Improved Approximation Algorithms for Balanced Partitioning Problems

Harald Räcke[1] **and Richard Stotz**[2]

1 **Technische Universität München, Garching, Germany**
raecke@in.tum.de
2 **Technische Universität München, Garching, Germany**
stotz@in.tum.de

Abstract

We present approximation algorithms for balanced partitioning problems. These problems are notoriously hard and we present new bicriteria approximation algorithms, that approximate the optimal cost and relax the balance constraint.

In the first scenario, we consider Min-Max k-Partitioning, the problem of dividing a graph into k equal-sized parts while minimizing the maximum cost of edges cut by a single part. Our approximation algorithm relaxes the size of the parts by $(1+\varepsilon)$ and approximates the optimal cost by $\mathcal{O}(\log^{1.5} n \log\log n)$, for every $0 < \varepsilon < 1$. This is the first nontrivial algorithm for this problem that relaxes the balance constraint by less than 2.

In the second scenario, we consider strategies to find a minimum-cost mapping of a graph of processes to a hierarchical network with identical processors at the leaves. This Hierarchical Graph Partitioning problem has been studied recently by Hajiaghayi et al. who presented an $(\mathcal{O}(\log n), (1+\varepsilon)(h+1))$ approximation algorithm for constant network heights h. We use spreading metrics to give an improved $(\mathcal{O}(\log n), (1+\varepsilon)h)$ approximation algorithm that runs in polynomial time for arbitrary network heights.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases graph partitioning, dynamic programming, scheduling

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.58

1 Introduction

The problem of scheduling the processes of a parallel application onto the nodes of a parallel system can in its full generality be viewed as a graph mapping problem. Given a graph $G = (V_G, E_G)$ that represents the *workload*, and a graph $H = (V_H, E_H)$ that represents the *physical computing resources*, we want to map G onto H such that on the one hand the processing load is well balanced, and on the other hand the communication load is small.

More concretely, nodes of G represent processes and are weighted by a weight function $w : V_G \to \mathbb{R}_+$. Edges of G represent communication requirements between processes, and are weighted by $c_G : E_G \to \mathbb{R}_+$. Nodes and edges of H represent processors and communication links, respectively, and may also be weighted with $w_H : V_H \to \mathbb{R}_+$ representing the processing capacity of a physical node, and $c_H : E_H \to \mathbb{R}_+$ representing the bandwidth capacity of a communication link. The goal is to map nodes of G to nodes of H, and to map edges of G to paths in H between the corresponding end-points such that a) every node in H obtains approximately the same weight and b) the communication load is small.

This general problem is very complex, and research has therefore focused on special cases. Many results can be interpreted as mapping to a star graph H with k leaves where the center

33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016).
Editors: Nicolas Ollinger and Heribert Vollmer; Article No. 58; pp. 58:1–58:14

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


node x has $w_H(x) = 0$, i.e., processes may not be mapped to the center ($c_H(e)$ is assumed to be uniform). When the goal is to minimize *total communication load* (sum over all edge loads in H) this problem is known as Min-Sum k-partitioning. You want to partition G into k disjoint pieces $V_1, \ldots, V_k$ such that the weight of any piece is at most $w(V_i) \leq w(V)/k$ but the total capacity of edges between partitions is minimized.

For this problem it is not possible to obtain meaningful approximation guarantees without allowing a slight violation of the balance constraints ($w(V_i) \leq w(V)/k$). Even et al. [5] obtain a bicriteria approximation guarantee of $(\mathcal{O}(\log n), 2)$, which means they obtain a solution that may violate balance constraints by a factor of 2, and that has a cost that is at most an $\mathcal{O}(\log n)$-factor larger than the cost of the optimum solution that does *not* violate balance constraints. This has been improved by Krauthgamer et al. [9] to $(\mathcal{O}(\sqrt{\log n \log k}), 2)$. If one aims to violate the balance constraints by less than 2, completely different algorithms seem to be required that are based on Dynamic Programming. Andreev and Räcke [1] obtain an $(\mathcal{O}(\log^{1.5} n), 1+\varepsilon)$-approximation which has been improved by Feldmann and Foschini to $(\mathcal{O}(\log n), 1+\varepsilon)$. For the case that the weight function w_H is non-uniform Krauthgamer et al. [10] obtain an $(\mathcal{O}(\log n), \mathcal{O}(1))$-approximation, which has been improved by Makarychev and Makarychev [11] to $(\mathcal{O}(\sqrt{\log n \log k}), 5+\varepsilon)$.

When the goal is to minimize the *congestion* (maximum load of a link) in the star network H, instead of the total communication load, the problem turns into the so-called Min-Max k-Partitioning problem. You want to partition a given graph G into k-parts $V_1, \ldots, V_k$ of equal size such that $\max_i c(V_i)$ is minimized, where $c(V_i)$ denotes the total weight of edges leaving V_i in G. For this problem Bansal et al. [2] obtain an $(\mathcal{O}(\sqrt{\log n \log k}), 2+\varepsilon)$-approximation but no non-trivial approximation that violates the balance constraint by less than 2 is known.

Hajiaghayi et al. [7] consider the mapping problem when H is not just a star but exhibits parallelism on many different levels. For example, a large parallel system may consist of many workstations, each equipped with several CPUs, etc. Such parallel systems are usually highly symmetric. Therefore, Hajiaghayi et al. model them as a tree in which all subtrees routed at a specific level are isomorphic. They develop an approximation algorithm for a Hierarchical Graph Partitioning problem which then gives rise to a scheduling algorithm for such a *regular tree topology* with an approximation guarantee of $(\mathcal{O}(\log n), (1+\varepsilon)(h+1))$. This algorithm runs in polynomial time as long as the height h of the hierarchy is constant.

1.1 Our Contribution

In this paper we first consider the Min-Max k-Partitioning problem and show how to obtain a polylogarithmic approximation on the cost while only violating the balance constraints by a factor of $1+\varepsilon$. For this we use a dynamic programming approach on trees, and then use an approximation of the graph by a *single* tree to obtain our result. Note that results that approximate the graph by a convex combination of trees [12] are not useful for obtaining an approximation for Min-Max k-Partitioning, as the objective function is not linear.

▶ **Theorem 1.** *There is a polynomial-time $(1+\varepsilon, 1+\varepsilon)$-approximation algorithm for Min-Max k-Partitioning on trees. This gives rise to an $(\mathcal{O}(\log^{1.5} n \log\log n), 1+\varepsilon)$-approximation algorithm for general graphs.*

Then we consider the Hierarchical Graph Partitioning problem as introduced by Hajiaghayi et al. [7]. We give a slight improvement on the factor by which the balance constraints are violated, while maintaining the same approximation w.r.t. cost. More crucially, our result also works if the height h is not constant.

▶ **Theorem 2.** *There exists an* $(\mathcal{O}(\log n), (1+\varepsilon)h)$ *approximation algorithm for Hierarchical Graph Partitioning whose running time is polynomial in* h *and* n*.*

Our technique is heavily based on spreading metrics and we show that a slight variation of the techniques in [5] can be applied.

At first glance the result of Theorem 2 does not look very good as the factor by which the balance constraints are violated seems to be very high as it depends on h. However, we give some indication that a large dependency on h might be necessary. We show that an approximation guarantee of $\alpha \leq h/2$ implies an algorithm for the following approximate version of parallel machine scheduling (PMS). Given a PMS instance I, with a set T of tasks, a length w_t, $t \in T$ for every task, and a deadline d, we define the *g-copied instance of I* ($g \in \mathbb{N}$) as the instance where each task is copied g times and the deadline is multiplied by g. The approximate problem is the following: Given I, either certify that I is not feasible or give a solution to the g-copied instance of I for some $g \leq \alpha$. It seems unlikely that this problem can be solved efficiently for constant α, albeit we do not have a formal hardness proof. These results have been deferred to the full version.

1.2 Basic Notation

Throughout this paper, $G = (V, E)$ denotes the input graph with $n = |V|$ vertices. Its edges are given a cost function $c : E \to \mathbb{N}$ and its vertices are weighted by $w : V \to \mathbb{N}$. For a subset of vertices $S \subseteq V$, $w(S)$ denotes the total weight of vertices in S and $c(S)$ denotes the total cost of edges leaving S, i.e., $c(S) = \sum_{e=\{x,y\}\in E,\ |\{x,y\}\cap S|=1} c(e)$. We will sometimes refer to $w(S)$ as the *size* of S and to $c(S)$ as its *boundary cost*. In order to simplify the presentation, we assume that all edge costs and vertex weights are polynomially bounded in the number of vertices n. This can always be achieved with standard rounding techniques at the loss of a factor $(1+\varepsilon)$ in the approximation guarantee.

A *partition* $\mathcal{P}$ of the vertex set V is a collection $\mathcal{P} = \{P_i\}_i$ of pairwise disjoint subsets of V with $\bigcup_i P_i = V$. We define two different cost functions for partitions, namely $\text{cost}^{\text{sum}}(\mathcal{P}) = \sum_i c(P_i)$ and $\text{cost}^{\max}(\mathcal{P}) = \max_i c(P_i)$. We drop the superscript whenever the type is clear from the context. A (k,ν)-*balanced* partition is a partition into at most k parts, such that the inequality $w(P_i) \leq \nu \cdot w(V)/k$ holds for all parts. The costs of the $(k,1)$-balanced partition with minimum cost w.r.t. cost^{sum} and $\text{cost}^{\max}$ are denoted with $\mathsf{OPT}^{\text{sum}}(k,G)$ and $\mathsf{OPT}^{\max}(k,G)$, respectively.

A *hierarchical partition* $\mathcal{H} = (\mathcal{P}_1, \ldots, \mathcal{P}_h)$ of height h is a sequence of h partitions, where $\mathcal{P}_{\ell+1}$ is a *refinement* of $\mathcal{P}_\ell$, i.e., for every $S \in \mathcal{P}_{\ell+1}$, there is a set $S' \in \mathcal{P}_\ell$ with $S \subseteq S'$. We call $\mathcal{P}_\ell$ the level-ℓ-partition of $\mathcal{H}$. For any cost vector $\vec{\mu} \in \mathbb{R}^h$, the cost of $\mathcal{H}$ is given by $\text{cost}_{\vec{\mu}}(\mathcal{H}) = \sum_{\ell=1}^{h} \mu_\ell \,\text{cost}^{\text{sum}}(\mathcal{P}_\ell)$. A $(\vec{k},\nu)$-*balanced* hierarchical partition is a hierarchical partition where $\mathcal{P}_\ell$ is (k_ℓ,ν)-balanced. The minimum cost of a $(\vec{k},1)$-balanced hierarchical partition w.r.t. $\vec{\mu}$ is denoted with $\mathsf{OPT}_{\vec{\mu}}(\vec{k},G)$.

An (α,β)-*approximate* hierarchical partition with respect to $\vec{\mu}$ and $\vec{k}$ is a $(\vec{k},\beta)$-balanced hierarchical partition whose cost is at most $\alpha \cdot \mathsf{OPT}_{\vec{\mu}}(\vec{k},G)$. An (α,β)-*approximation algorithm* for Hierarchical Graph Partitioning finds an (α,β)-approximate hierarchical partition for any given input graph and cost vector. This also gives an (α,β)-approximation for scheduling on regular tree topologies (see [7]).

2 Min-Max K-Partitioning

In this section, we present an approximation algorithm for Min-Max k-Partitioning. Recall that this problem asks to compute a $(k,1)$-balanced partition $\mathcal{P}$ of a given graph that

minimizes $\text{cost}^{\max}(\mathcal{P})$. We first consider instances, where the input graph is a tree $T = (V, E)$. A suitable approximation of the graph by a tree (see [3], [8], [13]) allows to extend the results to arbitrary graphs with a small loss in the approximation factor. Our algorithm is a decision procedure, i.e., it constructs for a given bound b a solution whose cost is at most $(1+\varepsilon)b$ or proves that $b < \mathsf{OPT}^{\max}(k, G)$. A $(1+\varepsilon, 1+\varepsilon)$-approximation algorithm follows by using binary search. In order to simplify the presentation, we assume throughout this chapter that all vertex weights are 1 and that k divides n. We further assume that the approximation constant ε is at most 1.

The algorithm heavily relies on the construction of a *decomposition of* T, which is defined as follows. A decomposition of T w.r.t. k and b is a partition $\mathcal{D} = \{D_i\}_i$, whose parts D_i are connected components, have size $w(D_i) \leq n/k$ and boundary cost $c(D_i) \leq b$. With each decomposition $\mathcal{D}$ we associate a corresponding *vector set* $\mathcal{I}_\mathcal{D}$. This set $\mathcal{I}_\mathcal{D}$ contains a vector $(c(D_i)/b, w(D_i)k/n)$ that encodes the size and boundary cost of D_i in a normalized way. By this $\mathcal{I}_\mathcal{D}$ contains (most of) the information about $\mathcal{D}$ and our algorithm can just work with $\mathcal{I}_\mathcal{D}$ instead of the full decomposition $\mathcal{D}$. Note that every vector in $\mathcal{I}_\mathcal{D}$ is bounded by $(1, 1)$.

We call a partition of $\mathcal{I}_\mathcal{D}$ into k subsets $\{I_i\}_i$, s.t. the vectors in each subset sum to at most α in both dimensions, an *α-packing* of $\mathcal{I}_\mathcal{D}$. The decomposition $\mathcal{D}$ is called *α-feasible* if an α-packing of $\mathcal{I}_\mathcal{D}$ exists. Note that this implies $\alpha \geq 1$. Also note that this definition of feasibility may be nonstandard, as feasibility incorporates the decomposition cost, i.e., a 1-packing has cost b (the bound that we guessed for the optimal solution).

Determining whether there is an α-packing of a given vector set is an instance of the NP-hard Vector Scheduling problem. Chekuri and Khanna [4] gave a polynomial-time approximation scheme (PTAS) for the Vector Scheduling problem which leads to the following result.

▶ **Lemma 3.** *Let $\mathcal{D}$ be an α-feasible decomposition of T w.r.t. k and b, then there is a polynomial-time algorithm to construct a $(k, (1+\varepsilon)\alpha)$-balanced partition of T of cost at most $(1+\varepsilon)\alpha \cdot b$, for any $\varepsilon > 0$.*

Proof. The proof has been deferred to the full version. ◀

An optimal (but slow) algorithm for Min-Max k-Partitioning could iterate over all possible decompositions of the graph and check if any of these decompositions is 1-feasible. If a 1-feasible decomposition is found, the corresponding 1-packing is computed, otherwise the bound b is rejected.

We modify this approach to obtain a polynomial-time approximation algorithm. As the number of decompositions is exponential, we partition them into a polynomial number of *classes*. This classification is such that decompositions of the same class are similar w.r.t. feasibility. More precisely, we show that if a decomposition is α-feasible, then all decompositions of the same class are $(1+\varepsilon)\alpha$-feasible.

We present a dynamic program that, for every class, checks whether a decomposition in that class exists and that computes some representative decomposition that is in this class. Then, we check the feasibility of every representative. As an exact check is NP-hard, we use the approximate Vector Scheduling as described above. If a 1-feasible decomposition exists, then a representative of the same class has been computed and furthermore, this representative is $(1+\varepsilon)$-feasible. Consequently, we obtain a $(1+\varepsilon)^2$-packing of the representative, which induces a $(k, (1+\varepsilon)^2)$-balanced partition of T with cost at most $(1+\varepsilon)^2 b$.

2.1 Classification of Decompositions

We now describe the classification of decompositions using the vector set $\mathcal{I}_\mathcal{D}$. A vector $\vec{p} \in \mathcal{I}_\mathcal{D}$ is *small*, if both its coordinates are at most ε^3 , otherwise it is *large*. Small and large vectors are henceforth treated separately.

We define a *type of a large vector* $\vec{p} = (p_1, p_2)$ as follows. Informally, the type of a large vector classifies the value of both coordinates. Let $i = 0$ if $p_1 < \varepsilon^4$, otherwise let i be the integer such that p_1 lies in the interval $[(1+\varepsilon)^{i-1}\varepsilon^4, (1+\varepsilon)^i\varepsilon^4)$. Let $j = 0$ if $p_2 < \varepsilon^4$, otherwise let j be the integer such that p_2 lies in $[(1+\varepsilon)^{j-1}\varepsilon^4, (1+\varepsilon)^j\varepsilon^4)$. Note that i and j can each take $t = \lceil \log_{1+\varepsilon}(1/\varepsilon^4) \rceil + 1$ different values because $\vec{p}$ is upper-bounded by $(1,1)$. This is a consequence of normalizing the vectors in $\mathcal{I}_\mathcal{D}$ as mentioned earlier. The pair (i,j) is the *type of* $\vec{p}$. We number types from $1, \ldots, t^2$ arbitrarily.

We next define the *type of a small vector* $\vec{q} = (q_1, q_2)$. Informally, the type of a small vector approximates the *ratio* q_1/q_2 of its coordinates. Let $s = \lceil \log_{1+\varepsilon}(1/\varepsilon) \rceil$ and subdivide $[\varepsilon, 1/\varepsilon]$ into $2s$ intervals of the form $[(1+\varepsilon)^i, (1+\varepsilon)^{i+1})$ for $i = -s, \ldots, s-1$. Crop the first and the last interval at ε and $1/\varepsilon$ respectively. Add two outer intervals $[0, \varepsilon)$ and $[1/\varepsilon, \infty)$ to obtain a discretization of the positive reals into $2s+2$ types $-(s+1), \ldots, s$. The vector $\vec{q} = (q_1, q_2)$ has type i, if its ratio q_1/q_2 falls into the i-th interval.

Using these terms, we define the size signature and ratio signature of a vector set $\mathcal{I}_\mathcal{D}$. The size signature stores the number of large vectors of every type while the ratio signature stores the *sum* of small vectors of every type. Let $\mathcal{I}_\mathcal{D}^{\text{large}}$ denote the subset of large vectors and $\mathcal{I}_{\mathcal{D},j}^{\text{small}}$ denote the subset of small vectors of type j.

▶ **Definition 4** (Signature). Let $\mathcal{D}$ be a set of disjoint connected components. The vector $\vec{\ell} = (\ell_1, \ldots, \ell_{t^2})$ is the *size signature* of $\mathcal{I}_\mathcal{D}$, if $\mathcal{I}_\mathcal{D}^{\text{large}}$ contains exactly ℓ_i vectors of type i for $i = 1, \ldots, t^2$. The vector $\vec{h} = (\vec{h}_{-(s+1)}, \ldots, \vec{h}_s)$ is the *ratio signature* of $\mathcal{I}_\mathcal{D}$ if $\vec{h}_j = \sum_{\vec{p} \in \mathcal{I}_{\mathcal{D},j}^{\text{small}}} \vec{p}$ for $j = -(s+1), \ldots, s$. We call $\vec{g} = (\vec{\ell}, \vec{h})$ the signature of $\mathcal{D}$ and $\mathcal{I}_\mathcal{D}$.

We prove in the following that decompositions with the same signature have nearly the same feasibility properties. We start with a technical claim whose proof is omitted here.

▶ **Claim 5.** *Let* $0 < \varepsilon \leq 1$ *and* $s = \lceil \log_{1+\varepsilon}(1/\varepsilon) \rceil$. *Then* $s \cdot \varepsilon^3 \leq 2\varepsilon$.

Proof. The proof has been deferred to the full version. ◀

▶ **Lemma 6.** *Assume that* $\mathcal{D}$ *is an* α*-feasible decomposition of signature* $\vec{g}$ *and that* $\mathcal{D}'$ *has the same signature. Then* $\mathcal{D}'$ *is* $(1+9\varepsilon)\alpha$*-feasible.*

Proof. Given an α-packing $\{I_i\}_i$ of $\mathcal{I}_\mathcal{D}$, we describe a $(1+9\varepsilon)\alpha$-packing $\{I'_i\}_i$ of $\mathcal{I}_{\mathcal{D}'}$. For every vector $\vec{p} \in \mathcal{I}_{\mathcal{D}'}$, we have to select a set I'_i to add this vector to. We only argue about the first coordinate of the resulting packing, the reasoning for the second coordinate is analogous.

We start with the large vectors. We choose an arbitrary bijection $\pi : \mathcal{I}_{\mathcal{D}'}^{\text{large}} \to \mathcal{I}_\mathcal{D}^{\text{large}}$ such that only vectors of the same type are matched. This can be done easily, since $\mathcal{I}_\mathcal{D}$ and $\mathcal{I}_{\mathcal{D}'}$ have the same size signature. Now, we add a vector $\vec{p} \in \mathcal{I}_{\mathcal{D}'}^{\text{large}}$ to set I'_i if and only if $\pi(\vec{p}) \in I_i$.

As vector $\vec{p} = (p_1, p_2)$ and $\pi(\vec{p})$ share the same signature, their sizes are similar in both coordinates. More precisely if the first coordinate of $\vec{p}$ is larger than ε^4, then the first coordinate of $\pi(\vec{p})$ is at most $(1+\varepsilon)p_1$. If p_1 is smaller than ε^4, then p_2 must be larger than ε^3 because $\vec{p}$ is a large vector. Consequently there can be at most α/ε^3 large vectors with such a small first coordinate in the same part of α-packing $\{I_i\}_i$. Their sum is bounded by

$\alpha\varepsilon$ in the first coordinate. Let $(L_{i,\text{large}}, R_{i,\text{large}})$ denote the sum of large vectors in I_i and let $(L'_{i,\text{large}}, R'_{i,\text{large}})$ denote the sum of large vectors in I'_i. We have

$$L'_{i,\text{large}} \leq (1+\varepsilon)L_{i,\text{large}} + \alpha\varepsilon \ . \tag{1}$$

We now consider small vectors; recall that they are classified by their ratio. Let $(L_{i,j}, R_{i,j})$ denote the sum of small vectors of type j in I_i. We call types $j \geq 0$ *left-heavy* as they fulfill $L_{i,j} \geq R_{i,j}$, and accordingly we call types $j < 0$ *right-heavy* as they fulfill $L_{i,j} < R_{i,j}$.

For left-heavy types, add vectors of $\mathcal{I}^{\text{small}}_{\mathcal{D}',j}$ to I'_i until the sum $(L'_{i,j}, R'_{i,j})$ of these vectors exceeds $L_{i,j}$ in the first coordinate. It follows that $L'_{i,j} \leq L_{i,j} + \varepsilon^3$, as the excess is at most one small vector. Summing over all left-heavy types gives

$$\sum_{j=0}^{s} L'_{i,j} \leq (s+1)\varepsilon^3 + \sum_{j=0}^{s} L_{i,j} \leq 3\varepsilon\alpha + \sum_{j=0}^{s} L_{i,j} \ , \tag{2}$$

where we used $\alpha \geq 1$ and Claim 5 in the last step.

For right-heavy types, add vectors of $\mathcal{I}^{\text{small}}_{\mathcal{D}',j}$ to I'_i until the sum of these vectors exceeds $R_{i,j}$ in the second coordinate. It follows that $R'_{i,j} \leq R_{i,j} + \varepsilon^3$. We remark that the above procedure distributes all small vectors of any type j. This follows because on the one hand the ratio signature ensures that the sum of vectors of type j is the same in $\mathcal{I}_{\mathcal{D}}$ and $\mathcal{I}_{\mathcal{D}'}$, and on the other hand our Greedy distribution assigns at least as much mass in the heavier coordinate as in the solution for $\{I_i\}_i$. It remains to show an upper bound on $L'_{i,j}$ for right-heavy types.

First consider Type $-(s+1)$, which corresponds to interval $[0, \varepsilon)$. Combining the upper bound on $R'_{i,j}$ for right-heavy types and the fact that $L'_{i,-(s+1)}/R'_{i,-(s+1)} < \varepsilon$, it follows that $L'_{i,-(s+1)} \leq \varepsilon(R_{i,-(s+1)} + \varepsilon^3)$. As $\varepsilon^3 \leq \alpha$ and $R_{i,-(s+1)} \leq \alpha$, this implies $L'_{i,-(s+1)} \leq 2\varepsilon\alpha$.

The remaining types correspond to an interval $[(1+\varepsilon)^j, (1+\varepsilon)^{j+1})$ and both $L_{i,j}/R_{i,j}$ and $L'_{i,j}/R'_{i,j}$ must lie in that interval. We derive a bound on $L'_{i,j}$ as follows:

$$\begin{aligned} L'_{i,j} &\leq (1+\varepsilon)^{j+1} R'_{i,j} \leq (1+\varepsilon)^{j+1}(R_{i,j} + \varepsilon^3) \leq (1+\varepsilon)^{j+1}\left(\frac{L_{i,j}}{(1+\varepsilon)^j} + \varepsilon^3\right) \\ &= (1+\varepsilon)L_{i,j} + (1+\varepsilon)^{j+1}\varepsilon^3 \leq (1+\varepsilon)L_{i,j} + \varepsilon^3 \ . \end{aligned}$$

The first step uses the fact that $L'_{i,j}/R'_{i,j} < (1+\varepsilon)^{j+1}$. The second step uses the upper bound on $R'_{i,j}$ for right-heavy types and the third step uses the bound $L_{i,j}/R_{i,j} \geq (1+\varepsilon)^j$. As $j < 0$ for right-heavy types and $\varepsilon > 0$, the last step follows. Summing up the bounds on all right-heavy types and using Claim 5 gives

$$\sum_{j=-(s+1)}^{-1} L'_{i,j} \leq s\varepsilon^3 + 2\varepsilon\alpha + (1+\varepsilon) \sum_{j=-(s+1)}^{-1} L_{i,j} \leq 4\varepsilon\alpha + (1+\varepsilon) \sum_{j=-(s+1)}^{-1} L_{i,j} \ . \tag{3}$$

We now combine all bounds derived for part I'_i. Let (L_i, R_i) be the sum of vectors in I_i and let (L'_i, R'_i) denote the sum of vectors in I'_i. Clearly $L'_i = L'_{i,\text{large}} + \sum_{j=-(s+1)}^{s} L'_{i,j}$ and a respective equality holds for L_i. The first term $L'_{i,\text{large}}$ is upper-bounded in Equation (1). The sum $\sum_{j=0}^{s} L'_{i,j}$ is upper-bounded in Equation (2), and the sum $\sum_{j=-(s+1)}^{-1} L'_{i,j}$ is

upper-bounded in Equation (3). Combining these bounds gives

$$\begin{aligned} L'_i &= L'_{i,\text{large}} + \sum_{j=-(s+1)}^{-1} L'_{i,j} + \sum_{j=0}^{s} L'_{i,j} \\ &\leq (1+\varepsilon) L_{i,\text{large}} + 8\varepsilon\alpha + \sum_{j=0}^{s} L_{i,j} + (1+\varepsilon) \sum_{j=-(s+1)}^{-1} L_{i,j} \\ &= (1+\varepsilon) L_i + 8\varepsilon\alpha \leq (1+9\varepsilon)\alpha \ . \end{aligned}$$

The last step follows from the assumption that $\{I_i\}_i$ is an α-packing. ◀

2.2 Finding Decompositions of the Tree

We now present a dynamic program that computes a set of decompositions $\mathbb{D}$ with the following property: If there exists a decomposition of T with signature $\vec{g}$, then $\mathbb{D}$ contains a decomposition with that signature. The dynamic program adapts a procedure given by Feldmann and Foschini [6]. We first introduce some terminology.

Fix an arbitrary root r of the tree and some left-right ordering among the children of each internal vertex. This defines the *leftmost* and *rightmost* sibling among the children. For each vertex v, let L_v be the set of vertices that are contained in the subtree rooted at v or in a subtree rooted at some left sibling of v. The dynamic program iteratively constructs sets of connected components of a special form, defined as follows.

A *lower frontier* $\mathcal{F}$ is a set of disjoint connected components with nodes, such that vertices that are not covered by $\mathcal{F}$ form a connected component including the root. In addition we require that components of a lower frontier have at most size n/k and at most cost b. We define the *cost* of a lower frontier $\mathcal{F}$ as the total cost of edges with exactly one endpoint covered by $\mathcal{F}$. Note that this may be counter-intuitive, because the cost of a lower frontier does not consider the boundary cost of the individual connected components.

The algorithm determines for every vertex v, signature $\vec{g}$, cost $\kappa \leq b$ and number of vertices $m \leq n$, if there exists a lower frontier of L_v with signature $\vec{g}$ that covers m vertices with cost κ. It stores this information in a table $D_v(\vec{g}, m, \kappa)$ and computes the entries recursively using a dynamic program. Along with each positive entry, the table stores the corresponding lower frontier. In the following paragraphs, we describe the dynamic program in more detail.

First, consider the case where v is a leaf and the leftmost among its siblings. Let v_p be the parent of v. As $L_v = \{v\}$, the lower frontier of L_v is either empty or a single connected component $\{v\}$. Therefore $D_v((\vec{0}, \vec{0}), 0, 0) = 1$ and $D_v(\vec{g}(1, c(v, v_p)), 1, c(v, v_p)) = 1$. Here, we use $\vec{g}(|S|, c(S))$ to denote the signature of a set that just contains connected component S (recall that $c(S)$ denotes the cost of S). For all other values, $D_v(\vec{g}, m, \kappa) = 0$.

Second, we consider the case where v is neither a leaf, nor the leftmost among its siblings. The case when v is a leaf but not leftmost sibling and the case when v is an inner vertex that is leftmost sibling and follow from an easy adaption. Let w be the left sibling of v and u its rightmost child. Observe that L_v is the disjoint union of L_u, L_w and $\{v\}$.

If the edge from v to its parent is not cut by a lower frontier $\mathcal{F}$ of L_v, then v is not covered by $\mathcal{F}$. Consequently $\mathcal{F}$ splits into two lower frontiers $\mathcal{F}_u$ and $\mathcal{F}_w$ of L_u and L_w respectively. Denote their signatures with $\vec{g}_u$ and $\vec{g}_w$; they must sum up to $\vec{g}$. If $\mathcal{F}_u$ covers m_u vertices, then $\mathcal{F}_w$ needs to cover $m - m_u$ vertices. Similarly, if $\mathcal{F}_u$ has cost κ_u, then $\mathcal{F}_w$

needs cost $\kappa - \kappa_u$. Hence, if the edge from v to its parent is not cut, $D_v(\vec{g}, m, \kappa)$ is equal to

$$\bigvee_{\substack{m_u \leq m,\ \kappa_u \leq \kappa,\\ \vec{g}_u + \vec{g}_w = \vec{g}}} (D_w(\vec{g}_w, m - m_u, \kappa - \kappa_u) \wedge D_u(\vec{g}_u, m_u, \kappa_u)) \ . \tag{4}$$

If the edge from v to its parent v_p is cut by the lower frontier $\mathcal{F}$ of L_v, then $\mathcal{F}$ covers the entire subtree of v. It follows that $\mathcal{F}$ consists of three disjoint parts: a component S that contains v, a lower frontier $\mathcal{F}_u$ of L_u and a lower frontier $\mathcal{F}_w$ of L_w. Denote the cost of $\mathcal{F}_u$ by κ_u and the cost of $\mathcal{F}_w$ by κ_w. Let T_v denote the subtree of v. The cost $c(S)$ equals $\kappa_u + c(v, v_p)$, because S is connected to v_p in the direction of the root and with the highest vertices of $\mathcal{F}_u$ in the direction of the leaves. The cost κ of the lower frontier $\mathcal{F}$ is $\kappa_w + c(v, v_p)$, because all edges leaving $\mathcal{F}_u$ have their endpoints in $\mathcal{F}$. The signatures of the three parts of $\mathcal{F}$ must sum up to $\vec{g}$, therefore $\vec{g} = \vec{g}_u + \vec{g}_w + \vec{g}(|S|, c(S))$. Hence, if the edge from v to its parent is cut, $D_v(\vec{g}, m, \kappa)$ is equal to

$$\bigvee_{\substack{|S| \leq n/k,\ c(S) \leq b,\\ \vec{g}_u + \vec{g}_w + \vec{g}(|S|, c(S)) = \vec{g}}} \Big(D_w(\vec{g}_w, m - |T_v|, \kappa - c(v, v_p)) \wedge D_u(\vec{g}_u, |T_v| - |S|, c(S) - c(v, v_p)) \Big) \ . \tag{5}$$

We conclude that $D_v(\vec{g}, m, \kappa) = 1$, if Term (4) or Term (5) evaluate to 1.

The running time of the dynamic program depends on the number of signatures that need to be considered at each vertex. The following lemma bounds their number, its proof is omitted due to space constraints.

▶ **Lemma 7.** *At vertex v, the dynamic program considers $|L_v|^{t+4s+4}(2b)^{2s+2}\gamma$ signatures, where $|L_v|$ is the size of L_v and $\gamma = (k/\varepsilon^2)^{t^2}$.*

Proof. The proof has been deferred to the full version. ◀

As the number of signatures is polynomial in n and k, it follows that the above dynamic program only needs a polynomial number of steps.

▶ **Observation 8.** *Let $\mathbb{D}$ be the set of decompositions corresponding to positive entries of $D_r(\vec{g}, n, 0)$ as computed by the dynamic program. Then for any decomposition of T with signature $\vec{g}$, there exists a decomposition in $\mathbb{D}$ with the same signature.*

Combining the dynamic program with the properties of signatures, we obtain an approximation algorithm for Min-Max k-Partitioning on trees.

▶ **Theorem 9.** *There is a polynomial-time $(1+\varepsilon, 1+\varepsilon)$-approximation algorithm for Min-Max k-Partitioning on trees.*

Proof. Follows from Lemmas 6, 7 and Observation 8. The running time is polynomial as both the dynamic program and the Vector Scheduling subroutine run in polynomial time. ◀

We extend our results to arbitrary undirected graphs $G = (V, E, c)$ with the help of a decomposition tree that acts as a cut sparsifier (see Räcke and Shah [13]). Theorem 1 follows by standard arguments that are deferred to the full version.

Algorithm 1 merge$(\mathcal{H}' = (\mathcal{P}'_1, \dots, \mathcal{P}'_h), \vec{k})$

$\mathcal{P}_1 \leftarrow$ Merge two sets of $\mathcal{P}'_1$ with minimum combined weight until $|\mathcal{P}'_1| = k_1$.
for $\ell \leftarrow 2, \dots, h$ **do**
  $d \leftarrow k_\ell / k_{\ell-1}$.
  for all $P_{\ell-1,i} \in \mathcal{P}_{\ell-1}$ **do**
    $Q \leftarrow \{P' \in \mathcal{P}'_\ell \mid P' \cap P_{\ell-1,i} \neq \emptyset\}$. // Subclusters of $P_{\ell-1,i}$
    $P_{\ell,1} \leftarrow \emptyset, \dots, P_{\ell,d} \leftarrow \emptyset$.
    while $Q \neq \emptyset$ **do**
      Assign Q's next element to $P_{\ell,j}$ with smallest weight.
    end while
    $\mathcal{P}_\ell \leftarrow \mathcal{P}_\ell \cup \bigcup_j P_{\ell,j}$.
  end for
end for
Return $\mathcal{H} = (\mathcal{P}_1, \dots, \mathcal{P}_h)$.

3 Hierarchical Partitioning

In this section, we present a bicriteria approximation algorithm for Hierarchical Graph Partitioning. In the Hierarchical Graph Partitioning problem, we are given a graph $G = (V, E)$ and parameters $\vec{k}$ and $\vec{\mu}$. We are asked to compute a $(\vec{k}, 1)$-balanced hierarchical partition $\mathcal{P}$ of G that minimizes $\text{cost}_{\vec{\mu}}(\mathcal{P})$ among all such partitions. We assume furthermore that a $(\vec{k}, 1)$-balanced partition of G exists and therefore $k_{\ell+1}/k_\ell$ is integral for all levels. This assumption is fulfilled in particular when $\vec{k}$ is derived from a regular hierarchical network. Note that this is an extension of Min-Sum k-Partitioning and that $\text{cost}_{\vec{\mu}}$ minimizes the weighted *sum* of all edges cut by all subpartitions. This contrasts with the objective function $\text{cost}^{\max}$ considered in the previous section.

Our algorithm relies on the construction of *graph separators*. Graph separators are partitions in which the maximum weight of a part is bounded, but the number of parts is arbitrary. More precisely, a *σ-separator* of G is a partition $\mathcal{P}$ of G, such that $w(P_i) \leq w(V)/\sigma$ for every i.

For any positive vector $\vec{\sigma} \in \mathbb{R}^h$, a *$\vec{\sigma}$-hierarchical separator* of G is a hierarchical partition of G, where on every level $\mathcal{P}_\ell$ is a σ_ℓ-separator. Note that any (k, ν)-balanced partition is a (k/ν)-separator and any $(\vec{k}, \nu)$-balanced hierarchical partition is a $(\vec{k}/\nu)$-hierarchical separator. An *α-approximate* $\vec{\sigma}$-hierarchical separator w.r.t. $\vec{k}$ and $\vec{\mu}$ is a σ-hierarchical separator whose cost is at most $\alpha \cdot \mathsf{OPT}_{\vec{\mu}}(\vec{k}, G)$.

Approximate hierarchical separators can be transformed into approximate hierarchical partitions using the merging procedure described in Algorithm 1. It is essentially a greedy strategy for bin packing on every level that makes sure that partitions remain refinements of each other. The next lemma states that the approximation error of the merging procedure is linear in the height h.

▶ **Lemma 10.** *Let $\mathcal{H}' = (\mathcal{P}'_1, \dots, \mathcal{P}'_h)$ be an α-approximate $\vec{k}/(1+\varepsilon)$-hierarchical separator w.r.t. $\vec{k}$ and $\vec{\mu}$. Then Algorithm 1 constructs an $(\alpha, (1+\varepsilon)(h+1))$-approximate hierarchical partition $\mathcal{H}$ w.r.t. $\vec{\mu}$ and $\vec{k}$.*

Moreover, if $\mathcal{P}'_1$ contains at most k_1 parts, then $\mathcal{H}$ is an $(\alpha, (1+\varepsilon)h)$-approximate hierarchical partition w.r.t. $\vec{\mu}$ and $\vec{k}$.

Proof. We use induction over h. For $h = 1$, assume without loss of generality that $P_{1,1} \in \mathcal{P}_1$ has maximum weight. If $w(P_{1,1}) > 2(1+\varepsilon)w(V)/k_1$, then some merging must have occurred

in the first line of the algorithm. It follows that all distinct parts $P_{1,i}, P_{1,j} \in \mathcal{P}_1$ have combined weight at least $2(1+\varepsilon)w(V)/k_1$. This is a contradiction to the fact that the total weight of all parts is $w(V)$.

Assume that the claim holds for some $h \geq 1$, i.e., the weight of all parts of $\mathcal{P}_h$ is upper-bounded by $(1+\varepsilon)(h+1)w(V)/k_h$. Use $d = k_{h+1}/k_h$. Assume without loss of generality that $P_{h+1,1}$ receives the last element S from Q when considering $P_{h,i}$ in the fourth line of the algorithm; this implies that the weight of $P_{h+1,1}$ is smallest before receiving S. To derive a contradiction, assume that $w(P_{h+1,1}) > (1+\varepsilon)(h+2)w(V)/k_{h+1}$. It follows that

$$\begin{aligned} w(P_{h,i}) &\geq w(S) + \sum_{j=1}^{d} w(P_{h+1,1} \setminus S) > \sum_{j=1}^{d} (w(P_{h+1,1}) - w(S)) \\ &> d \cdot \Big((1+\varepsilon)(h+2)\frac{w(V)}{k_{h+1}} - (1+\varepsilon)\frac{w(V)}{k_{h+1}} \Big) \end{aligned}$$

The last line equals $(1+\varepsilon)(h+1)\cdot w(V)/k_h$ which is a contradiction to the induction hypothesis. For the first inequality, we used that part $P_{h+1,1}$ had the smallest weight before receiving S. The last step uses the fact that the weight of all $S \in Q$ is at most $(1+\varepsilon)w(V)/k_{h+1}$.

As merging sets does not increase the cost of a hierarchical partition, the approximation factor α on the cost does not change. This finishes the proof of the first statement.

To see the second statement, observe that if $\mathcal{P}'_1$ contains at most k_1 parts, then there is nothing to do in the first line of the algorithm. As no merging occurs, every set in $\mathcal{P}'_1$ has at most weight $(1+\varepsilon)w(V)/k_1$. With a similar induction argument as above, it follows that the merging procedure returns an $(\alpha, (1+\varepsilon)h)$-approximate hierarchical partition. ◀

In the following, we describe an algorithm that computes an $\mathcal{O}(\log n)$-approximate $\vec{k}/(1+\varepsilon)$-hierarchical separator $\mathcal{H}' = (\mathcal{P}'_1, \ldots, \mathcal{P}'_h)$. We use $\tilde{\mu}_\ell$ as a shorthand for $\sum_{j=\ell}^{h} \mu_j$.

Our algorithm first computes partition $\mathcal{P}'_1$ using the Min-Sum k-Partitioning algorithm by Feldmann and Foschini [6]. It can easily be adapted to handle polynomial vertex weights.

The following theorem states the result for the Min-Sum k-Partitioning algorithm.

▶ **Theorem 11** ([6]). *There is a polynomial-time algorithm that computes a $(k_1, 1+\varepsilon)$-balanced partition of G with cut cost at most $\mathcal{O}(\log n) \cdot \mathsf{OPT}^{\mathrm{sum}}(k_1, G)$.*

Note that $\tilde{\mu}_1 \cdot \mathsf{OPT}^{\mathrm{sum}}(k_1, G)$ is an upper bound on the cost of any $(\vec{k}, 1)$-balanced hierarchical partition.

The algorithm to construct $(\mathcal{P}'_2, \ldots, \mathcal{P}'_h)$ is an adaptation of the algorithm by Even et al. for Min-Sum k-Partitioning. Note that the following description is basically the description in [5]. It relies on a distance assignment $\{d(e)\}_{e \in E}$ to the edges of G, the *spreading metric*. The distances are used in a procedure called cut-proc that permits to find within a subgraph of G a subset of vertices T, whose cut cost is bounded by its volume $\mathrm{vol}(T)$. The volume of a set of vertices is approximately the weighted length of the edges in its cut.

Given the procedure cut-proc, the algorithm constructs all the remaining partitions $(\mathcal{P}'_2, \ldots, \mathcal{P}'_h)$ as follows. First recall that $\mathcal{P}'_1$ is already given by Theorem 11. For $\ell \geq 2$, the algorithm constructs $\mathcal{P}'_\ell$ by examining all parts of $\mathcal{P}'_{\ell-1}$ separately. On a given part $P'_{\ell-1} \in \mathcal{P}'_{\ell-1}$, it uses procedure cut-proc to identify a subgraph H of $P'_{\ell-1}$. This subgraph becomes a part of $\mathcal{P}'_\ell$ and cut-proc is restarted on $P_{\ell-1} \setminus H$. This process is repeated until less than $(1+\varepsilon)w(V)/k_\ell$ vertices are left. The remaining vertices also constitute a part of $\mathcal{P}'_\ell$.

It is important to note during the following discussion that the boundary cost $c(S)$ of a set $S \subseteq V$ depends on the subgraph that S was chosen from. Sometimes, we will choose S from a subgraph H of G; in this case, $c(S)$ only counts the cost of edges leaving S towards H.

The spreading metric used is an optimal solution to the following linear program, called spreading metrics LP.

$$\begin{aligned} \min & \sum_{e\in E} c(e)d(e) \\ \text{s.t.} & \sum_{u\in S} \text{dist}_V(v,u)\cdot w(u) \geq \tilde{\mu}_\ell\left(w(S) - \frac{w(V)}{k_\ell}\right), \qquad 1\leq \ell \leq h,\ S\subseteq V,\ v\in S \\ & 0\leq d(e)\leq \tilde{\mu}_1, \qquad e\in E\ . \end{aligned} \tag{6}$$

Using the distances $d(e)$, $\text{dist}_S(v,u)$ denotes the length of the shortest path between vertices v and u in subgraph S. Let τ denote the optimal value of the spreading metrics LP.

The following two lemmas prove the basic properties of solutions of the spreading metrics LP. Their proofs are omitted due to space constraints.

▶ **Lemma 12.** *Any $(\vec{k},1)$-balanced hierarchical partition $\mathcal{H}=(\mathcal{P}_1,\ldots,\mathcal{P}_h)$ induces a feasible solution to the spreading metrics LP of value $\text{cost}_{\vec{\mu}}(\mathcal{H})$.*

Proof. The proof has been deferred to the full version. ◀

The above lemma implies that τ is a lower bound on $\text{cost}_{\vec{\mu}}(\mathcal{H})$. We define the radius of vertex v in some $S\subseteq V$ as $\text{radius}(v,S) := \max\{\text{dist}_S(v,u)\mid u\in S\}$.

The following lemma proves that any set of sufficient weight has at least constant radius.

▶ **Lemma 13.** *If $S\subseteq V$ satisfies $w(S) > (1+\varepsilon)w(V)/k_\ell$ for some level ℓ, then for every vertex $v\in S$, $\text{radius}(v,S) > \frac{\varepsilon}{1+\varepsilon}\tilde{\mu}_\ell$.*

Proof. The proof has been deferred to the full version. ◀

Even though the spreading metrics LP contains an exponential number of constraints, it can be solved in polynomial time using the ellipsoid algorithm and a polynomial-time separation oracle. By rewriting the Constraints (6), we obtain for all levels ℓ, $S\subseteq V$ and $v\in S$ the constraint $\sum_{u\in S}(\text{dist}_V(u,v)-\tilde{\mu}_\ell)w(u)\geq w(V)/k_\ell$.

For any vertex v, the left-hand side of the constraint is minimized when the subset $S_v=\{u\in V\mid \text{dist}_V(v,u)\leq\tilde{\mu}_\ell\}$ is chosen. Consequently, h single-source shortest path computations from every vertex provide a polynomial-time separation oracle for the spreading metrics LP.

The notation introduced next serves the definition of procedure cut-proc in Algorithm 2. A ball $B(v,r)$ in subgraph $G'=(V',E')$ is the set of vertices in V' whose distance to v is strictly less than r, thus $B(v,r) := \{u\in V'\mid \text{dist}_{V'}(u,v)<r\}$. The set $E(v,r)$ denotes the set of edges leaving $B(v,r)$. The volume of $B(v,r)$, denoted by $\text{vol}(v,r)$, is defined by

$$\text{vol}(v,r) := \eta\cdot\tau + \sum_{\substack{(x,y)\in E(v,r),\\ x\in B(v,r)}} c(x,y)(r-\text{dist}_{B(v,r)}(v,x))\ , \tag{7}$$

where $\eta=\frac{1}{hn}$. Note that the volume only considers the edges in the cut, not the edges within the ball, which is different from [5]. The following lemma is a corollary of Even et al. [5].

▶ **Lemma 14.** *Given a subgraph $G'=(V',E')$ that satisfies $w(V')>(1+\varepsilon)w(V)/k_\ell$, and given a spreading metric $\{d(e)\}_e$, procedure cut-proc finds a subset $T\subseteq V'$ that satisfies $w(T)\leq(1+\varepsilon)w(V)/k_\ell$.*

Algorithm 2 cut-proc$(V', E', \{c(e)\}_{e\in E'}, \{d(e)\}_{e\in E'}, \tilde{\mu}_\ell)$

Choose an arbitrary $v \in V'$
$\tilde{r} \leftarrow \frac{\varepsilon}{1+\varepsilon}\tilde{\mu}_\ell$
$T \leftarrow \{v\}$
$v' \leftarrow$ closest vertex to T in $V' \setminus T$
while $c(T) > \frac{1}{\tilde{r}} \cdot \ln(\frac{\mathrm{vol}(v,\tilde{r})}{\mathrm{vol}(v,0)}) \cdot \mathrm{vol}(v, \mathrm{dist}_{V'}(v, v'))$ **do**
  $T \leftarrow T \cup \{v'\}$
  $v' \leftarrow$ closest vertex to T in $V' \setminus T$
end while
Return T

Proof. The proof has been deferred to the full version. ◀

The approximation guarantee on the cost induced by partitions $\mathcal{P}'_1$ to $\mathcal{P}'_h$ is given by the next theorem.

▶ **Theorem 15.** *The sequence $\mathcal{H}' = (\mathcal{P}'_1, \ldots, \mathcal{P}'_h)$ is a $\vec{k}/(1+\varepsilon)$-hierarchical separator with $|\mathcal{P}'_1| \le k_1$ and $\mathrm{cost}_{\vec{\mu}}(\mathcal{H}) \le \alpha \cdot \mathsf{OPT}_{\vec{\mu}}(\vec{k}, G)$, where $\alpha = \mathcal{O}(\log n)$.*

Proof. The fact that $\mathcal{H}'$ is a $\vec{k}/(1+\varepsilon)$-hierarchical separator follows immediately from the construction and Algorithm 2. In particular, Theorem 11 ensures that $|\mathcal{P}'_1| \le k_1$. The cost of $\mathcal{P}'_2$ to $\mathcal{P}'_h$ w.r.t. $\vec{\mu}$ equals $\sum_{\ell=2}^{h} \mu_\ell \mathcal{P}'_\ell$. By Lemma 14, the algorithm constructs sets T_ℓ for $\mathcal{P}'_\ell$ whose cost $c(T_\ell)$ is at most

$$\frac{1+\varepsilon}{\varepsilon\tilde{\mu}_\ell} \cdot \ln\left(\frac{\mathrm{vol}(v_\ell, \varepsilon\tilde{\mu}_\ell/(1+\varepsilon))}{\mathrm{vol}(v_\ell, 0)}\right) \cdot \mathrm{vol}(v_\ell, \mathrm{dist}_{V'}(v_\ell, v'_\ell)) \ , \tag{8}$$

where v_ℓ and v'_ℓ are some vertices in T_ℓ. Recall from the description of the algorithm that the left-hand side of this inequality ignores all edges that have been cut on a higher level or that have been cut previously on the same level. By using the shorthand $\tilde{\mu}_\ell$, we account for cuts on subsequent levels, thus obtaining $\sum_{\ell=2}^{h} \mu_\ell \mathcal{P}_\ell \le \sum_{\ell=2}^{h} \tilde{\mu}_\ell \sum_{T_\ell} c(T_\ell)$.

Observe that $\mathrm{vol}(v, 0) \ge \eta\tau$ and $\mathrm{vol}(v, \varepsilon\tilde{\mu}_\ell/(1+\varepsilon)) \le (1+\eta)\tau$ for all $v \in V$. We apply this to Equation (8) and obtain

$$c(T_\ell) \le \frac{1+\varepsilon}{\varepsilon\tilde{\mu}_\ell} \cdot \ln\left(\frac{1+\eta}{\eta}\right) \cdot \mathrm{vol}(v_\ell, \mathrm{dist}_{V'}(v_\ell, v'_\ell)) \ . \tag{9}$$

It follows that

$$\begin{aligned}
&\sum_{\ell=2}^{h} \sum_{T_\ell} \tilde{\mu}_\ell \cdot c(T_\ell) \le \frac{1+\varepsilon}{\varepsilon} \ln\left(\frac{1+\eta}{\eta}\right) \sum_{\ell=2}^{h} \sum_{T_\ell} \mathrm{vol}(v_\ell, \mathrm{dist}_{V'}(v_\ell, v'_\ell)) \\
&= \frac{1+\varepsilon}{\varepsilon} \ln\left(\frac{1+\eta}{\eta}\right) \sum_{\ell=2}^{h} \sum_{T_\ell} (\eta\tau + \sum_{e \in E(v_\ell, \mathrm{dist}_{V'}(v_\ell, v'_\ell))} c(e)d(e)) \\
&\le \frac{1+\varepsilon}{\varepsilon} \ln\left(\frac{1+\eta}{\eta}\right) (hn\eta\tau + \sum_{\ell=2}^{h} \sum_{T_\ell} \sum_{e} c(e)d(e)) \ .
\end{aligned}$$

We plug in Equation (9) to obtain the first step and use the definition of the volume in Equation (7) for the second step.

We now claim that every edge $e \in E$ appears at most once in the triple sum above. If e is cut by cut-proc for $\mathcal{P}'_\ell$, it is removed from the subgraph for all further iterations of cut-proc

on the same level ℓ. Moreover, edge e is not considered on all remaining levels $\ell' > \ell$, because procedure cut-proc is initiated on these levels with subgraphs of the parts of $\mathcal{P}_\ell$. It follows that the triple sum is upper-bounded by τ.

Using $\eta = \frac{1}{hn}$ and the fact that ε is constant, we conclude that

$$\sum_{\ell=2}^{h} \sum_{T_\ell} \tilde{\mu}_\ell \cdot c(T_\ell) \leq \frac{1+\varepsilon}{\varepsilon} \ln\left(\frac{1+\eta}{\eta}\right)(hn\eta+1)\tau = \mathcal{O}(\log n)\tau \ .$$

Recall that by Theorem 11, the weighted cost of the first partition $\tilde{\mu}_1 \cdot \text{cost}^{\text{sum}}(\mathcal{P}'_1)$ is at most $\mathcal{O}(\log n)$ times the cost of the optimal hierarchical partition $\mathsf{OPT}_{\vec{\mu}}(\vec{k}, G)$. As furthermore $\tau \leq \mathsf{OPT}_{\vec{\mu}}(\vec{k}, G)$, the theorem follows. ◀

In other words, the above theorem states that $\mathcal{H}'$ is an $\mathcal{O}(\log n)$-approximate $\vec{k}/(1+\varepsilon)$-hierarchical separator. Combining Lemma 10 and Theorem 15 gives the following.

▶ **Theorem 16.** *There exists a $(\mathcal{O}(\log n), (1+\varepsilon)h)$ approximation algorithm for Hierarchical Graph Partitioning whose running time is polynomial in h and n.*

References

1. Konstantin Andreev and Harald Räcke. Balanced graph partitioning. *Theory Comput. Syst.*, 39(6):929–939, 2006. doi:10.1007/s00224-006-1350-7.
2. Nikhil Bansal, Uriel Feige, Robert Krauthgamer, Konstantin Makarychev, Viswanath Nagarajan, Joseph (Seffi) Naor, and Roy Schwartz. Min-max graph partitioning and small set expansion. In *Proc. of the 52nd FOCS*, pages 17–26, 2011. doi:10.1109/FOCS.2011.79.
3. Marcin Bienkowski, Miroslaw Korzeniowski, and Harald Räcke. A practical algorithm for constructing oblivious routing schemes. In *Proc. of the 15th SPAA*, pages 24–33, 2003. doi:10.1145/777412.777418.
4. Chandra Chekuri and Sanjeev Khanna. On multidimensional packing problems. *SIAM J. Comput.*, 33(4):837–851, 2004. doi:10.1137/S0097539799356265.
5. Guy Even, Joseph (Seffi) Naor, Satish Rao, and Baruch Schieber. Fast approximate graph partitioning algorithms. *SIAM J. Comput.*, 28(6):2187–2214, 1999. Also in *Proc. 8th SODA*, 1997, pp. 639–648. doi:10.1137/S0097539796308217.
6. Andreas E. Feldmann and Luca Foschini. Balanced partitions of trees and applications. In *Proc. of the 29th STACS*, pages 100–111, 2012. doi:10.4230/LIPIcs.STACS.2012.100.
7. Mohammad Taghi Hajiaghayi, Theodore Johnson, Mohammad Reza Khani, and Barna Saha. Hierarchical graph partitioning. In *Proc. of the 26th SPAA*, pages 51–60, 2014. doi:10.1145/2612669.2612699.
8. Chris Harrelson, Kirsten Hildrum, and Satish Rao. A polynomial-time tree decomposition to minimize congestion. In *Proc. of the 15th SPAA*, pages 34–43, 2003. doi:10.1145/777412.777419.
9. Robert Krauthgamer, Joseph (Seffi) Naor, and Roy Schwartz. Partitioning graphs into balanced components. In *Proc. of the 20th SODA*, pages 942–949, 2009. arXiv:soda:1496770.1496872.
10. Robert Krauthgamer, Joseph (Seffi) Naor, Roy Schwartz, and Kunal Talwar. Non-uniform graph partitioning. In *Proc. of the 25th SODA*, pages 1229–1243, 2014. arXiv:soda:2634165.
11. Konstantin Makarychev and Yuri Makarychev. Nonuniform graph partitioning with unrelated weights. In *Proc. of the 41st ICALP*, pages 812–822, 2014. doi:10.1007/978-3-662-43948-7_67.

12 Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proc. of the 40th STOC*, pages 255–264, 2008.

13 Harald Räcke and Chintan Shah. Improved guarantees for tree cut sparsifiers. In *Proc. of the 22nd ESA*, pages 774–785, 2014. doi:10.1007/978-3-662-44777-2_64.