algorithm $\mathcal{A}$ which repositions cut edges from a perfectly balanced partition into $4k$ parts. As long as there is an uncut edge

$e \in A \cup B$, $\mathcal{A}$ removes a cut edge left 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 still left in C denoted by C'. Then $|C'|$ is the minimum number of additional edges cut in the case at most k/ρ triples that sum up to exactly s can be formed from the integers. When repositioning a cut edge from C to $A \cup B$, or when removing the edges in 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 the component 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 7 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 $\mathcal{A}$ is 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 and $3k$ integers of size s. The integers in I can be partitioned into at most k/ρ triples of size exactly s. Hence 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 perfectly balanced partition of T into a solution to `GAP-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 given that $|C'|$ is the number of additional cuts required. □

Proof of Lemma 7. It is sufficient to prove that the lemma holds for all disjoint subtrees T' of T that contain at least one edge from $(A \cup B) \setminus C^*$. That is, if E' is the edge set of T' we set out to prove $|C \cap E'| \geq 2|((A \cup B) \setminus C^*) \cap E'|$, for all pairwise disjoint trees T' such that their union contains $(A \cup B) \setminus C^*$. Consider all connected components cut out by a balanced partition of T that contain at least a root of an s-tree. Create a subtree T' from each such component W by re-attaching all the cut s-tree branches incident to it and all a_i-paths incident to the edges in $A \setminus C^*$ contained in W (see Figure 5). Note that T' contains the same edges from A and B as W. In particular no edges from $(A \cup B) \cap C^*$ are in T'. We distinguish two cases.

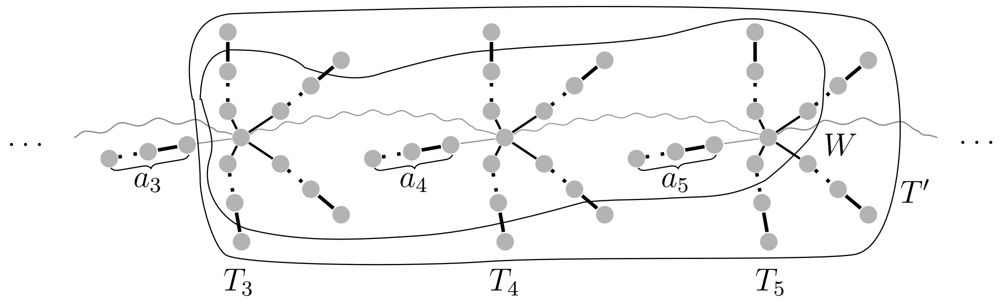

Figure 5: Part of the construction of Lemma 7 with components W and T' highlighted.

First, consider a T' that contains no edges from B. If T' also does not contain an edge from A then the lemma is trivially true. Otherwise, the tree T' is simply a gadget T_i and has a total size of $s + a_i$. Since $a_i > s/4$ and each branch of the s-tree in T' has at most $s/4$ vertices, there must be at least two edges from C in T' in order to cut away a_i vertices. This proves the lemma in the case T' contains no edges from B.

Consider now a subtree T' that contains at least one edge from B, and the connected component W from which T' originated. Let A' and B' denote the edges in W from A and B, respectively. Each branch of an s-tree and each a_i-path in T' has at least $s/4 - 1$ vertices. As $s > 20$, if 5 or more branches of s-trees or a_i-paths were fully included in W (before the extension to T') this connected component would contain more than s vertices. This is a contradiction since every connected component has size at most s. Therefore there are at most 4 such included branches. The branches that are not fully included in W but are in T', each contains an edge from C. Since T' contains at least $4(|B'|+1)$ s-tree branches and $|A'|$ a_i-paths, we can conclude that the number of edges from C in T' is at least $|A'|+4|B'|$. Notice that $|B'| \geq |A'|-1$ since otherwise W would be disconnected. Using the fact that $|B'| \geq 1$ we obtain

$$|A'| + 4|B'| \geq 2|A'| - 1 + 3|B'| \geq 2|A' \cup B'|.$$

This proves our claim in the case T' contains at least one edge from B, and concludes the proof of the lemma. $\square$

Using similar ideas as in the proof of Theorem 5, if we restrict the degree to be at most 5 we can still show that the problem remains NP-hard. For this we again reduce from `3-PARTITION` and use a slightly different construction

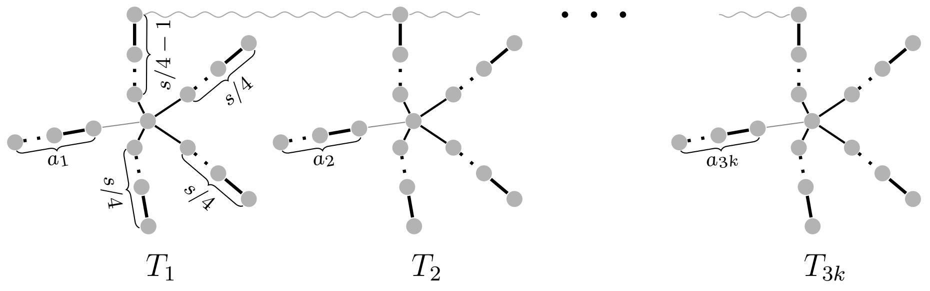

Figure 6: Construction for Theorem 8. Each gadget T_i is composed by a path on a_i nodes (a_i-path) connected to the root of a tree on s vertices (s-tree). Each s-tree branches from the root in three paths of length $s/4$ each and a shorter one of length $s/4-1$. The leaves of the shorter branch of each s-tree are connected by edges of the set A (wiggled thin grey) in a path. In each T_i, the a_i-path and the s-tree are connected by an edge belonging to the set B (straight thin grey).

than the one shown in Figure 4. Instead of connecting the s-trees through their roots, the B edges connect the leaves of the shortest branches of the s-trees (Figure 6). It is then possible to show that exactly the $6k-1$ edges in A and B are cut if all k integers in the instance I can be partitioned into triples of size exactly s, while otherwise at least $6k$ edges are cut. Since the `3-PARTITION` problem is NP-hard [18] this suffices to establish the following result.

Theorem 8. *Unless P=NP, the k-`BALANCED PARTITIONING` problem on unweighted trees with maximum degree at most 5 has no polynomial time algorithm.*

Proof. Consider an instance I of `3-PARTITION` where s is a multiple of 4. Clearly the `3-PARTITION` problem remains strongly NP-hard even if restricted to such instances since we may multiply all integers by 4. For each a_i in I, construct a gadget T_i composed by a path on a_i vertices (hereinafter, a_i-path) connected to the root of a tree on s vertices (hereinafter, 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 leaves of the shortest branch of each s-tree in a path, as shown in Figure 6. We call A the set of edges connecting different T_is and B the set of edges connecting an a_i-path with the corresponding s-tree in each T_i.

At a high level, we want to argue that an algorithm that can find a perfectly balanced partition of T into $4k$ sets could be used to solve the

`3-PARTITION` problem on I. More precisely, we set out to prove that T can be split into $4k$ sets using exactly $6k-1$ cut edges if and only if I is a YES instance. To this end, we call a partition *light* if it splits T into connected components each of size at most s. We leverage the following lemma which provides the structure of the constructed tree T needed to prove Theorem 8.

Lemma 9. *The only cut set of minimum size that breaks a tree T constructed for a `3-PARTITION` instance I into a light partition contains all and only the $6k-1$ edges in $A \cup B$.*

Before proving Lemma 9 we see how Theorem 8 follows from it. Clearly, breaking T into connected components of size less than or equal to s is a necessary condition to obtain a perfectly balanced partition into $4k$ sets. It follows that no such partition can cut less than $6k-1$ edges. If I is a YES instance then cutting all the $6k-1$ edges in $A \cup B$ suffices to find a perfectly balanced partition of T into $4k$ sets. This is because the vertices of the a_i-paths can then be partitioned into sets of size s with no additional cuts. By Lemma 9 it also follows that if cutting $6k-1$ edges suffices to find a perfectly balanced partition of T into $4k$ sets, the cut set must coincide with $A \cup B$. Therefore no additional cut edges are required only if the integers corresponding to the a_i-paths can be packed into triples of size s each. This yields a solution to the `3-PARTITION` instance I, given that each s-tree separated by the $A \cup B$ cut set completely fills a set of size s. Hence if I is a NO instance then the cut cost is at least $6k$, which completes the proof. □

Proof of Lemma 9. First observe that cutting $A \cup B$ suffices to create a light partition. We show that *any* light partition that does not use some edge in $A \cup B$ can be transformed into a light partition that has the same number of cut edges and cuts all edges in $A \cup B$ plus at least one more. Therefore the only light partition of minimum cut cost is the one where all edges in $A \cup B$ are cut.

We begin by showing how to convert a light partition that does not use an edge $b \in B$, connecting an a_i-path to an s-tree, into one that uses all edges in B without increasing the cut cost. Suppose T_l is a gadget where an edge $b \in B$ is not cut. Since the size of T_l is $s + a_l$ at least a_l vertices have to be separated from T_l to form a light partition. Each branch of the s-tree is strictly smaller than a_l and hence, without using b, there must be at least two edges cut in two different branches of T_l. At least one of these edges c cannot be on the shortest branch of the s-tree, which is connected to a neighbouring T_i. There are two cases. If c cuts some edge of the a_l-path (other than b), then c can be replaced with b without invalidating the light partition property. Else, if no cut edge is on the a_l-path, there is a cut edge c on one of the longest branches of the s-tree of T_l which can be replaced with b. This again results in a light partition as the a_l-path has size strictly larger than any branch of the s-tree of T_l.

Repeatedly applying the transformation described above for each gadget T_i, we can obtain a light partition of T where all edges in B are cut and the total size of the cut set has not increased. Also at least one additional edge in such a gadget T_i is cut. We are then left to prove that if any edge $a \in A$ is not cut then the cut set is strictly suboptimal.

Suppose not all edges from A are cut. Then there exist chains of length $p > 1$ of T_is connected by edges in A that are not cut. By the above transformation, we can assume that all T_is in such a chain have their corresponding a_i-path cut out. Furthermore, each T_i in the chain must have at least one more edge cut. If that was not the case the size of the connected component of the light partition containing a T_i that violates the property would be at least $s+1$. This is because the edge $a \in A$ from T_i to its neighbouring gadget in the chain is not cut. Consider T_l, the first T_i of such a chain. The cut edge $c \notin B$ in T_l can be replaced by the edge $a \in A$ to the neighbouring gadget without invalidating the light partition property. This follows from the fact that cutting a will completely separate T_l from the chain and the total size of T_l with its a_i-path removed is exactly s. The same process can be repeated for all the $p-2$ remaining edges from A on the chain of gadgets. This implies that for each chain of T_is all but at least one cut edge can be replaced with edges in A.

Hence after replacing each cut edge not in $A \cup B$ with one from this set we obtain a light partition with the same number of cut edges. However in this light partition at least one additional edge is cut in addition to those in $A \cup B$ and hence the lemma follows. $\square$

3 Near-Balanced Partitions for Trees

The previous section shows that approximating the cut cost of k-`BALANCED PARTITIONING` is hard even on unweighted 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 guarantee. 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 with edge weights. It computes near-balanced partitions but returns a cut cost no larger than the optimum of a perfectly balanced solution.

Assume we are given an edge-weighted tree $T = (V, E, \omega)$ with weight function $\omega : E \to \mathbb{R}^+$. Conceptually one could find a *perfectly balanced* partition of T with minimum cut cost 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 $x \in \{1, \dots, \lceil n/k \rceil\}$. In a first step the set of connected components that achieves the cut of minimum cost 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 overall cut cost is returned. Clearly such an algorithm finds the optimal solution to the k-`BALANCED PARTITIONING` problem, but the runtime is exponential in n as, in particular, the total number of equivalence classes is exponential. To get around this problem we instead group sets of connected components into coarser equivalence classes. These are determined by subdividing the possible component sizes into intervals. A coarse class then consists of cuts for which each interval in total contains the same number of component sizes. By making the lengths of the intervals appropriately depend on ε, this reduces the equivalence classes to a polynomial number if ε is constant. However this also introduces an approximation error in the balance of the solution.

Definition 10. Let $\mathcal{S}$ be a set of disjoint connected components of the vertices of T, and $\varepsilon > 0$. A vector $\vec{g} = (g_0, \dots, g_t)$, 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 $i \in \{1, \dots, t\}$.

The first stage of our algorithm uses a dynamic programming scheme to find a set of connected components of minimum cut cost among those with signature $\vec{g}$, for any possible $\vec{g}$. This dynamic programming scheme is a

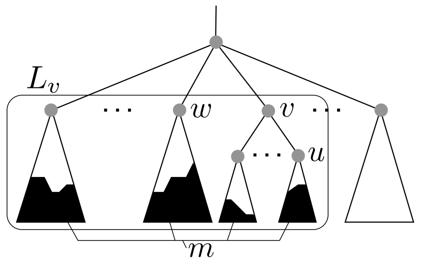

Figure 7: A part of a tree in which a vertex v, its rightmost child u, its predecessor w, the set of vertices L_v, and the m covered vertices by some lower frontier with signature $\vec{g}$ are indicated.

generalisation of those found for the `BISECTION` problem (see e.g. [26, 34, 36]). 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 [19, 35] 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 cost. Both stages of the algorithm have a runtime exponential in t. Hence the runtime is polynomial if ε is a constant.

3.1 The Cutting Phase

We now describe the dynamic programming scheme to compute the set of connected components of minimum cut cost among those whose signature is $\vec{g}$, for every possible $\vec{g}$. The method we use generalises the algorithms for the `BISECTION` problem on trees [26, 34, 36]. 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. 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 7). 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 cost in each recursion step. Let $C_v(\vec{g}, m)$, for any vertex $v \neq r$ and any integer m, denote the optimal cut cost 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.

Let $\mathcal{F}^*$ denote an optimal lower frontier associated with $C_v(\vec{g}, m)$. We will consider the four cases resulting from whether or not the vertex v is a leaf, and whether or not it is the leftmost among its siblings. First consider the case when both properties are met. That is, 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 cost is 0. In the former it is the weight $\omega(e)$ of the edge e incident to the leaf that is cut from the tree. Thus $C_v((0, \ldots, 0), 0) = 0$ and $C_v(\vec{e}(1), 1) = \omega(e)$ while all other function values equal infinity. Now consider the case when v is neither a leaf nor the leftmost among its siblings. Let w be the predecessor among v's siblings 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 cost 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 one having a smaller cut cost—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,

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

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 costs. 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 e to v's parent is cut,

$$\begin{aligned} C_v(\vec{g}, m) = \omega(e) + \min\big\{&C_w(\vec{g}_w, m-n_v) + C_u(\vec{g}_u, n_v - x) \mid \\ &1 \leq x < \mu \wedge \vec{g}_w + \vec{g}_u + \vec{e}(x) = \vec{g}\big\}. \end{aligned} \quad (2)$$

Taking the minimum value of the formulas given 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. For these cases the recursive definitions of $C_v(\cdot,\cdot)$ can easily be derived from Equations (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 $C_v(\cdot,\cdot)$ give a framework for a dynamic programming scheme that computes the wanted solution set $\mathbb{S}$ in polynomial-time if ε is a constant, as the next theorem shows.

Theorem 11. *For any tree T and any constant $\varepsilon > 0$ there is an algorithm that computes $\mathbb{S}$ in polynomial time.*

Proof. If the tree contains only one vertex the theorem obviously holds. Otherwise the optimum solution from $\mathbb{S}$ that has signature $\vec{g}$ must contain a connected component that includes the root r and has some size x. Clearly x is at least 1 and at most μ. Hence, if u denotes the rightmost child of the root r, the cut cost $C(\vec{g})$ of the optimal solution for $\vec{g}$ can be computed in linear time using the formula

$$C(\vec{g}) = \min\{C_u(\vec{g} - \vec{e}(x), n - x) \mid 1 \leq x < \mu\}. \quad (3)$$

An optimal set of connected components with signature $\vec{g}$ can be computed using the dynamic program given by the above equation together with the recursive definition of C_v by keeping track of the set of connected components used in each recursion step.

To analyse the runtime let us first bound the number of signatures $\vec{g}$ that have to be considered for a vertex v in the dynamic program. Let $N_v = |L_v|$ denote the number of vertices in the subtrees rooted at the siblings left of v and at v itself. There are N_v vertices that can be distributed into connected components of different sizes to form a lower frontier $\mathcal{S}$ of L_v. Each entry g_i of $\vec{g}$ counts components of size at least the lowest value of the i-th interval as specified in Definition 10. Hence each g_i is upper-bounded by $N_v/((1+\varepsilon)^{i-1} \cdot \varepsilon n/k) \leq k/((1+\varepsilon)^{i-1} \cdot \varepsilon)$ if $i \in \{1, \ldots, t\}$, and N_v if $i = 0$. Therefore the total number of signatures $\vec{g}$ considered for a vertex v is upper-bounded by

$$N_v \cdot \prod_{i=1}^{t} \frac{k}{(1+\varepsilon)^{i-1} \cdot \varepsilon} = N_v \left(\frac{k}{\varepsilon}\right)^t \cdot \left(\frac{1}{1+\varepsilon}\right)^{\frac{(t-1)t}{2}} \leq N_v \left(\frac{k}{\sqrt{\varepsilon}}\right)^t,$$

where the inequality holds since $t - 1 = \lceil \log_{1+\varepsilon}(1/\varepsilon) \rceil$. The latter value can be upper-bounded by $\lceil 1/\varepsilon \cdot \log(1/\varepsilon) \rceil$ if $\varepsilon \leq 1$, since then $1 + \varepsilon \geq 2^\varepsilon$. Hence we can conclude that the number of signatures is γN_v, where $\gamma \in \mathcal{O}((k/\sqrt{\varepsilon})^{1+\lceil \frac{1}{\varepsilon} \log(\frac{1}{\varepsilon}) \rceil})$.

We bound the runtime as follows. For each vertex v we calculate the number of steps T_v that are needed to compute all entries $C_{v'}(\vec{g}, m)$ for all $v' \in L_v$. We claim that $T_v \leq \frac{3}{2}\gamma^2 N_v^4$ for any vertex v. According to Formulas (1) and (2), in addition to the number of steps T_u and T_w to compute the tables for L_u and L_w, for each m and $\vec{g}$ the minimum value over two options is found by going through all possible x, $\vec{g}_u$, and $\vec{g}_w$. For any fixed x there are at most $\gamma N_u \cdot \gamma N_w$ many possibilities to combine vectors $\vec{g}_u$ and $\vec{g}_w$ to form a signature $\vec{g}$. Since m and x are both upper-bounded by N_v and $N_u + N_w \leq N_v$ we get

$$\begin{aligned} T_v &\leq T_u + T_w + 2\gamma^2 N_u N_w N_v^2 \\ &\leq \frac{3}{2}\gamma^2 N_v^2 \left(N_u^2 + N_w^2 + 2N_u N_w\right) \\ &\leq \frac{3}{2}\gamma^2 N_v^4. \end{aligned}$$

Since the time to compute Formula (3) for each signature is $\mathcal{O}(\gamma n)$, we conclude that the total runtime is $\mathcal{O}(\gamma^2 n^4) = \mathcal{O}(n^4 (k/\sqrt{\varepsilon})^{2+2\lceil \frac{1}{\varepsilon} \log(\frac{1}{\varepsilon}) \rceil})$, which is polynomial if ε is constant. $\square$

3.2 The Packing Phase

The second stage of the algorithm attempts to pack each set of connected components $\mathcal{S} \in \mathbb{S}$, computed by the first stage, 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 [19], which we briefly describe as presented in [35].

Let $\mathcal{S} \in \mathbb{S}$ be a set of connected components with signature $\vec{g} = (g_0, \dots, g_t)$. First the algorithm constructs an instance I of the `BIN PACKING` problem containing only the components of $\mathcal{S}$ larger than $\varepsilon\lceil n/k \rceil$. In particular, the bin capacity is set to be $\lceil n/k \rceil$ and for every entry $1 \le i \le t$ of $\vec{g}$, g_i elements of size $(1+\varepsilon)^{i-1} \cdot \varepsilon\lceil n/k \rceil$ are introduced in I. That is, the size of each component is converted to the lower endpoint of the interval which contains it according to Definition 10. The number of elements in I is $\sum_{i \ge 1} g_i \le n/(\varepsilon\lceil n/k \rceil) \le k/\varepsilon$ since there are n vertices in V and the smallest size of an element in I is $\varepsilon\lceil n/k \rceil$. An optimal bin packing for I can be found in $\mathcal{O}((k/\varepsilon)^{2t})$ time, using a dynamic programming scheme (for more details see [19, 35]). That is, the runtime is exponential in the number t of different sizes of the elements. A packing of I into the minimum number of bins of capacity $\lceil n/k \rceil$ translates into a packing of the components of $\mathcal{S}$ larger than $\varepsilon\lceil n/k \rceil$ into bins of capacity $(1+\varepsilon)\lceil n/k \rceil$. This is besause each element in I underestimates the size of the component in $\mathcal{S}$ that it represents by a factor of at most $1+\varepsilon$.

To complete the packing of $\mathcal{S}$ the algorithm distributes the remaining components of size less than $\varepsilon\lceil n/k \rceil$ by greedily putting them into bins without exceeding the capacity of $(1+\varepsilon)\lceil n/k \rceil$. A new bin is created if there is no room for a component in any of the open bins. Distributing the remaining components can be performed in $\mathcal{O}(n)$ time. Let $\varphi(\mathcal{S})$ denote the number of bins that this algorithm needs to pack $\mathcal{S}$. Note that for two sets of components having the same signature the components larger than $\varepsilon\lceil n/k \rceil$ will always be distributed in the same way by the algorithm. However the greedy distribution of the remaining small components may create more bins for one of the sets. We show next that if a set of components computed by the first stage has the same signature $\vec{g}^*$ as the set of components induced by an optimal perfectly balanced partition, then the second stage of the algorithm packs it into at most k bins with capacity $(1+\varepsilon)\lceil n/k \rceil$.

Lemma 12. *Let $\mathcal{S}^*$ having signature $\vec{g}^*$ be the set of connected components in an optimal perfectly balanced partition. For the set $\mathcal{S} \in \mathbb{S}$ with signature*

$\vec{g}^*$ *it holds that* $\varphi(\mathcal{S}) \leq k$.

Proof. We distinguish two cases for the greedy distribution of the components of $\mathcal{S}$ that have size less than $\varepsilon\lceil n/k \rceil$ depending on whether or not new bins are created. If no new bins are created then $\varphi(\mathcal{S})$ is solely determined by the output of the bin packing algorithm, run with capacities $\lceil n/k \rceil$ on the instance I. Since $\mathcal{S}^*$ has the same signature $\vec{g}^*$ as $\mathcal{S}$, all elements $e_i \in I$ can be mapped to distinct components $S_i \in \mathcal{S}^*$ such that $e_i \leq |S_i|$. Hence any packing of $\mathcal{S}^*$ into bins of capacity $\lceil n/k \rceil$ requires at least $\varphi(\mathcal{S})$ many bins which is optimal for I. Since $\mathcal{S}^*$ requires at most k optimally packed bins by definition, this proves the claim in the case no new bins are opened.

If new bins are created by the greedy step, then at least the first $\varphi(\mathcal{S}) - 1$ bins of the final solution are filled beyond the extent of $\lceil n/k \rceil$. Otherwise small components of size at most $\varepsilon\lceil n/k \rceil$ could have been fit without requiring the creation of the last bin. Therefore the total number of vertices in $\mathcal{S}$ strictly exceeds $(\varphi(\mathcal{S}) - 1)\lceil n/k \rceil$. Since the total number of vertices contained in $\mathcal{S}^*$ equals that of $\mathcal{S}$, it follows that at least $\varphi(\mathcal{S})$ bins are required to pack $\mathcal{S}^*$ into bins of capacity $\lceil n/k \rceil$. This proves the claim in the case new bins were created by the greedy step. $\square$

The final step of the algorithm is to output the packing of a set $\mathcal{S} \in \mathbb{S}$ of minimum cut cost among those with $\varphi(\mathcal{S}) \leq k$. The next theorem proves correctness and bounds the runtime of the algorithm.

Theorem 13. *For any tree T with positive edge weights, $\varepsilon > 0$, and $k \in \{1, \ldots, n\}$, 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 cost is at most that of an optimal perfectly balanced partition of the tree. Furthermore the runtime is polynomial if ε is a constant.*

Proof. Let $\widetilde{\mathcal{S}} \in \mathbb{S}$ be the set of connected components returned by the algorithm, i.e. if $C(\vec{g})$ denotes the cut cost of the set $\mathcal{S} \in \mathbb{S}$ with signature $\vec{g}$, then

$$\widetilde{\mathcal{S}} = \arg_{\mathcal{S} \in \mathbb{S}} \min\{C(\vec{g}) \mid \mathcal{S} \text{ has signature } \vec{g} \wedge \varphi(\mathcal{S}) \leq k\}. \tag{4}$$

By Lemma 12 we know that if $\mathcal{S} \in \mathbb{S}$ has signature $\vec{g}^*$ then $\varphi(\mathcal{S}) \leq k$. Thus the minimisation of (4) ensures that the cut cost of $\widetilde{\mathcal{S}}$ is at most that of a set of components $\mathcal{S} \in \mathbb{S}$ with signature $\vec{g}^*$. Since $\mathbb{S}$ retains the set of components with minimum cut cost among all those having the same signature, it follows that the cut cost of $\widetilde{\mathcal{S}}$ is at most that of $\mathcal{S}^*$, which concludes the proof of correctness.

To bound the runtime of the algorithm, recall from the proof of Theorem 11 that the total number of considered signatures $\vec{g}$ is γn, where $\gamma \in \mathcal{O}((k/\sqrt{\varepsilon})^{1+\lceil \frac{1}{\varepsilon}\log(\frac{1}{\varepsilon})\rceil})$. By Theorem 11 the set $\mathbb{S}$, whose size is at most γn, can be computed in time $\mathcal{O}(n^4\gamma^2)$. Each of the sets of components of $\mathbb{S}$ requires at most $\mathcal{O}((k/\varepsilon)^{2t}+n)$ time to be packed in the second stage of the algorithm. Hence the second stage can be performed in $\mathcal{O}(\gamma n((k/\varepsilon)^{2t}+n))$ total time. This means that the overall runtime of the algorithm is $\mathcal{O}(n^4(k/\varepsilon)^{1+3\lceil \frac{1}{\varepsilon}\log(\frac{1}{\varepsilon})\rceil})$, which concludes the proof. □

4 Extension to General Graphs

In this section we present an algorithm that uses the PTAS given in Section 3 to find a near-balanced partition of an edge-weighted graph G. The algorithm presented computes a cut cost of at most $\alpha \in \mathcal{O}(\log n)$ times that of an optimal perfectly balanced partition of G. It relies on using the PTAS of Section 3 to compute near-balanced partitions of a set of decomposition trees that well approximate the cuts in G. This decomposition can be found using the results by Räcke [30]. An in-depth discussion of decomposition trees can be found in [27, 30, 36]. Here we only introduce the basic concepts needed for our purposes.

For a graph $G = (V, E, \omega)$ with weight function $\omega : E \to \mathbb{R}^+$, the quality of a *cut* $W \subseteq V$ is measured using its *cut cost* $C(W)$. It is defined to be the sum of the weights of the edges connecting vertices in W and $V \setminus W$. A *decomposition tree* of G is a tree $T = (V_T, E_T, \omega_T)$, with weight function $\omega_T : E_T \to \mathbb{R}^+$, 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. We also define a *leaf cut* $K \subseteq L$ of a tree and its *cut cost* $C(K)$ is the minimum weight of edges in E_T that have to be removed in order to disconnect K from $L \setminus K$. We map cuts W on G to leaf cuts K on its decomposition tree 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 leverage the following properties of decomposition trees which are proved in [27, 30, 36].

Theorem 14. *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 $C(K_i) \leq \mathcal{O}(\log n) \cdot C(W)$. That is, in at least one tree T_i the cut structure of G is not distorted too much. This allows for a fast approximation of balanced partitions in graphs using a modified version of the PTAS given in Section 3. To this end the PTAS must be adapted to compute near-balanced *leaf partitions* of each T_i. That is, the adapted PTAS 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 cost* $C(\mathcal{L})$ 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.

The adaptation of the PTAS given in Section 3 to compute near-balanced leaf partitions is straightforward (see also [36] for the corresponding adaption of the `BISECTION` algorithm for trees). First, signatures need to count leaves instead of vertices in Definition 10. Also, 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). The following result summarises the properties of the adapted PTAS.

Corollary 15. *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 cost 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.*

Next we show our main result. It entails that the adapted PTAS can be used to compute near-balanced partitions for general graphs so that the cut cost deviates by only a logarithmic factor from the optimum.

Theorem 16. *Let $G = (V, E, \omega)$ be a graph with n vertices, $\varepsilon > 0$ be a constant, and $k \in \{0, \ldots, n\}$. There is an algorithm that computes a partition of V into k sets such that each set has size at most $(1+\varepsilon)\lceil n/k \rceil$ and its cut* *cost is at most $\mathcal{O}(\log n)$ times that of the optimal perfectly balanced solution. The runtime is polynomial if ε is constant.*

Proof. We use Theorem 14 to compute a family of decomposition trees with the properties listed therein. This family has a polynomial number of trees since the runtime is polynomial. For each such tree we compute a partition of its leaves into k sets of size at most $(1+\varepsilon)\lceil n/k\rceil$ using Corollary 15. We select the computed leaf partition $\mathcal{L}^*$ of the decomposition tree T^* having the smallest cut cost when applied to G. That is, $\mathcal{L}^*$ minimises the quantity $C(m_G(\mathcal{L}))$ among all computed leaf partitions, where for a leaf partition $\mathcal{L} = \{L_1, \ldots, L_k\}$ we define $m_G(\mathcal{L}) = \{m_G(L_1), \ldots, m_G(L_k)\}$. The output of the algorithm is then the vertex partition $m_G(\mathcal{L}^*)$ of G.

By Theorem 14, for some decomposition tree T' and the corresponding leaf partition $m_{T'}(\mathcal{V}^*) = \{m_{T'}(V_1^*), \ldots, m_{T'}(V_k^*)\}$ of the optimal perfectly balanced partition $\mathcal{V}^* = \{V_1^*, \ldots, V_k^*\}$, we get the upper bound $C(m_{T'}(\mathcal{V}^*)) \leq \mathcal{O}(\log n) \cdot C(\mathcal{V}^*)$. This is due to the observation that for any (vertex or leaf) partition $\mathcal{X} = \{X_1, \ldots, X_k\}$ it holds that $C(\mathcal{X}) = \frac{1}{2}\sum_{j=1}^{k} C(X_j)$. By Corollary 15 we know that the cut cost of the computed leaf partition $\mathcal{L}'$ in T' is at most the cut cost of the optimal perfectly balanced leaf partition in T', hence $C(\mathcal{L}') \leq C(m_{T'}(\mathcal{V}^*))$. From the lower bound in Theorem 14 we can conclude that $C(m_G(\mathcal{L}')) \leq C(\mathcal{L}')$. Finally, since we chose $\mathcal{L}^*$ to minimise the quantity $C(m_G(\mathcal{L}))$ among all computed leaf partitions, we get $C(m_G(\mathcal{L}^*)) \leq C(m_G(\mathcal{L}'))$. This implies $C(m_G(\mathcal{L}^*)) \leq \mathcal{O}(\log n) \cdot C(\mathcal{V}^*)$, and concludes the proof. □

5 Conclusions

In this article, we studied the k-`BALANCED PARTITIONING` problem on tree instances. Typically, NP-hard graph problems become trivial when restricted to trees. For k-`BALANCED PARTITIONING` however we showed that trees are an interesting benchmark, for various reasons.

When a perfectly balanced solution is required, we showed that even if the maximum degree of the tree is constant the k-`BALANCED PARTITIONING` problem is hard to approximate. Even more disheartening, on unrestricted unweighted connected trees no approximation algorithm can substantially outperform the trivial one that cuts all edges of the tree. This behavior matches that of general graphs, for which no algorithm with bounded approximation ratio exists. In this sense, trees represent a simple unit that still captures 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 cost 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 to be contrasted with the strong inapproximability results for the case of unrestricted k when the solutions are to be perfectly balanced.

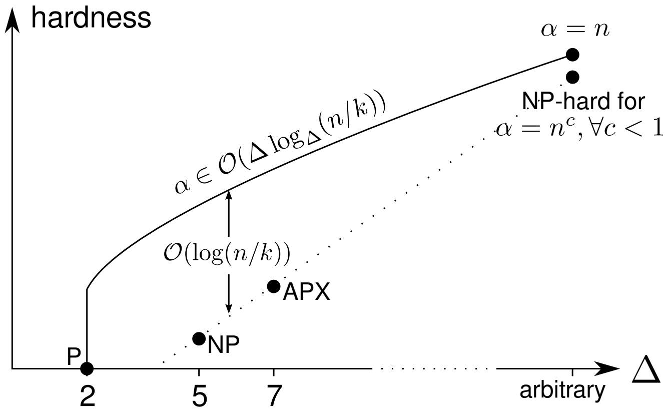

Figure 8: The hardness of trees versus their maximum degree Δ. The hardness results from Section 2 (large dots) indicate that the hardness grows with Δ (dotted line). A modification of MacGregor's [26] algorithm yields an approximation of $\mathcal{O}(\Delta \log_\Delta(n/k))$ (solid line). This means there is a gap of size $\mathcal{O}(\log(n/k))$ between the lower and upper complexity bounds in case Δ is constant.

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 8). 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 [26], remains open. In particular, Theorem 5 does not rule out an algorithm that approximates the cut cost by the factor $\alpha = 25/24$ if $\Delta \leq 7$.

For near-balanced partitions it remains to be seen what other graph classes permit better approximation ratios on the cut size than those known for general graphs. It is worth mentioning though that the dynamic program of the PTAS presented in this article can be generalised to graphs with bounded tree-width, using standard techniques [34]. Hence also for these graphs a factor of $\alpha = 1$ is achievable.

Another performance metric that may be improved in our results is the runtime. We showed that we can achieve approximations on the cut cost that do not depend on ε, whereas the dependency on ε in the runtime is quite conspicuous. This begs the question whether an algorithm offering a better tradeoff in the dependency of ε between runtime and cut cost can be found. A recent result [12] 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 *fully* polynomial time algorithm with the following properties: The computed solution is near-balanced and the cut cost 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 $\varepsilon = 1$ and the case $\varepsilon < 1$, studied in this article (recall Figure 2). For the case $\varepsilon = 1$ the algorithm by Krauthgamer *et al.* [22] 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 $\varepsilon = 1$ do not extend to $\varepsilon < 1$. Furthermore, the graph decomposition results that 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. *Theory of Computing Systems*, 39(6):929–939, 2006.

[2] P. Arbenz, G. van Lenthe, U. Mennel, R. Müller, and M. Sala. Multi-level μ-finite element analysis for human bone structures. In *Proceedings of the 8th Workshop on State-of-the-art in Scientific and Parallel Computing (PARA)*, pages 240–250, 2007.

[3] 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 (STOC)*, pages 222–231, 2004.

[4] N. Bansal, D. Coppersmith, and B. Schieber. Minimizing setup and beam-on times in radiation therapy. In *Proceedings of the 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 27–38, 2006.

[5] S.N. Bhatt and F. T. Leighton. A framework for solving VLSI graph layout problems. *Journal of Computer and System Sciences*, 28(2):300–343, 1984.

[6] H. Bodlaender, P. Schuurman, and G. Woeginger. Scheduling of pipelined operator graphs. *Journal of Scheduling*, 15:323–332, 2012.

[7] D. Delling, A. Goldberg, T. Pajor, and R. Werneck. Customizable route planning. *Experimental Algorithms*, pages 376–387, 2011.

[8] J. Díaz, J. Petit, and M. J. Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34(3):313–356, 2002.

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

[10] G. Even, J.S. Naor, S. Rao, and B. Schieber. Fast approximate graph partitioning algorithms. *SIAM Journal on Computing*, 28(6):2187–2214, 1999.

[11] A. E. Feldmann. *Balanced Partitioning of Grids and Related Graphs: A Theoretical Study of Data Distribution in Parallel Finite Element Model Simulations*. PhD thesis, ETH Zurich, April 2012. Diss.-Nr. ETH: 20371.

[12] A. E. Feldmann. Fast balanced partitioning is hard, even on grids and trees. In *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 2012.

[13] A. E. Feldmann, S. Das, and P. Widmayer. Restricted cuts for bisections in solid grids: A proof via polygons. In *Proceedings of the 37th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 143–154, 2011.

[14] A. E. Feldmann and L. Foschini. Balanced partitions of trees and applications. In *29th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 100–111, 2012.

[15] 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 (ESA)*, pages 143–154, 2011.

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

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

[18] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co., 1979.

[19] 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.

[20] D. G. Kirkpatrick and P. Hell. On the completeness of a generalized matching problem. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC)*, pages 240–245, 1978.

[21] 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 (STOC)*, pages 682–690, 1993.

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

[23] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics*, 22(3):277–286, 2003.

[24] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, 1999.

[25] R.J. Lipton and R.E. Tarjan. Applications of a planar separator theorem. *SIAM Journal on Computing*, 9:615–627, 1980.

[26] R. M. MacGregor. *On Partitioning a Graph: a Theoretical and Empirical Study.* PhD thesis, University of California, Berkeley, 1978.

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

[28] J. K. Park and C. A. Phillips. Finding minimum-quotient cuts in planar graphs. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC)*, pages 766–775, 1993.

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

[30] H. Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, 2008.

[31] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[32] D.B. Shmoys. Cut problems and their application to divide-and-conquer. In D. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, pages 192–235. PWS Publishing Co., 1996.

[33] H. D. Simon and S. H. Teng. How good is recursive bisection? *SIAM Journal on Scientific Computing*, 18(5):1436–1445, 1997.

[34] K. Soumyanath and J. S. Deogun. On the bisection width of partial k-trees. In *Proceedings of the 20th Southeastern Conference on Combinatorics, Graph Theory, and Computing*, volume 74 of *Congressus Numerantium*, pages 25–37, 1990.

[35] V. V. Vazirani. *Approximation Algorithms*. Springer, 2003.

[36] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[37] Z. Wu and R. Leahy. An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1101–1113, 1993.