Balanced partitions of trees and applications

Balanced Partitions of Trees and Applications [*]

Andreas Emil Feldmann[†] Luca Foschini[‡]

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 finite 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.

[*]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.

[†]Institute of Theoretical Computer Science, ETH Zürich, Switzerland

[‡]Department of Computer Science, U.C. Santa Barbara, Santa Barbara, USA

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 PARTITIONING` is a notoriously hard problem. The special case of $k = 2$, commonly known as the `BISECTION` problem, is already NP-complete. 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 [3]. 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 the size of the vertex sets is allowed to be off by a factor of two from the perfectly balanced solution. For this case, the best result is by Krauthgamer *et al.* [16] 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 would be 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 [3] 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 [3], 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 $\varepsilon \geq 1$ [6, 16] 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 [3], we show that it is NP-hard to approximate the cut-size within any factor better than $\alpha = n^c$ for any

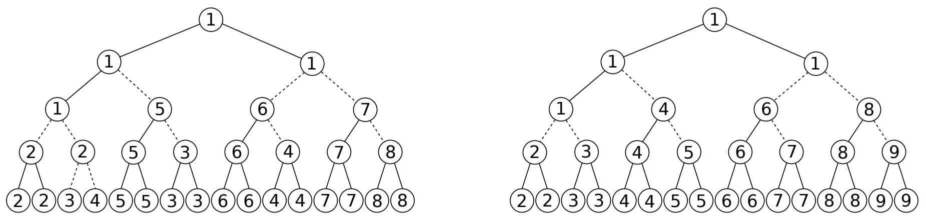

Figure 1: Two optimally partitioned binary trees. For the tree on the left $k = 8$ (with a cut-size of 10) whereas $k = 9$ (with a cut-size of 8) for the tree on the right. The numbers in the vertices indicate the set they belong to and the cut-set is represented by the dashed edges.

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 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 $2 < \Delta < 5$ at which k-`BALANCED PARTITIONING` becomes hard, is an interesting open problem. These results should be contrasted with a greedy algorithm by MacGregor [18] 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 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 [14].

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 [3] 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 [3]: the ad-hoc approach of [3] 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. Our paper directly extends the results in [3], 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 [3] 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.* [6] present an algorithm with $\alpha \in \mathcal{O}(\log n)$ using spreading metrics techniques. Later Krauthgamer *et al.* [16] 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 [15][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, has been well studied. It is NP-hard in the general case [11] but polynomial time approximations on the cut-size are known. For instance Räcke [22] gives an algorithm with approximation ratio $\alpha \in \mathcal{O}(\log n)$ (and $\varepsilon = 0$). For near-balanced partitions Leighton and Rao [17] 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 [17] it was shown that $\gamma \in \mathcal{O}(\log n)$, and this result was improved by Arora *et al.* [4] to $\gamma \in \mathcal{O}(\sqrt{\log n})$. For planar graphs it is possible to achieve a constant min-ratio approximation [20]. If a perfectly balanced solution is required for planar graphs, Dìaz *et al.* 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 [20], 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 [5]. For instance an $\mathcal{O}(n^4)$ algorithm for grid graphs without holes has been found [7], while for trees an $\mathcal{O}(n^2)$ algorithm [12, 18] exists. Substantial work on graph partitioning has leveraged spectral methods. Alon and Milman [1] presented a linear time algorithm for the sparsest cut problem with approximation ratio depending on the conductance (which could be $\Omega(n)$ in the worst case). Spielman and Teng ([23] and followup [2]) extended the approach of [1] to find balanced separators, however, the dependence on the conductance both for the runtime and approximation ratio remains.

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 [18] is easily generalised to solve the k-`BALANCED PARTITIONING` problem for any constant k in polynomial time. However the runtime will then be exponential in k. At the other end of the spectrum, i.e. when $k \in \Theta(n)$, it is known that the problem is NP-hard [14] for any $k \leq n/3$. Feo and Khellaf [9] give a $\alpha = n/k$ approximation algorithm for the cut-size which was improved [8] to $\alpha = 2$ in case k equals $n/3$ or $n/4$. Figure 2 summarises the related work and the results presented here.

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 [19] for a survey). In [22], Räcke introduces an optimal decomposition with factor $\mathcal{O}(\log n)$, which we employ in the present work. In a recent work, Madry [19] shows that it is possible to generalise Räcke's insights so that any *cut based* problem (see [19] for more details) is solvable on graphs by computing solutions on tree decompositions of the input graph.

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 maximum degree are restricted to be constant. All reductions are from `MAX-3-PARTITION`, defined as follows.

Definition 1 (`MAX-3-PARTITION`)**.** Given $3k$ integers $a_1, \dots, a_{3k}$ 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.

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

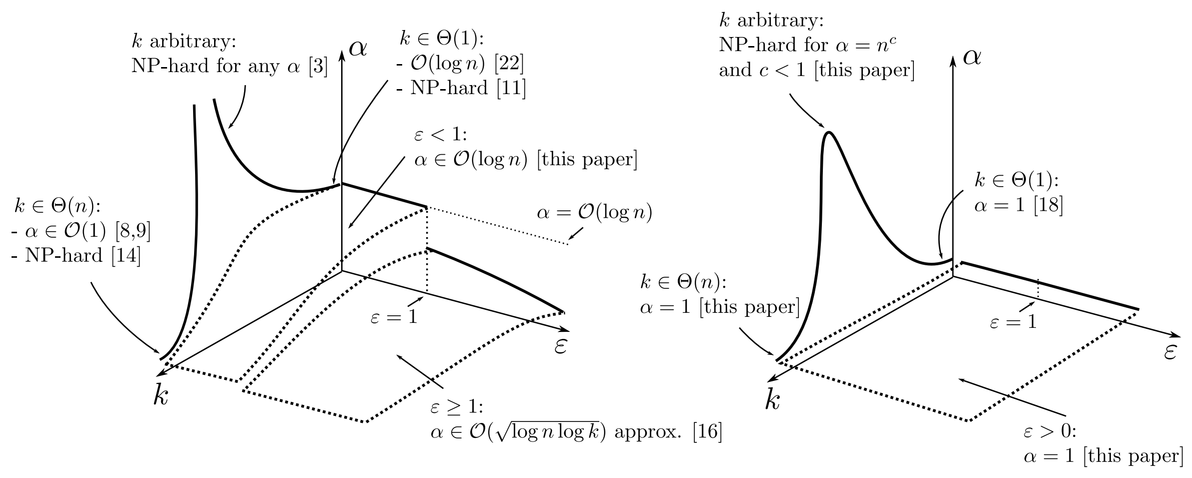

Figure 2: Illustrations of best approximation factor α known vs k and ε, for general graphs (left) and trees (right). The plane (α, k) represents the case of perfectly balanced solutions ($\varepsilon = 0$) and shows that the restriction to trees does not significantly change the asymptotic behaviour. However, when the balance constraints are relaxed ($\varepsilon > 0$) a much better (bi-criteria) approximations can be devised for trees. This remarkable behavior can be partially transplanted 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 `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 [10] and the results on the `3D-MATCHING` problem by Petrank [21] [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` 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 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 [3] for general graphs. The result holds even if the diameter of the tree is bounded by a constant.

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.*

Proof. We call an instance to the `MAX-3-PARTITION` problem a *NO instance* if less than k triples of the integers can be found that sum up to s and a *YES instance* otherwise.

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 `MAX-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. We now argue that an optimal perfectly balanced partition has cut-size at most $3k - 1$ if and only if I is a YES instance and it requires at least $m = 3kn^c$ cuts otherwise. This would allow an

[2]We thank Nikhil Bansal for pointing out the connection between the reductions in [10] and the results by Petrank [21]

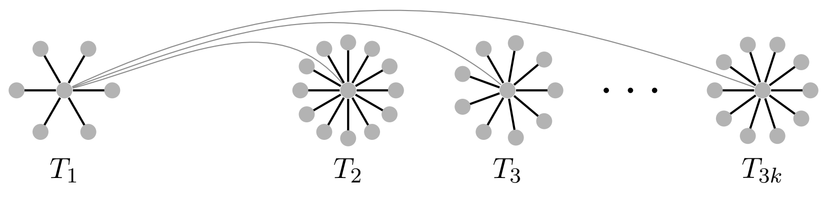

Figure 3: Construction for the reduction of Theorem 2. Thin grey edges link star centres, darkened edges connect centres to leaves.

approximation algorithm with approximation factor at most n^c to decide between a YES and a NO instance of the `MAX-3-PARTITION` problem. Since the latter is NP-hard this proves the theorem.

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 $\{V_1,\ldots,V_k\}$, 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-size 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 $|V_l|$ 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. 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 technique (and quite more involved) 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 to construct gadgets with structural properties that guarantee more edges to be cut the less integers can be put 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 step. As a consequence all integers are divisible by 4 and $s > 20$, which will become important later in

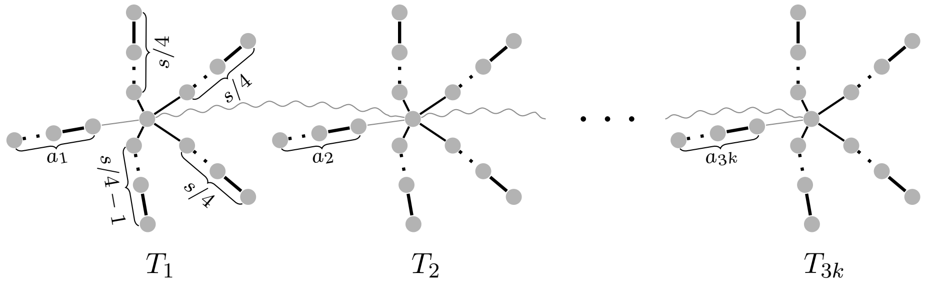

Figure 4: Construction for Theorem 3. Each gadget T_i is composed by an a_i-path connected to the root of an s-tree through an edge from A (straight grey). Each s-tree branches into four paths of (almost) the same length. Two adjacent gadgets in a path are connected through the roots of their s-trees with an edge from B (wiggled grey).

the proof. 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 roots of the s-trees in a path, as shown in Figure 4. 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 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 $|C| \geq 2|(A \cup B) \setminus C^*|$.*

Proof. The crucial observation leading to the claim is that in any $4k$-balanced partition of T, after removing the cut-set C^* every connected component has size at most s. We distinguish two cases. First consider an edge $e \in A \setminus C^*$ adjacent only to cut edges in $B \cap C^*$. Let T_i be the gadget in which e is contained. The tree T_i has a total size of $s + a_i$. Since $a_i > s/4$ and each branch of the s-tree in T_i has at most $s/4$ vertices, there must be at least two edges from C in T_i in order to cut away a_i vertices.

Otherwise consider a connected component W that contains uncut edges from $(A \cup B) \setminus C^*$. Let A' and B' denote the edges in W from A and B, respectively. Notice that $|B'| \geq 1$ since the other case was considered above. Let T' be the subtree of T that results by extending W with all the a_i-paths incident to an edge in A' and all the s-trees incident to an edge in B' (see

Figure 5). Each branch of an s-tree and each a_i-path in T' has at least $s/4-1$ vertices. Hence if at least 5 branches of s-trees or a_i-paths were fully included in W this connected component would contain more than s vertices, as $s>20$—a contradiction. Therefore there are at most 4 such included branches. Since T' contains at least $4(|B'|+1)$ s-tree branches and $|A'|$ a_i-paths in T', we can hence 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 then obtain $|A'|+4|B'|\geq 2|A'|-1+3|B'|\geq 2|A'\cup B'|$, which proves our claim for the tree T'. Since the gadgets in any possible T' and the gadgets considered in the first case are pairwise disjoint, this concludes the proof. □

Consider the following algorithm $\mathcal{A}$ to reposition cut edges from a $4k$-balanced partition into an approximate solution 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 only 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', the sizes of the sets in the partition induced by the cut-set are modified and the balance might be lost. In particular, when a cut edge is removed by $\mathcal{A}$, two connected components induced by the cut-set are joined to form a single component and moved to 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, 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 4: instead of connecting the s-trees through their roots, the B edges

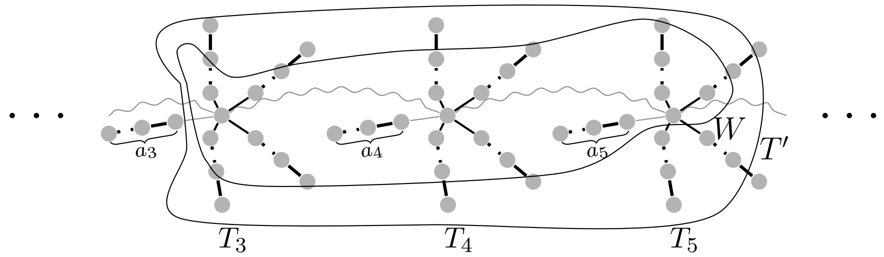

Figure 5: Part of the construction of Theorem 3 with components W and T' highlighted

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. We defer the detailed proof to the full version of this paper.

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 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 $x \in \{1, \dots, \lceil n/k \rceil\}$. 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 $\vec{g} = (g_0, \dots, g_t)$, 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-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 [13, 24] 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.

3.1 The First Stage

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 6). 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.

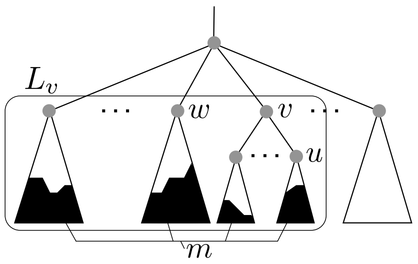

Figure 6: A part of a tree in which a vertex v, its rightmost child u, its predecessor w, the set of vertices L_v, and the m covered vertices by some lower frontier with signature $\vec{g}$ are indicated.

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 frontier $\mathcal{F}^* \setminus \{S\}$ of $L_u \cup L_w$ with signature $\vec{g}_u + \vec{g}_w$ must have minimum cut-size. 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 programming scheme that computes the wanted solution set $\mathbb{S}$ in polynomial-time if ε is a constant, as the next theorem shows.

Theorem 7. *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 r and has some size x. Clearly x is at least 1 and at most μ. If q denotes the rightmost child of the root r the cut-size of the optimal solution for $\vec{g}$ can thus be computed in linear time using the formula

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

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 6. 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 $t - 1 = \lceil \log_{1+\varepsilon}(1/\varepsilon) \rceil$. Because the latter value can be upper bounded by $\lceil 1/\varepsilon \cdot \log(1/\varepsilon) \rceil$ if $\varepsilon \leq 1$, since then $1 + \varepsilon \geq 2^\varepsilon$, 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 the formulas in (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

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

Since the time to compute formula (3) for each signature is $\mathcal{O}(\gamma n)$, we conclude that the total runtime is $\mathcal{O}(\gamma^2 n^4)$, which is polynomial if ε is constant. □

3.2 The Second Stage

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 [13], which we briefly describe as presented in [24].

Let $\mathcal{S} \in \mathbb{S}$ be a set of connected components with signature $\vec{g} = (g_0, \ldots, g_t)$. First the algorithm constructs an instance I of the `BIN PACKING` problem comprising 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 $1 \leq i \leq t$ 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 6. The number of elements in I is $\sum_{i \geq 1} g_i \leq n/(\varepsilon\lceil n/k \rceil) \leq k/\varepsilon$ 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 time $\mathcal{O}((k/\varepsilon)^{2t})$, i.e. it is exponential in the number t of different sizes of the elements (for more details see [13, 24]). A packing of I into the minimum number of bins of capacity $\lceil n/k \rceil$ translates into a packing of the components larger than $\varepsilon\lceil n/k \rceil$ of $\mathcal{S}$ into bins of capacity $(1+\varepsilon)\lceil n/k \rceil$, since 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 placing a component in any bin would exceed the capacity. 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 8. *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 $\lceil n/k \rceil$, which 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-size among those with $\varphi(\mathcal{S}) \leq k$. The next theorem proves correctness and bounds the runtime of the algorithm.

Theorem 9. *For any tree T, $k \in \{1, \dots, n\}$, 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.*

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-size 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 8 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-size 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-size among all those having the same signature, it follows that the cut-size of $\widetilde{\mathcal{S}}$ is at most that of $\mathcal{S}^*$, which concludes the proof of correctness.

To bound the runtime of the second stage, recall 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 7 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, which concludes the proof. □

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 [22]. The reason why this process yields a $\mathcal{O}(\log n)$ 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 sum over the capacities of those edges in E_T that are necessary to disconnect K from $L \setminus K$. We make use of the following result which can be found in [19, 22].

Theorem 10. *For any graph $G = (V, E, u)$, 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 cuts 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 $\mathcal{L} = \{L_1, \dots, L_k\}$ of the leaves L of a tree T into k sets of size at most $(1+\varepsilon)\lceil n/k \rceil$ each. The *cut-size* $C(\mathcal{L})$ in this case is the minimum sum of the capacities of those edges that are necessary to disconnect each $L_i \in \mathcal{L}$ from all other leaves. The PTAS given in Section 3 can be adapted to compute near-balanced leaf partitions in a natural way: 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. This yields the following result.

Corollary 11. *For any tree T, $\varepsilon > 0$, and $k \in \{1, \ldots, n\}$, there is an algorithm that computes a partition of T's leaves into k sets such that each set includes at most $(1+\varepsilon)\lceil n/k \rceil$ leaves and its cut-size is at most that of an optimal perfectly balanced leaf partition. Furthermore the runtime is polynomial 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 12. *Let $G = (V, E, u)$ be a graph, $\varepsilon > 0$ be a constant, and $k \in \{0, \ldots, n\}$. Also let $\mathcal{V}^*$ be an optimal perfectly balanced partition of G. 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 $\mathcal{O}(\log n) \cdot C(\mathcal{V}^*)$.*

Proof. We use Theorem 10 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 11. We select the computed leaf partition $\mathcal{L}^*$ of the decomposition tree T^* having the smallest cut-size 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 10, 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 $C(m_{T'}(\mathcal{V}^*)) \leq \mathcal{O}(\log n) \cdot C(\mathcal{V}^*)$. By Corollary 11, we know that the cut-size of the computed leaf partition $\mathcal{L}'$ in T' is at most the cut-size 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 10, using the observation that for any partition $\mathcal{X} = \{X_1, \ldots, X_k\}$ it holds that $C(\mathcal{X}) = \frac{1}{2}\sum_{j=1}^{k} C(X_j)$, 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 Conclusion

In this paper, we demonstrate the benefits of studying the k-`BALANCED PARTITIONING` on restricted graph instances, in this particular case trees. When a perfectly balanced solution is required, we show 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 [3] 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 solution 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 7). 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 [18], remains open. In particular, Theorem 3 does not rule out an algorithm that approximates the cut-size by the factor $\alpha = 25/24$ if $\Delta \leq 7$.

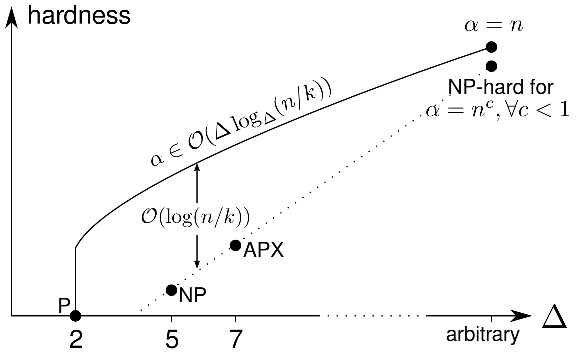

Figure 7: A plot of the hardness of partitioning trees versus their maximum degree Δ: the three hardness results from Section 2 (big dots) indicate that the hardness grows with Δ (dotted line). On the positive side a modification of MacGregor's [18] algorithm yields an approximation factor of $\mathcal{O}(\Delta \log_\Delta(n/k))$ (continuous line). In particular 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 general graphs and near-balanced partitions, the main challenge we see 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.* [16] 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] N. Alon and V. Milman. Isoperimetric inequalities for graphs, and superconcentrators. *Journal of Combinatorial Theory, Series B*, 38(1):73–88, 1985.

[2] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer*, page 475486. IEEE Computer Society, 2006.

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

[4] 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. ACM, 2004.

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

[6] G. Even, J. Naor, S. Rao, and B. Schieber. Fast approximate graph partitioning algorithms. In *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, SODA '97, pages 639–648, Philadelphia, PA, USA, 1997. Society for Industrial and Applied Mathematics.

[7] 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*, 2011.

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

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

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the Theory of NP-completeness.* W.H. Freeman and Co., San Fransisco, California, 1979.

[11] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

[12] M. Goldberg and Z. Miller. A parallel algorithm for bisection width in trees. *Computers & Mathematics with Applications*, 15(4):259–266, 1988.

[13] 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.

[14] 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. ACM, 1978.

[15] P. Klein, S. 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. ACM, 1993.

[16] 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. Society for Industrial and Applied Mathematics, 2009.

[17] 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.

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

[19] 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, oct. 2010.

[20] 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*, STOC '93, pages 766–775, New York, NY, USA, 1993. ACM.

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

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

[23] D. Spielman and S. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90. ACM, 2004.

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