Hierarchical Graph Partitioning

Mohammadtaghi Hajiaghayi *
University of Maryland
College Park, MD
hajiagha@cs.umd.edu

Theodore Johnson
AT&T Research Laboratory
Florham Park, NJ
johnsont@research.att.com

Mohammad Reza Khani *
University of Maryland
College Park, MD
khani@cs.umd.edu

Barna Saha
AT&T Research Laboratory
Florham Park, NJ
barna@research.att.com

ABSTRACT

One of the important optimization questions in highly parallel systems is the problem of assigning computational resources to communicating tasks. While scheduling tasks/operators, tasks assigned to nearby resources (e.g. on the same CPU core) have low communication costs, whereas tasks assigned to distant resources (e.g. on different server racks) have high communication costs. An optimal solution of task to resource assignment minimizes the communication cost of the task ensemble while satisfying the load balancing requirements. We model such an optimization question of minimizing communication cost as a new class of graph partitioning problems called *hierarchical graph partitioning*.

In hierarchical graph partitioning we are given a graph $G = (V, E)$, vertices representing the tasks and edges representing the communication among the vertices. We are also given vertex demands $d : V(G) \to \mathbb{R}^+$ denoting the processing load of each task and edge weights $w : E(G) \to \mathbb{R}^+$ denoting the amount of communication and our goal is to decompose G into k parts/servers of nearly equal weight (for load balancing) and minimize the total cost of the edges being cut (communication cost). However, unlike traditional k-balanced graph partitioning where the cost of an edge cut is independent of the parts containing the two respective end vertices, here the cost varies with the distance of the servers corresponding to the two parts. Since, the servers are generally arranged in a hierarchy, distance is given by a tree metric. In this paper, we initiate the study of hierarchical graph partitioning problem and give efficient algorithms with approximation guarantee. Hierarchical graph partitioning is a significant generalization of graph partitioning problem and faithfully captures several practical scenarios that have served as major motivating applications for graph partitioning.

*supported in part by NSF CAREER award 1053605, ONR YIP award N000141110662, DARPA/AFRL award FA8650-11-1-7162.

SPAA'14, June 23–25, 2014, Prague, Czech Republic.

http://dx.doi.org/10.1145/2612669.2612699.

Categories and Subject Descriptors

G.2.2 [**Mathematics of Computing**]: Graph Theory—*Graph algorithms*

General Terms

Theory; Algorithms

Keywords

Approximation algorithms; Graph Partitioning

1. INTRODUCTION

The research in this paper is motivated by practical problems encountered in optimizing a parallelized data stream processing system such as TidalRace at AT&T [12]. Such a system consists of a collection of streaming inputs, a collection of streaming outputs, and a DAG of processing tasks between the inputs and outputs. A modern commodity server will have 64 or more cores available for executing these tasks, often configured with four CPU sockets, eight cores per CPU socket, and two virtual cores per actual core through the use of hyperthreading. A modern parallelized operating system such as Linux will schedule a ready task on any available CPU core. Although a modern operating system will exhibit some locality preferences in its scheduling decisions, we found that we could significantly improve performance (i.e., maximum throughput) by *pinning* tasks to particular cores, and thus ensure that tasks execute on cores with warm memory caches. We further observed that we could further improve performance by pinning tasks with high communication volumes on *nearby* cores.

Tasks which are pinned to the same core (or pair of hyperthreaded cores) have a low communication cost because they share L1 through L3 caches. Ideally, we would pin all strongly communicating tasks to the same core. However, we face a scheduling constraint: a task has a CPU demand, which we can describe as a fraction of the CPU core that it will use. If we try to oversubscribe a CPU core's resources, the tasks pinned to that core will not be able to keep up with their workload and will drop data in an uncontrolled fashion.

Tasks on more "distant" cores have higher communication costs. For example, task pinned to cores on the same CPU socket will share the same L3 cache, while tasks pinned to cores on different sockets share only the same memory backplane. Our goal is therefore to pin tasks to cores in a way that does not oversubscribe any core but which minimizes overall communication costs.

The stream optimization problem can also be seen in the context of task placement in distributed steaming systems such as IBM's Infosphere Stream of Facebook's Storm [11].

We study the hierarchical graph partitioning problem which models the optimization problem of minimizing communication cost while placing communicating tasks in such distributed environment. Hierarchical graph partitioning is a generalization of the well-studied k-Balanced Graph Partitioning (k-BGP) problem. In the k-BGP, the given graph is broken into k parts with nearly equal number of vertices such that the edges across the parts are minimized. When $k = 2$, one obtains the famous *Minimum Bisection problem*; this NP Hard problem has played an instrumental role in the study and development of approximation algorithms [3, 5, 9, 19, 24]. However, in these problems, one assumes a uniform metric, that is the cost of cutting an edge is same independent of which parts the two end points belong to. This model is not realistic in distributed streaming system where natural hierarchy is observed among the resources.

The hierarchical graph partitioning (HGP) problem is defined as follows. We are given a hierarchy tree (H) with k leaves that correspond to CPU cores with same capacity to which we assign tasks. Each non-leaf node represents a group. For example, in task scheduling, a set of leaves with common parent corresponds to a set of cores having the same CPU socket. Apart from H, we are given a graph $G = (V, E)$, $|V| = n$, where each node $v \in V$ has a demand $d(v)$ and each edge $e = (u, v)$ in E has weight $w(e)$. Our goal is to assign each node of G to a leaf of H such that the following holds.

Let us denote the root of H as r_H and define the *level* of a node to be the number of edges in the unique path between the node and r_H. If a node is at level j in H we refer to it as a Level-(j) node. We assume that H is regular at each level meaning that every Level-(j) node has exactly $\text{DEG}^{(j)}$ children and the height of H is h. Such regularity assumption captures the most common modern scenarios.

Each level j in H is assigned with cost multiplier $cm(j)$ where $cm(i) \geq cm(i+1)$ for all $i \in \{0, \ldots, h-1\}$, For two leaves a_H and b_H of H, let $\text{LCA}_H(a_H, b_H)$ be the level of the lowest common ancestor of a_H and b_H. A solution to the HGP problem is a function $p : V(G) \to \text{LEAVES}(H)$ which assigns each node of G to a leaf in H such that

$$\forall l \in \text{LEAVES}(H), \quad \sum_{v:p(v)=l} d(v) \leq 1,$$

which minimizes cost $c(p)$ where

$$c(p) = \sum_{u,v \in V(G), u \neq v} cm\left(\text{LCA}_H(p(u), p(v))\right) \cdot w((u,v)). \quad (1)$$

Here we assume that the capacity of leaves of H is normalized to one. Note that the k-BGP problem is the special case of HGP where H has height 1 in which $cm(0) = 1$ and $cm(1) = 0$ and each node of G has demand $\frac{k}{n}$.

1.1 Related Works

The *minimum bisection* (MBS) is a central problem in design and analysis of approximation algorithms. An (α, β)-bicriteria approximation algorithm for this problem gives a solution whose cut size is at most α times the size of an optimal solution but violates the size constraint of each section by at most factor β. The $(O(\log n), 1.5)$ bi-criteria approximation by Leighton and Rao [19], as well as the improvement to $(O(\sqrt{\log n}), 1.5)$ by Arora et al. [3], are seminal papers. Recent work by Räcke [24] obtains an elegant $O(\log n)$ (true) approximation for this problem improving upon the previous bound of $O(\log^{1.5} n)$ by Feige and Krauthgamer [9]. Finally, different inapproximability results shown in the sequence of papers [5, 14, 16] relate MBS to the unique games conjecture and to refuting random 3SAT formulas, and have transformed the area.

The k-BGP appears as a natural generalization of MBS and has also been an active area of study. No true approximation that does not violate the "balanced" constraint is possible unless P=NP [1]. Leighton et al. [18] and Simon and Teng [25] achieved an $(O(\log k \log n), 2)$-bicriteria approximation. Later works improved the first criterion of the approximation to $O(\log n)$ [8] and finally to $O(\sqrt{\log k \log n})$ [17]. However, all these came at a price of factor 2 violation in the balanced constraint. The first improvement in that regard was achieved by Andreev and Räcke [1] who showed a $O(\frac{1}{\epsilon^2} \log^{1.5} n, (1+\epsilon))$ approximation bound that was finally improved to $O(\log n, (1+\epsilon))$ by Feldmann and Foschini [10].

Metric embedding problems are another class of closely related problems to HGP. The metric labeling problem is a generalization of graph partitioning problem where in addition there is a cost for labeling vertices to parts. Starting from the seminal paper of Kleinberg and Tardos [15], there has been a long line of works considering several variants of the problem. Naor and Schwartz [21] introduced balanced metric labeling in which each label has a capacity l, and additionally the semimetric on the labels is uniform. Naor and Schwartz [21] gave an $(O(\log n), O(\min\{l, \log k\}))$- bicriteria approximation algorithm for the problem. Capacitated metric labeling is a generalization of balanced metric labeling where labels can have arbitrary capacity and the metric may not be uniform. When the number of labels is fixed Andrews et al. [2] gave a $O(\log n)$ approximation algorithm for capacitated metric labeling.

The HGP problem is also expensively studied from the non-theory perspective and different heuristics have been proposed. Walshaw and Cross [29] develop algorithms for what they call the mapping problem, which is similar to the HGP problem, except it allows mapping an arbitrary task graph G onto an arbitrary architecture graph. The cost function used is identical. Another approach called "dual recursive bipartitioning" to the mapping problem was developed in Pellegrini [22]. The technique uses recursive bisection of both the task graph and the processor graph while minimizing a similar cost function. More recently, Moulitsas and Karypis [20] work on a class of algorithms called "Architecture-Aware Partitioning Algorithms", in which the task graph is mapped onto nodes of an arbitrary architecture graph. Their algorithm starts with an initial partition and iteratively refine the partition in order to minimize objective functions such as communication time or total time. A hierarchical architecture graph for mapping tasks has also been studied in several other works [4, 26, 27]. There are a number of commonly-used graph partitioning softwares used for HGP type problems such as Zoltan[7], SCOTCH [23], JOSTLE [28], Para-PART [6], etc.

1.2 Results and Techniques

In this section we precisely introduce our result and give a summary of our techniques. There is no true approximation algorithm which does not violate the capacity constraint of the leaves of H for HGP as such bounds are not realizable even for k-BGP, which is the special case of HGP when $h = 1$, unless P=NP [1]. Therefore we need bicriteria approximations.

Definition 1 *An (α, β)-bicriteria approximation algorithm for the HGP problem gives a solution which violates the capacity of the*

leaves of H by at most factor β and has cost at most α times the cost of an optimal solution which has no capacity violation.

Our main result is the following.

Theorem 1 *For any positive real value ϵ there is an $(O(\log n), (1+\epsilon)\cdot(1+h))$-bicriteria approximation algorithm for HGP when height h is constant.*

The factor $(1+\epsilon)$ blow out in the capacity constraint is the standard consequence of rounding the demands of the nodes of G which is similar to the Knapsack problem. As one may notice height of tree H is the source of difficulty in our algorithm as we require it to be constant and there is $(1+h)$ factor in the second criterion. Remember that MBS and k-BGP are the special cases of HGP when height h is equal to one. In the following we argue that the usual techniques used in MBS and k-BGP cannot be generalized for HGP even when h is equal to two. In fact we show that when h is equal to two the states of the dynamic programs used in the two problems can be exponential and hence we expect increasing height h makes HGP significantly harder. At the best of our knowledge this is the first paper which deals with such a multiple hierarchy partitioning problem in a theoretical setting. Finally we note that the $O(\log n)$ factor loss in the cost, the first criterion, is the standard consequence of embedding G into a distribution of trees.

The general framework for solving packing problems such as MBS and k-BGP has two steps. The first step is to embed the graph into a distribution of trees using Räcke's hierarchal decomposition for congestion minimization [24]. The embedding guarantees that the cut size of each subset of nodes in the graph is closely approximated with the expected cut size of the corresponding nodes in the distribution of trees. Using this property it can be proved that solving the packing problem for trees suffices to solve the problem for general graphs but with an extra $O(\log n)$ factor blow up in the approximation factor. The second step is to solve the packing problem for trees using a dynamic program. The main idea of the dynamic program is to solve the same problem for each sub-tree of the tree and combine them to get the final result. For example in the k-BGP problem it is only important to keep track of how much each partition is filled when solving the problem for each sub-tree. Therefore, two sub-problems can be combined if no partition overflows using the state of the partitions in each sub-problem. If the capacity of each partition is $\frac{n}{k}$ then there are at most $\frac{n}{k}$ different states for each partition. By allowing a rounding error of ϵ we can assume that there are at most $\frac{1}{\epsilon}$ different states for each partition. Therefore, if for each state s we keep how many partitions are in state s, the overall states of all the partitions can be kept in $k^{1/\epsilon}$.

Our algorithm also has the same two steps. In the first step we extend the paradigm of [2, 24] to round the graph into a probability distribution of decomposition trees for the case when we have multiple levels. The main genuine idea of this paper is dealing with the second step. The standard techniques described above for the second step is not applicable to HGP even when there are only two levels in the hierarchy tree as the number of different states can be exponential in n. Our main idea here is to relax the problem in a way which causes the violation in the capacity constraints but makes the problem tractable. We explain this idea more carefully in the following.

First we describe the standard framework which is usually used for the bin packing type problems first introduced in [13]. We show that although this framework can be applied well to the k-BGP problem, it does not work for the HGP problem. Consider the problem in which we want to pack n items $a_1, a_2, \ldots, a_n$ each has demand one into m bins all with the same size B. The idea of the framework of Hochbaum and Shmoys [13] is to maintain the state of the bins after packing the first n' items by a B-dimensional vector $\boldsymbol{\alpha}$ whose ith entry specifies how many bins are filled with exactly i items. Hence, the problem can be solved efficiently by a dynamic programming over subproblems $bp(\boldsymbol{\alpha}, n')$ which returns true if the first n' items can be packed into the bins such that $\sum_i \boldsymbol{\alpha}(i)$ is equal to m. Because there are exponentially many different values (m^B) for $\boldsymbol{\alpha}$, instead of solving for the exact portion each bin is packed; for each bin they maintain if it is filled in the range $[(1+\epsilon)^{j-1}\cdot\epsilon\cdot B, (1+\epsilon)^j\cdot\epsilon\cdot B)$ for $j \in O(\log_{1+\epsilon}\frac{1}{\epsilon})$ where ϵ is a fixed constant. This way there are only $O(\log_{1+\epsilon}\frac{1}{\epsilon})$ different states for each bin which is a constant number. Therefore the total number of different values for $\boldsymbol{\alpha}$ is $m^{(\log_{1+\epsilon}\frac{1}{\epsilon})}$ and hence polynomial in the size of the input but they violate the capacities by factor $(1+\epsilon)$.

We can apply this framework to the HGP problem and maintain the states of the leaves of H. However, even for the case when height of H is two there is a major issue. Suppose H has height 2 and each Level-(1) H-node has $\text{DEG}^{(1)}$ children which are leaves of H, all having the same capacity. In order to maintain the state of each leaf we need to maintain how much it is filled and moreover to which Level-(1) H-node it belongs.

In order to avoid the above subtlety we define *mirror functions*. Let $\text{SUB}(a_H)$ be the subtree induced by node a_H in tree H. For a solution $p : V(G) \to \text{LEAVES}(H)$ we define its *mirror function* $\mathcal{P} : V(H) \to 2^{V(G)}$ where for each H-node $a_H \in V(H)$

$$\mathcal{P}(a_H) = \{v \in V(G) | p(v) \in \text{SUB}(a_H)\}. \quad (2)$$

We show that cost of solution p can be rewritten using mirror function $\mathcal{P}$ as

$$c(\mathcal{P}) = \sum_{j=1}^{h} \sum_{a_H^{(j)} \in V(H)} w(\text{CUT}(\mathcal{P}(a_H^{(j)}))) \cdot \frac{cm(j-1) - cm(j)}{2}. \quad (3)$$

where $a_H^{(j)}$ is a Level-(j) H-node and $\text{CUT}(\mathcal{P}(a_H^{(j)}))$ is the minimum cut separating $\mathcal{P}(a_H^{(j)})$ form the rest of the leaves. This way, cost of p depends only on $\text{CUT}(\mathcal{P}(a_H))$ for $a_H \in V(H)$.

Suppose we want to solve HGP for rooted tree T. We build a partial mirror function $\mathcal{P}_v$ for each node $v \in V(T)$ where $\mathcal{P}_v$ is the same function as $\mathcal{P}$ except it maps H-nodes to the leaves only at $\text{SUB}(v)$. Building a mirror function is also hard. Lets fix a Level-(j) H-node $a_H^{(j)}$ for which we want to find $\mathcal{P}_v(a_H^{(j)})$. In fact not only we have to find $\mathcal{P}_v(a_H^{(j)})$ we need to partition this set into at most $\text{DEG}^{(j)}$ subsets and assign them to the children of $a_H^{(j)}$. First we relax the condition that $\mathcal{P}_v(a_H^{(j)})$ has to be partitioned into at most $\text{DEG}^{(j)}$ subsets by allowing it to be partitioned into any number of subsets as far as the total demand of each of them is $\text{CP}^{(j+1)}$. Using the relaxed condition we prove a few structural results which at the end shows that there exists an optimal mirror function $\mathcal{P}$ where for each node v and each level j, there is at most one Level-(j) H-node $a_H^{(j)}$ (we refer to $\mathcal{P}(a_H^{(j)})$ as an (v,j)-active set) such that neither $\mathcal{P}(a_H^{(j)}) \cap \text{SUB}(v) = \emptyset$ nor $\mathcal{P}(a_H^{(j)}) \subseteq \text{SUB}(v)$ holds.

From the above conditions we note that it is just enough to maintain the status of (v,j)-active set for each Level-(j). We use a dynamic programming to find the best solution which has at most one (v,j)-active set at each Level-(j). The dynamic programming is involved since we need to keep the consistency among all the levels.

2. PRELIMINARY

In this section we introduce the required definitions which we use throughout the paper. We use subscript H in order to refer to nodes or edges of H but we drop the subscript for nodes or edges of G. We also note that proofs that are omitted due to page constraint can be found in a full version of the paper.

We call cost multiplier cm is normalized if $cm(h) = 0$. The following lemma proves that an α-approximation algorithm for HGP with a normalized cost multipliers can be changed to an algorithm with at most the same approximation factors for HGP with a general cost multipliers.

Lemma 1 *Let ALG be an α-approximation algorithm for HGP which works only for normalized cost multipliers. Using ALG we can find algorithm ALG' which is an α-approximation algorithm for HGP working for any cost multiplier.*

In the rest of the paper we assume that the cost multiplier cm is normalized and hence $cm(h) = 0$.

For a subset of nodes $P \subseteq V(G)$, we define $\text{CUT}(P)$ to be the set of all edges that have one endpoint in P and one endpoint in $V(G) \setminus P$. We use $w(\text{CUT}(P))$ to refer to the sum of the weights of the edges in $\text{CUT}(P)$. We use $\text{SUB}(a_H)$ to denote the subtree rooted at node a_H in tree H.

Recall the definition of mirror function $\mathcal{P}$ from Section 1.2. In the following lemma we show that the cost of solution p and mirror function $\mathcal{P}$ are the same.

Lemma 2 *The cost assigned to solution p (shown in Equation (1)) is equal to the cost of its mirror function $\mathcal{P}$ (shown in Equation (3)).*

Working with mirror function $\mathcal{P}$ as opposed to p has the benefit that it exploits better the fact that graph H is a tree. In fact we can build a solution to the HGP problem by building a mirror function as follows. At the start all the nodes in G are assigned to the Level-(1) node of H (which is root r_H) and hence $\mathcal{P}(r_H) = V(G)$. After that, we partition nodes of G into smaller sets which we refer to them as Level-(2) sets. We assign each Level-(2) set to a Level-(2) node of H. Afterward, we take each Level-(2) set and partition it into smaller sets (Level-(3) sets) and assign them to the Level-(3) nodes. In the rest of the paper we work only with mirror functions and refer to them as solutions to the HGP problem.

3. HGP ON TREES

First we formally define the HGP problem on Trees (HGPT) where the objective is to partition only the leaves of a tree. Remember that the hierarchal separation tree H is $\text{DEG}^{(j)}$ regular at each Level-(j), *i.e.*, each H-node at Level-(j) has exactly $\text{DEG}^{(j)}$ children. Moreover the capacity of each Level-(h) H-node is one. We assign capacity $\text{CP}^{(j)}$ to each Level-(j) H-node ($a_H^{(j)}$) which is the total capacity of all the leaves in $\text{SUB}(a_H^{(j)})$. Because H is regular at each level we have $\text{CP}^{(j)} = \prod_{j'=j}^{h} \text{DEG}^{(j')}$. Remember that cm is the cost multiplier for the different layers of hierarchical separation H.

Definition 2 *Let T be a tree rooted at r where each leaf v of T represents a job with demand $0 < d(v) \leq 1$. The function $p : \text{LEAVES}(T) \rightarrow \text{LEAVES}(H)$ where $\forall a_H \in \text{LEAVES}(H)$, $\sum_{v:p(v)=a_H} d(v) \leq 1$ is a solution to the HGPT problem.*

Remember form Section 2 that from a mirror function $\mathcal{P}$ we can uniquely define its corresponding solution p to HGP and its cost is the same as p. Therefore, we refer to the mirror functions as solutions.

We note that solving HGP for only the leaves of a tree (as we defined in Definition 2) is more general than solving it for all its nodes. The later can be reduced to the former by adding a dummy leaf for each node and connecting them by an edge of infinity weight. As no partition separates a node from its dummy node, any solution partitioning only leaves in the modified tree can be transformed to a solution which partitions all the nodes in the original tree with the same cost. Moreover, every solution which partitions the nodes in the original tree defines a corresponding solution in the leaves of the modified graph with the same cost. Therefore, the optimal cost is the same in both trees.

Let tree T be the communication graph where each of its leaves v corresponds to a job and has demand $d(v)$. Remember that $\text{CP}^{(j)}$ is the capacity of each Level-(j) H-node and that each Level-(j) H-node has exactly $\text{DEG}^{(j)}$ children. Here we have $\text{CP}^{(h)} = 1$ and $\text{CP}^{(j)} = \prod_{i=j}^{h-1} \text{DEG}^{(i)}$ since H is $\text{DEG}^{(j)}$ regular at each Level-(j).

An intuitive way of seeing a solution to the HGPT problem is the following. At the start we have a single set which is equal to $\text{LEAVES}(T)$ we assign this set to r_H ($\mathcal{P}(r_H) = \text{LEAVES}(T)$). Now we need to partition $\mathcal{P}(r_H)$ into $\text{DEG}^{(1)}$ sets and assign each of them to a children of r_H such that the total demand of each set is at most $\text{CP}^{(1)}$. Then we partition the set assigned to each children ($a_H^{(1)}$) of r_H into $\text{DEG}^{(2)}$ sets each with total demand $\text{CP}^{(2)}$ and assign them to the children of $a_H^{(1)}$. More formally, at iteration j for each Level-(j) H-node $a_H^{(j)}$ we partition $\mathcal{P}(a_H^{(j)})$ into $\text{DEG}^{(j)}$ sets each with total demand at most $\text{CP}^{(j+1)}$. Let $\mathcal{S}^{(j)}$ be a collection containing all the sets that are assigned to the Level-(j) nodes of H. In the following we show that knowing only $\mathcal{S}^{(j)}$ for $0 \leq j \leq h$ is enough to rebuild a solution to the HGPT problem since H is $\text{DEG}^{(j)}$ regular at each Level-(j) and each Level-(j) H-node has capacity $\text{CP}^{(j)}$.

First, we formally redefine a solution to the HGPT problem as follows.

Definition 3 *A solution to the HGPT problem is a family of collections $\mathcal{S}^{(0)}, \mathcal{S}^{(1)}, \ldots, \mathcal{S}^{(h)}$. We refer to $\mathcal{S}^{(j)}$ as the Level-(j) collection where each $S^{(j)} \in \mathcal{S}^{(j)}$ is a subset of $\text{LEAVES}(T)$. We refer to each $S^{(j)} \in \mathcal{S}^{(j)}$ as the Level-(j) set. We need to have the following properties.*

1. *There is exactly one Level-(0) set in $\mathcal{S}^{(0)}$.*
2. *For $j \in [h]$, all the Level-(j) sets form a partition of $\text{LEAVES}(T)$. i.e., no two Level-(j) sets intersect and $\bigcup_{S^{(j)} \in \mathcal{S}^{(j)}} S^{(j)} = \text{LEAVES}(T)$.*
3. *The total demand of each Level-(j) set is at most $\text{CP}^{(j)}$.*
4. *For $j \in [h-1]$, each Level-(j) set is a refinement of at most $\text{DEG}^{(j)}$ Level-$(j+1)$ sets, ie each Level-(j) set is exactly equal to the union of at most $\text{DEG}^{(j)}$ Level-$(j+1)$ sets from $\mathcal{S}^{(j+1)}$.*

Note that $\bigcup_{j=0}^{h} \mathcal{S}^{(j)}$ forms a laminar family for $\text{LEAVES}(T)$. The cost of this solution is $\sum_{j=1}^{h} \sum_{S^{(j)} \in \mathcal{S}^{(j)}} w_T(\text{CUT}_T(S^{(j)})) \cdot \frac{cm(j-1)-cm(j)}{2}$. where $\text{CUT}_T(S)$ for $S \subseteq \text{LEAVES}(T)$ is the minimum weight set of edges which separates the leaves in S from the rest of the leaves in T

The following lemma shows that Definition 2 and Definition 3 are the same.

Lemma 3 *For each solution defined by Definition 2 we have a corresponding solution defined by Definition 3 with the same cost and for each solution defined by Definition 3 we have a corresponding solution defined by Definition 2 with the same cost.*

In this section we prove the following theorem about the HGPT problem.

Theorem 2 *There is an algorithm for the HGP problem on trees which finds a solution partitioning the leaves of the tree with optimal cost and violates the capacity of H-nodes at most by a factor of $(1+h)\cdot(1+\epsilon)$ where ϵ is an arbitrary small positive value.*

The structure of our proof is as follows. We relax the HGPT problem by removing bound $\text{DEG}^{(j)}$ on the total number of subsets form Constraint 4 of Definition 3. The optimal cost of the relaxed HGPT problem is less than the optimal cost of HGPT as there are less constraints. We solve the relaxed HGPT problem in polynomial time by a dynamic programming. Finally we show that we can change a solution to the relaxed HGPT problem to a solution of HGPT by violating the demand constraint (Constraint 3 in Definition 3) by a factor of at most $(1+j)$, *i.e.*, each Level-(j) set has the total demand of at most $(1+j)\cdot \text{CP}^{(j)}$.

Definition 4 *A solution to the Relaxed Hierarchal Graph Partitioning on Tree (RHGPT) problem is a family of collections $\mathcal{S}^{(0)},\mathcal{S}^{(1)},\ldots,\mathcal{S}^{(h)}$, where*

1. *There is exactly one Level-(0) set in $\mathcal{S}^{(0)}$.*
2. *For $j\in[h]$, all the Level-(j) sets form a partition of $\text{LEAVES}(T)$, i.e., no two Level-(j) sets intersect and $\bigcup_{S^{(j)}\in\mathcal{S}^{(j)}} S^{(j)} = \text{LEAVES}(T)$.*
3. *The total demand of each Level-(j) set is at most $CP^{(j)}$.*
4. *For $j\in[h-1]$, each Level-(j) set is a refinement of Level-$(j+1)$ sets (there is no constraint on the number of them), ie, each Level-(j) set is exactly equal to the union of some of Level-$(j+1)$ sets from $\mathcal{S}^{(j+1)}$.*

The cost of this solution is $\sum_{j=1}^{h}\sum_{S^{(j)}\in\mathcal{S}^{(j)}} w_T(\text{CUT}_T(S^{(j)}))\cdot \frac{cm(j-1)-cm(j)}{2}$.

Remember that $\text{CUT}_T(S)$ for $S\subseteq\text{LEAVES}(T)$ is the minimum weight set of edges which separates the leaves in S from the rest of the leaves in T and if there are multiple such sets we select the one which minimizes the number of nodes that are connected to S. If there are still multiple such sets we select the one which comes lexicographically first. Let $T\setminus\text{CUT}_T(S)$ be the forest obtained form T after removing $\text{CUT}_T(S)$ from T.

Definition 5 *The* mirror set *of $\text{CUT}_T(S)$ where $S\subseteq\text{LEAVES}(T)$ which we denote by $N(S)$ to be the set of all nodes that are in the connected components of $T\setminus\text{CUT}_T(S)$ which have at least one node of S.*

Note that $N(S)$ contains at least nodes of S. We are not going to find $N(S)$ but we use this definition to prove structural properties about solutions of HGPT.

Let $\mathcal{S}^{(0)},\mathcal{S}^{(1)},\ldots,\mathcal{S}^{(h)}$ be an arbitrary solution to the RHGPT problem and $S^{(j)}\in\mathcal{S}^{(j)}$. We prove the following two lemmas about the structure of mirror set $N(S^{(j)})$.

Lemma 4 *For $j\in\{1,\ldots,h\}$, for any two distinct sets $S_1^{(j)},S_2^{(j)}\in\mathcal{S}^{(j)}$ we have $N(S_1^{(j)})\cap N(S_2^{(j)})=\emptyset$.*

Lemma 5 *For $j\in[h-1]$, for any two sets $S^{(j)}\in\mathcal{S}^{(j)}$ and $S^{(j+1)}\in\mathcal{S}^{(j+1)}$ where $S^{(j+1)}\subseteq S^{(j)}$ we have $N(S^{(j+1)})\subseteq N(S^{(j)})$.*

In the following we define a class of special solutions to the RHGPT problem which are easily tractable by dynamic programming.

Definition 6 *A nice solution to the RHGPT problem is a solution $s=(\mathcal{S}^{(0)},\mathcal{S}^{(1)},\ldots,\mathcal{S}^{(h)})$ where for each node $v\in V(T)$ and each level $j\in[h]$ the following holds.*

1. *There is at most one set ($S_v^{(j)}\in\mathcal{S}^{(j)}$) for which $v\in N(S_v^{(j)})$ we refer to this set if it exists as the (v,j)-active set.*
2. *For all sets $S_v^{(j)'}\in\mathcal{S}^{(j)}$ other than the (v,j)-active set ($S_v^{(j)'}\neq S_v^{(j)}$) we have either $N(S_v^{(j)'})\subseteq\text{SUB}(v)$ or $N(S_v^{(j)'})\cap\text{SUB}(v)=\emptyset$.*

Using Lemma 4 and Lemma 5 we can prove the following theorem.

Theorem 3 *There exists an optimal solution $(\mathcal{S}^{(0)},\mathcal{S}^{(1)},\ldots,\mathcal{S}^{(h)})$ to the RHGPT problem which is a nice solution.*

PROOF. First we note that any solution $s=(\mathcal{S}^{(0)},\mathcal{S}^{(1)},\ldots,\mathcal{S}^{(h)})$ has at most one (v,j)-active set for any $(v,j)\in V(T)\times[h]$. Because for any $S_{v,1}^{(j)},S_{v,2}^{(j)}\in\mathcal{S}^{(j)}$, by Lemma 4 mirror sets $N(S_{v,1}^{(j)})$ and $N(S_{v,2}^{(j)})$ are disjoint which implies that the mirror set of at most one set of $\mathcal{S}^{(j)}$ contains v. Therefore the Condition 1 of a nice solution in Definition 6 holds for any solution s.

Let $s=(\mathcal{S}^{(0)},\mathcal{S}^{(1)},\ldots,\mathcal{S}^{(h)})$ be a solution to RHGPT.

Definition 7 *For each node $v\in V(T)$ and each level $j\in[h]$ we call $S_v^{(j)}\in\mathcal{S}^{(j)}$ a (v,j)-*bad set *if $S_v^{(j)}$ is not the (v,j)-active set and neither $N(S_v^{(j)})\subseteq\text{SUB}(v)$ nor $N(S_v^{(j)})\cap\text{SUB}(v)=\emptyset$. We denote the sum of the total number of (v,j)-bad sets over all $v\in V(T)$ and $j\in[h]$ by $BS(s)$.*

Note that if we prove that there exists an optimal solution s^* for which $BS(s^*)$ is zero then both the conditions of Definition 6 hold and the proof of the theorem follows.

Let solution $s=(\mathcal{S}^{(0)},\mathcal{S}^{(1)},\ldots,\mathcal{S}^{(h)})$ be an optimal solution to RHGPT that has the minimum $BS(.)$ number. We prove by contradiction that for each node $v\in V(T)$ and each level $j\in[h]$ there is no (v,j)-bad set $S_v^{(j)}$ in s. Assume otherwise, let $j\in[h]$ be the maximum level number for which there exists node $v\in V(T)$ such that there is at least one (v,j)-bad set ($S_v^{(j)}$).

Since $S_v^{(j)}$ is not the (v,j)-active set, $N(S_v^{(j)})$ does not include v. Moreover $N(S_v^{(j)})\cap\text{SUB}(v)\neq\emptyset$ and $N(S_v^{(j)})\not\subseteq\text{SUB}(v)$. In the following we show that we can divide $S_v^{(j)}$ to two subsets $N(S_v^{(j)})\cap\text{SUB}(v)$ and $N(S_v^{(j)})\setminus\text{SUB}(v)$ to obtain a new solution which has less (v,j)-bad sets than s while having the same cost.

Let U_1 be $S_v^{(j)}\cap\text{SUB}(v)$ and U_2 be $S_v^{(j)}\setminus\text{SUB}(v)$.

Observation 1 *All the following facts hold for sets U_1 and U_2 (see Figure 1).*

1. $N(U_1) \subseteq \text{SUB}(v) - \{v\}$.
2. $N(U_2) \subseteq (T \setminus \text{SUB}(v))$.
3. $N(S_v^{(j)}) = N(U_1) \cup N(U_2)$.
4. $\text{CUT}_T(S_v^{(j)}) = \text{CUT}_T(U_1) \cup \text{CUT}_T(U_2)$

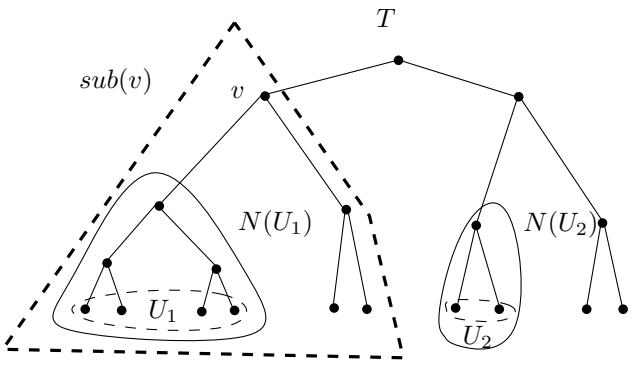

Figure 1: Dividing $N(S_v^{(j)})$ into U_1 and U_2.

We build solution $\hat{s} = (\hat{\mathcal{S}^{(0)}}, \hat{\mathcal{S}^{(1)}}, \ldots, \hat{\mathcal{S}^{(h)}})$ from s as follows. For $k \in [h] - \{j\}$ we have $\hat{\mathcal{S}^{(k)}} = \mathcal{S}^{(k)}$. Collection $\hat{\mathcal{S}^{(j)}}$ is the same as $\mathcal{S}^{(j)}$ except that we replace set $S_v^{(j)}$ with U_1 and U_2. In the full paper we show the following three claims: (1) $\hat{s}$ is a valid solution, (2) $BS(\hat{s})$ is less than $BS(s)$, and (3) $\hat{s}$ has the same cost as s and hence is an optimal solution. Existence of $\hat{s}$ contradicts with the fact that s is an optimal solution to RHGPT with minimum $BS(.)$. Therefore s does not have any (v, j)-bad set for any $(v, j) \in V(T) \times [h]$.

□

In the following we prove that the best nice solution can be found with dynamic programming which combined with Theorem 3 implies that we can find an optimal solution to the RHGPT problem.

Theorem 4 *We can find the best nice solution to the RHGPT problem by dynamic programming while violating the capacity constraints of Level-(j) sets (item 3 of Definition 4) by factor $(1 + \epsilon)$ for arbitrary small positive value ϵ.*

We can assume that each node of T has at most two children by repeating the following procedure. If node v of T has f_v children where $f_v > 2$, we create a binary tree of $f_v - 1$ dummy nodes where dummy nodes are connected to each other with the edges of weight infinity. If a dummy node has less than two children in addition to its child we connect it to children of v with the same weight as the edge between v and the children.

Remember that the processing demand of each node $v \in \text{LEAVES}(T)$ is $0 < d(v) \leq 1$. Let $0 < \epsilon$ be a sufficiently small constant. First we scale all the demands by factor ϵ/n and work with the new demand function d' where $d'(v) = \lfloor \frac{d(v)}{\epsilon/n} \rfloor$ for each leaf $v \in \text{LEAVES}(T)$. Note that the maximum value of d' is $\frac{n}{\epsilon}$ since $0 < d(v) \leq 1$. We also assume that the capacity of each Level-(j) H-node is scaled by factor ϵ/n. Note that the rounding error of the floor function in d' for each leaf is ϵ/n. As we can pack at most n nodes into one H-node, the total rounding error for each H-node is at most ϵ. Let $D = \sum_{v \in \text{LEAVES}(T)} d'(v)$ be the total demand of the leaves which is upper bounded by $\frac{n^2}{\epsilon}$ since we have at most n leaves. Here we develop an algorithm whose running time is polynomial in D and hence is polynomial in n and $\frac{1}{\epsilon}$.

We try to run the dynamic programming over the nodes of T in the sense that we define a subproblem for each $v \in V(T)$ and solve the subproblem by the solutions of subproblems defined for children of v.

Let $v \in V(T)$ be an arbitrary node in T and $s = (\mathcal{S}^{(0)}, \mathcal{S}^{(1)}, \ldots, \mathcal{S}^{(h)})$ be a nice solution. We define the induced family of collection $(s_v = (\mathcal{S}_v^{(0)}, \mathcal{S}_v^{(1)}, \ldots, \mathcal{S}_v^{(h)}))$ of s in v to be the family of collections where each set $S^{(j)} \in \mathcal{S}^{(j)}$ is replaced with $S^{(j)} \cap V(\text{SUB}(v))$ in $\mathcal{S}_v^{(j)}$ and if $S^{(j)} \cap V(\text{SUB}(v)) = \emptyset$ we completely remove $S^{(j)}$ from $\mathcal{S}_v^{(j)}$. Intuitively s_v is the reflection of s (in the sense that we only include nodes of $\text{SUB}(v)$) over the subtree $\text{SUB}(v)$.

Remember that for any nice solution $s = (\mathcal{S}^{(0)}, \mathcal{S}^{(1)}, \ldots, \mathcal{S}^{(h)})$ and each level $j \in [h]$ there is at most one (v, j)-active set (see Figure 2). This means that for all other Level-(j) sets ($S^{(j)} \in \mathcal{S}^{(j)}$) either mirror set $N(S^{(j)})$ and $\text{CUT}_T(S^{(j)})$ is fully in $\text{SUB}(v)$ or fully outside $\text{SUB}(v)$. For each Level-(j) set $S^{(j)}$ that is not the (v, j)-active set either $S^{(j)}$ retains all its elements or does not appear at all when induced to v form the Level-(j) collection of s_v.

Seeing from the opposite direction, in order to rebuild Level-(j) collection $\mathcal{S}^{(j)}$ from Level-(j) induced collection $\mathcal{S}_v^{(j)}$ we need to only add some new leaves to the (v, j)-active set of $\mathcal{S}_v^{(j)}$ while all the other sets of $\mathcal{S}_v^{(j)}$ remain intact, moreover, we might add some new sets of leaves to $\mathcal{S}^{(j)}$. This leads to an important property that we can replace the induced family of collection s_v over nodes of $\text{SUB}(v)$ with any other induced family of collection s'_v (again over nodes of $\text{SUB}(v)$) in solution s and obtain a valid solution as far as the total demand of the (v, j)-active set of s_v is the same as the (v, j)-active set of s'_v. We prove this property more formally later and show that this property results in the fact that an optimal solution is also an optimal induced family of collection over any node $v \in V(T)$. Therefore, we are able to show that the optimal induced family of collection for each node v can be found by considering the optimal induced family of collections over the children of v which is the main idea of our dynamic programming.

The following is a corollary of Lemma 5.

Corollary 1 *Let s be a nice solution of RHGPT. If for level $j \in [h]$ set $S^{(j)} \in \mathcal{S}^{(j)}$ is the (v, j)-active set then there exists set $S^{(j-1)} \in \mathcal{S}^{(j-1)}$ which is the $(v, j-1)$-active set and $S^{(j)} \subseteq S^{(j-1)}$.*

Let us denote the total demand of the (v, j)-active set of s by $D_v^{(j)}$ for any $j \in [h]$, *i.e.*, if $S^{(j)}$ is the (v, j) -active set we have $D_v^{(j)} = \sum_{v \in S^{(j)}} d'(v)$ for any $j \in [h]$. From Corollary 1 we can conclude that $D_v^{(j)} \geq D_v^{(j+1)}$ for any $j \in [h-1]$.

In the following definition we formally define a subproblem defined over subtree of v and the set of its valid solutions for any $v \in V(T)$. In fact, when solving for $\text{SUB}(v)$ we not only hierarchically partition leaves of $\text{SUB}(v)$, we also find a mirror set N_v which also shows how to cut the subset of the leaves from the rest. We will show that the only important information we need to retain when solving for the sub problem defined over $\text{SUB}(v)$ is the size of the (v, j)-active set for any $j \in [h]$ therefore we only have the h-tuple $(D_v^{(1)}, D_v^{(2)}, \ldots, D_v^{(h)})$ in the signature of the subproblem. We force the conditions of a RHGPT solution (Definition 4), con-

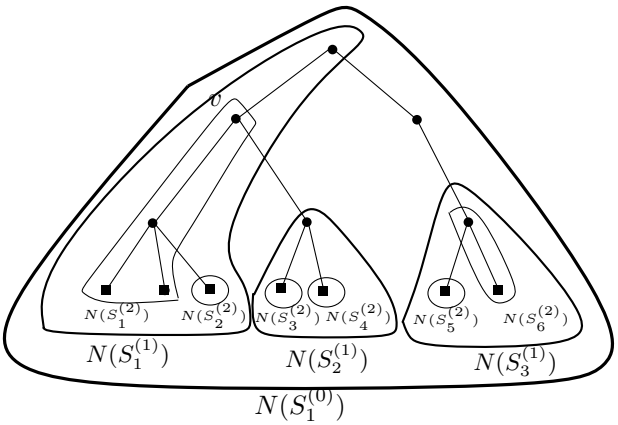

Figure 2: The mirror set $N(.)$ for all the sets in the collections of solution s. Here s has three collections $\mathcal{S}^{(0)}, \mathcal{S}^{(1)}, \mathcal{S}^{(2)}$ where $\mathcal{S}^{(0)} = \{S_1^{(0)}\}$, $\mathcal{S}^{(1)} = \{S_1^{(1)}, S_2^{(1)}, S_3^{(1)}\}$, and $\mathcal{S}^{(2)} = \{S_1^{(2)}, S_2^{(2)}, S_3^{(2)}, S_4^{(2)}, S_5^{(2)}, S_6^{(2)}\}$. Set $S_1^{(0)}$ is the $(v,0)$- active set, $S_1^{(1)}$ is the $(v,1)$-active set, and $S_1^{(2)}$ is the $(v,2)$-active set. Set $S_1^{(0)}$ is the $(v,0)$-active set, set $S_1^{(1)}$ is the $(v,1)$-active set, and set $S_1^{(2)}$ is the $(v,2)$-active set.

ditions of a nice solution (Definition 6, and conditions implied by Corollary 1 to the valid solutions of the subproblem.

Definition 8 *For a node $v \in V(T)$ an a h-tuple $(D_v^{(1)}, \ldots, D_v^{(h)}) \in \underbrace{[D] \times [D] \times \ldots \times [D]}_{h}$ (remember D is the upper bound on the total demand) which we refer to as signature, the goal of the subproblem $SP(v, (D^{(1)}, \ldots, D^{(h)}))$ is to find pair (s_v, N_v) where s_v is a family of collections $\left(\mathcal{S}_v^{(0)}, \mathcal{S}_v^{(1)}, \ldots, \mathcal{S}_v^{(h)}\right)$ and $N_v : \bigcup_j \mathcal{S}^{(j)} \rightarrow 2^{V(\text{SUB}(v))}$ is a mirror set. The partial solution (s_v, N_v) has to satisfy the following conditions.*

1. *s_v has to be a solution to the RHGPT problem defined over $\text{LEAVES}(\text{SUB}(v))$, i.e., all the conditions of Definition 4 hold over leaves of $\text{SUB}(v)$ for s_v.*
2. *For each $S_v^{(j)} \in \mathcal{S}_v^{(j)}$ for any $j \in [h]$ set $N_v(S_v^{(j)})$ has to contain only $S_v^{(j)}$ from the leaves in $\text{SUB}(v)$. From $N_v(S_v^{(j)})$ we can derive cut function $\text{CUT}_v : \bigcup_j \mathcal{S}^{(j)} \rightarrow 2^{E(\text{SUB}(v))}$ where $\text{CUT}_v(S_v^{(j)})$ is the set of all the edges that have exactly one endpoint in $N_v(S_v^{(j)})$.*
3. *For $j \in \{1, \ldots, h\}$, for any two distinct sets $S_1^{(j)}, S_2^{(j)} \in \mathcal{S}_v^{(j)}$ we have $N_v(S_1^{(j)}) \cap N_v(S_2^{(j)}) = \emptyset$.*
4. *For $j \in [h-1]$, for any two sets $S_v^{(j)} \in \mathcal{S}_v^{(j)}$ and $S_v^{(j+1)} \in \mathcal{S}_v^{(j+1)}$ where $S_v^{(j+1)} \subseteq S_v^{(j)}$ we have $N_v(S_v^{(j+1)}) \subseteq N_v(S_v^{(j)})$.*
5. *s_v and N_v have to be nice, i.e., for any $j \in [h]$ and any $u \in \text{SUB}(v)$ we have:*
   (a) *There is at most one set ($A_u^{(j)} \in \mathcal{S}_v^{(j)}$) for which $u \in N_v(A_u^{(j)})$ we refer to this set if it exists as the (u,j)-active set of s_v.*
   (b) *For all sets $S_v^{(j)} \in \mathcal{S}_v^{(j)}$ other than the (u,j)-active set ($S_v^{(j)} \neq A_u^{(j)}$) we have either $N_v(S_v^{(j)}) \subseteq \text{SUB}(v)$ or $N_v(S_v^{(j)}) \cap \text{SUB}(v) = \emptyset$. Note that if $D_v^{(j)} = 0$ then there is no (v,j)-active set and hence v is not in any mirror set of Level-(j) sets of $\mathcal{S}_v^{(j)}$.*
6. *Size of the (v,j)-active set of s_v has to be $D_v^{(j)}$, i.e., $\sum_{u \in A_v^{(j)}} d'(u) = D_v^{(j)}$ for any $j \in [h]$.*

cost of (s_v, N_v) which is denoted by $c_v(s_v, N_v)$ is

$$\sum_{j=1}^{h} \sum_{S_v^{(j)} \in \mathcal{S}_v^{(j)}} w(\text{CUT}_v(S_v^{(j)})) \cdot \frac{cm(j-1) - cm(j)}{2}.$$

Note that Corollary 1 implies that a signature $(D^{(1)}, \ldots, D^{(h)})$ is valid if $D^{(j)} > D^{(j+1)}$ for $j \in [h-1]$. Another property that $(D^{(1)}, \ldots, D^{(h)})$ has to have in order to be a valid signature is to not violate the Level-(j) H-nodes capacity ($\text{CP}^{(j)}$). In other words, $D^{(j)} \leq \text{CP}^{(j)}$ for all $j \in [h]$. Therefore, if signature $(D^{(1)}, \ldots, D^{(h)})$ is not valid then we return an empty solution with infinity cost for the subproblem $SP(v, (D^{(1)}, \ldots, D^{(h)}))$.

Remember that r is the root of T, therefore if (s_r, N_r) is a non-empty solution to subproblem $SP(r, (.))$ then s_r is a solution to RHGPT over T by Condition 1 in Definition 8.

Corollary 2 *Let (s_r, N_r) be a non-empty solution to subproblem $SP(r, (.))$ then $c_r(s_r, N_r) \geq c(s_r)$.*

Corollary 3 *Let s^* be an optimal nice solution to the RHGPT problem, $N(.)$ be the mirror set as defined in Definition 5, and for any $j \in [h]$ $D_r^{(j)}$ be the size of the (r,j)-active set of s^*. Then, (s^*, N) is a valid solution for subproblem $SP(r, (D_r^{(0)}, \ldots, D_r^{(h)}))$ and $c_r(s^*, N) = c_T(s^*)$.*

From corollaries 2 and 3 we conclude that there exists $(D_r^{(1)*}, \ldots, D_r^{(h)*}) \in [D] \times \ldots \times [D]$ such that the cost of the optimal solution to $SP(r, (D_r^{(1)*}, \ldots, D_r^{(h)*}))$ is the same as the cost of the optimal solution to the RHGPT problem. Therefore, we can find the optimal solution to the RHGPT problem by solving $SP(r, (D_r^{(1)}, \ldots, D_1^{(h)}))$ for any $(D_r^{(1)}, \ldots, D_1^{(h)}) \in [D] \times \ldots \times [D]$ and take the solution which has the minimum cost. In the appendix, we show how to solve $SP(v, (D_v^{(1)}, \ldots, D_v^{(h)}))$ for any $v \in V(T)$ and $(D_v^{(1)}, \ldots, D_v^{(h)}) \in [D] \times \ldots \times [D]$ using dynamic programming.

Dynamic Programming:. **Base case.** If v is a leaf then for subproblem $SP(v, (d'(v), \ldots, d'(v))$ we return a solution whose cost is zero and $\{v\}$ is the only set in each collection $\mathcal{S}_v^{(j)}$ which is also the (v,j)-active set for any $j \in [h]$. For any other signatures in subproblem $SP(v, .)$ we return an empty solution with infinity cost.

Recursive Step. In the following we show that $SP(v, (D_v^{(1)}, \ldots, D_v^{(h)}))$ is solvable for any $(D_v^{(1)}, \ldots, D_v^{(h)})$ knowing the solution to the subproblems of the children of v. Therefore, we can start from the leaves of T and recursively solve subproblem SP till reach to root r.

Here we assume that v has two children v_1 and v_2. The case when v has only one child is easier. Let $(D_v^{(1)}, \ldots, D_v^{(h)}) \in$

$[D] \times \ldots \times [D]$ be an arbitrary valid signature we focus on solving subproblem $SP(v, (D_v^{(1)}, \ldots, D_v^{(h)}))$.

We define induce function INDUCE which takes a solution to $SP(v, (D_v^{(1)}, \ldots, D_v^{(h)}))$ and a child of v and induce the solution to the subtree defined over the given child. More formally, let (s_v, N_v) be an optimal solution for $SP(v, (D_v^{(1)}, \ldots, D_v^{(h)}))$ then $\text{INDUCE}(v_1, s_v, N_v)$ is equal to (s_1, N_1). Here $s_1 = (\mathcal{S}_1^{(0)}, \mathcal{S}_1^{(1)}, \ldots, \mathcal{S}_1^{(h)})$ is the family of collections where for each set $S_v^{(j)} \in \mathcal{S}_v^{(j)}$ if $S_v^{(j)} \cap V(\text{SUB}(v_1)) \neq \emptyset$ we add $S_v^{(j)} \cap V(\text{SUB}(v_1))$ to $\mathcal{S}_1^{(j)}$ with mirror set $N_1(S_v^{(j)} \cap V(\text{SUB}(v_1))) = N_v(S_v^{(j)}) \cap V(\text{SUB}(v_1))$.

The important property we have about $SP(v, (D_v^{(1)}, \ldots, D_v^{(h)}))$ is that there exist an optimal solution which is also an optimal solution when induced to v_1 or v_2.

Lemma 6 *There exists an optimal solution (s_v^*, N_v^*) to $SP(v, (D_v^{(1)}, \ldots, D_v^{(h)}))$, $(D_1^{(1)}, \ldots, D_1^{(h)}), (D_2^{(1)}, \ldots, D_2^{(h)}) \in [D] \times \ldots \times [D]$ such that $\text{INDUCE}(v_1, s_v^*, N_v^*)$ is an optimal solution to $SP(v_1, (D_1^{(1)}, \ldots, D_1^{(h)}))$ and $\text{INDUCE}(v_2, s_v^*, N_v^*)$ is an optimal solution to $SP(v_2, (D_2^{(1)}, \ldots, D_2^{(h)}))$.*

From Lemma 6 we know that there exists an optimal solution for v which is an optimal solution for subproblems defined for both children v_1 and v_2 when induced to them. Therefore, in order to solve $SP(v, (D_v^{(1)}, \ldots, D_v^{(h)}))$, we loop over all the signatures $(D_1^{(0)}, \ldots, D_1^{(h)}), (D_2^{(0)}, \ldots, D_2^{(h)}) \in [D] \times \ldots \times [D]$ and solve subproblems $SP(v_1, (D_1^{(0)}, \ldots, D_1^{(h)}))$ and $SP(v_2, (D_2^{(0)}, \ldots, D_2^{(h)}))$ and merge their optimal solutions. Note that in the merging process the edge vv_1 which connects v to its children might be in the CUT_T of some Level-(j) sets and hence adds additional cost to the merged solution (the same thing can happen for vv_2). In the following we show carefully that the extra cost of merging two solutions of children of v only depends on the signatures of the solutions.

Let (s_1, N_1) be a solution for node v_1 with signature D_1^h and (s_2, N_2) be a solution for node v_2 with signature D_2^h. Suppose we want to merge these two solutions and obtain solution (s_v, N_v) for v such that it has signature D_v^h.

Consider a specific level number $j \in [h]$. Let the Level-(j) collection of s_1 be $\mathcal{S}_1^{(j)}$ and the Level-(j) collection of s_2 be $\mathcal{S}_2^{(j)}$ which we want to join and obtain the Level-(j) collection $(\mathcal{S}_v^{(j)})$ of s_v. Note that all the sets of $\mathcal{S}_1^{(j)}$ will remain intact when merging with $\mathcal{S}_2^{(j)}$ except the (v_1, j)-active set (see Condition 5 of Definition 8). In fact the only two sets that could merge together in $\mathcal{S}_v^{(j)}$ are the (v_1, j)-active set of $\mathcal{S}_1^{(j)}$ and the (v_2, j)-active set of $\mathcal{S}_2^{(j)}$. From the signatures of solutions we know that the size of (v_1, j)-active set is $D_1^{(j)}$ and the size of (v_2, j)-active set is $D_2^{(j)}$

For node v_1 (the case for v_2 is the same) and each level number j_1 we have two choices: (1) either we cut edge vv_1 (edge vv_1 is included in CUT_T of Level-(j_1) (v_1, j_1)-active set of s_1) and pay $w(vv_1) \cdot \frac{cm(j_1-1)-cm(j_1)}{2}$ or (2) we do not cut vv_1 at level j_1. In the former the (v_1, j_1)-active set of v_1 which has total demand of $D_1^{(j)}$ is closed and is not part of the (v, j_1)-active set of v. In the later if the (v_1, j_1)-active set has a non-zero total demand $(D_1^{(j)} \neq 0)$ then it is part of the (v, j_1)-active set, moreover, as the (v, j_1)-active set of v is a subset of all (v, k)-active sets for $k < j_1$ we cannot cut vv_1 for all the other lower levels. In other words, for node v_1 we need to decide about level j_1 where we cut edge vv_1 for all the levels $k > j_1$ and keep edge vv_1 for all levels $k \leq j_1$. This leads to definition of consistency between two signatures associated with children of v as follows.

Definition 9 *A valid signature $(D_1^{(1)}, \ldots, D_1^{(h)})$ is (j_1, j_2)-consistent with signature $(D_2^{(1)}, \ldots, D_2^{(h)})$ regarding signature $(D^{(1)}, \ldots, D^{(h)})$ if the following holds. For each $k \leq \min(j_1, j_2)$ we have $D^{(k)} = D_1^{(k)} + D_2^{(k)}$. For each $k > \max(j_1, j_2)$ we have $D^{(k)} = 0$. If $j_1 \geq j_2$ then for each $j_2 < k \leq j_1$ we have $D^{(k)} = D_1^{(k)}$ otherwise for each $j_1 < k \leq j_2$ we have $D^{(k)} = D_2^{(k)}$.*

Now we explain how to merge the partial solutions of two children of v to get a partial solution to v. Remember $s_1 = (\mathcal{S}_1^{(0)}, \ldots, \mathcal{S}_1^{(0)})$ is a partial solution in node v_1 with signature $(D_1^{(1)}, \ldots, D_1^{(h)})$ and $s_2 = (\mathcal{S}_2^{(0)}, \ldots, \mathcal{S}_2^{(0)})$ is a partial solution in v_2 with signature $(D_2^{(1)}, \ldots, D_2^{(h)})$. Assume that $(D_1^{(1)}, \ldots, D_1^{(h)})$ and $(D_2^{(1)}, \ldots, D_2^{(h)})$ are (j_1, j_2)-consistent with regard to $(D^{(1)}, \ldots, D^{(h)})$. Without loss of generality assume $j_1 \leq j_2$. The output of function $merge(s_{v_1}, j_1, s_{v_2}, j_2)$ is a partial solution $s_v = (\mathcal{S}_v^{(0)}, \ldots, \mathcal{S}_v^{(0)})$ for v where

- For $k > j_2$ we have $\mathcal{S}^{(k)} = \mathcal{S}_1^{(k)} \cup \mathcal{S}_2^{(k)}$ and there is no (v, k)-active set (size of the (v, k)-active set is zero).
- For $j_1 < k \leq j_2$ we have $\mathcal{S}^{(k)} = \mathcal{S}_1^{(k)} \cup \mathcal{S}_2^{(k)}$ and the (v, k)-active set is the (v_2, k)-active set of s_2.
- For $k \leq j_1$ lets $S_1^{(k)}$ be the (v_1, k)-active set of s_1 and $S_2^{(k)}$ be the (v_2, j_2)-active set of s_2 then $\mathcal{S}^{(k)} = \mathcal{S}_1^{(k)} \cup \mathcal{S}_2^{(k)}$ in which we replace $S_1^{(k)}$ and $S_2^{(k)}$ with $S_1^{(k)} \cup S_2^{(k)}$. The (v, k)-active set in this case is $S_1^{(k)} \cup S_2^{(k)}$.

Note that for each level $k > j_1$ if size of (v_1, k)-active set of s_1 $(D_1^{(k)})$ is larger than zero then we cut edge (vv_1) and pay $w(vv_1) \cdot \frac{cm(k-1)-cm(k)}{2}$; a similar argument holds for v_2. The partial cost of s_v is

$$\begin{aligned} c(s_v) &= c(s_1) + c(s_2) \\ &+ \sum_{k:k>j_1 \wedge D_1^{(k)} \neq 0} w(vv_1) \cdot \frac{cm(k-1) - cm(k)}{2} \\ &+ \sum_{k:k>j_2 \wedge D_2^{(k)} \neq 0} w(vv_1) \cdot \frac{cm(k-1) - cm(k)}{2} \quad (4) \end{aligned}$$

Now we prove the following claim to show how we can find an optimal partial solution for v given optimal partial solutions of its children.

Claim 1 *Let v be a node in T which has two children v_1 and v_2. We can find an optimal solution for subproblem $SP(v, (D^{(1)}, \ldots, D^{(h)}))$ if for any signature $(D_1^{(1)}, \ldots, D_1^{(h)})$ we know an optimal solution to $SP(v_1, (D_1^{(1)}, \ldots, D_1^{(h)}))$ and for any signature $(D_2^{(1)}, \ldots, D_2^{(h)})$ we know an optimal solution to $SP(v_2, (D_2^{(1)}, \ldots, D_2^{(h)}))$.*

PROOF. We run Dynamic Program (DP) algorithm as follows

1. Initialize s_v with an empty solution with cost infinity.
2. For each valid signature $(D_1^{(1)}, \ldots, D_1^{(h)}), (D_2^{(1)}, \ldots, D_2^{(h)}) \in \underbrace{[D] \times [D] \times \ldots \times [D]}_{h}$ and each $j_1, j_2 \in [h]$.

(a) $s_1 = SP(v_1, (D_1^{(1)}, \ldots, D_1^{(h)}))$.

(b) $s_2 = SP(v_2, (D_2^{(1)}, \ldots, D_2^{(h)}))$.

(c) if $c(s_1)$ and $c(s_2)$ are not infinity and $(D_1^{(1)}, \ldots, D_1^{(h)})$ and $(D_2^{(1)}, \ldots, D_2^{(h)})$ are (j_1, j_2)-consistent with regard to $(D^{(1)}, \ldots, D^{(h)})$ then:

i. $s'_v = merge(s_1, j_1, s_2, j_2)$.

ii. if $c(s'_v) < c(s_v)$ then

A. $s_v = s'_v$.

3. Return s_v.

Finally note that the running time of DP is D^{2h+2}. □

Note that we have in total $O(n)$ nodes and D^h different signatures where solving each subproblem SP takes $O(D^{2h+2})$. Therefore the total running time is $O(n \cdot D^{3h+2})$.

Finally in the following theorem, we show that we can transform any solution to the RHGPT problem to a solution of HGPT by violating the demand capacities.

Theorem 5 *Any solution to the RHGPT problem (Definition 4) can be transformed to a solution of HGPT (Definition 3) without increasing the cost by violating the demand capacities of each Level-(j) set (Constraint 3 in Definition 3) by a factor of $(1 + j)$.*

We apply the procedure of Theorem 5 to the solution of Theorem 4 which gives violation factor $(1 + \epsilon)(1 + h)$ in capacity and since cost is preserved optimally, we get the result stated in Theorem 2.

4. HGP ON ARBITRARY GRAPH G

We embed G into a probability distribution π of p decomposition trees $T_1, T_2, \ldots, T_p$ using Räecke's congestion minimization embedding [24]. A decomposition tree (T) comes along with a node mapping function $m_V : V(T) \to V(G)$ and an edge mapping function $m_E : E(T) \to E^*(G)$. Function m_V maps each vertex of T to a node in G such that it induces a bijection between the leaves of T and the nodes in G. Function m_E maps each edge $e_T = (u_T, v_T)$ of T to a corresponding path in G that connects $m_V(u_t)$ to $m_V(v_T)$. Let function m'_V be the reverse of m_V on the leaves of T which maps each node in G to a leaf in T.

Let u_T and v_T be two neighboring nodes in T with edge $e_T = (u_T, v_T)$ between them. The weight of edge e_T is defined as $w_T(u_T, v_T) = \sum_{m'(u) \in V_{u_T}, m'(v) \in V_{v_T}} w(u, v)$ where V_{u_T}, and V_{v_T} denote the two sets in the partition created by edge e_T over $\text{LEAVES}(T)$.

Let P_T be an arbitrary subset of leaves in T. We use $m(P_T)$ to refer to a set of nodes in G where $u \in m(P_T)$ if and only if $m'(u) \in P_T$. We define $\text{CUT}_T(P_T)$ where $P_T \subseteq \text{LEAVES}(T)$ to be the minimum weight set of edges which separates the leaves in P_T from the rest of the leaves in T. In other words, if we remove the edges of $\text{CUT}_T(P_T)$ from T then there will be no path between any leaf in P_T and any leave outside of P_T. If there are multiple such sets we select the one which minimizes the number of nodes that are connected to P_T . If there are still multiple such sets we select the one which comes lexicographically first. (We use this property in the next section when we design our algorithm). The following proposition is a direct result of the way we set the weight of each edge in decomposition tree T.

Proposition 1 *Let P_T be a subset of leaves of decomposition tree T then we have $w_T(\text{CUT}_T(P_T)) \geq w(\text{CUT}(m(P_T)))$.*

A flow $(\mathcal{F}_{s,t})$ between source node s and target t is a set of paths where each path $p_{s,t}$ starts from s, ends at t, and carries $f^p_{s,t}$ amount of flow. Here $\mathcal{F}_{s,t}$ carries the total flow of $\sum_{p \in \mathcal{F}_{s,t}} f^p_{s,t}$. A multi-commodity flow $(\mathcal{MF})$ is the set of $\binom{n}{2}$ flows, one flow for each pair of nodes $\{u, v\}$ in G. The congestion imposed by $\mathcal{MF}$ on G is defined as $max_e\ Load_{\mathcal{MF}}(e)/w(e)$ where $Load_{\mathcal{MF}}(e)$ is the total amount of flow that passes through edge e.

Similar to the edges and nodes we define functions m and m' in order to map a multi-commodity flow from decomposition tree T to G and vice versa. For a given multi-commodity flow $\mathcal{MF}$ in G we can obtain the corresponding multi-commodity flow $\mathcal{MF}_T = m'(\mathcal{MF}_T)$ in decomposition tree T by routing the total flow between each pair of nodes u and v in $\mathcal{MF}$ in the unique path between $m'_V(u)$ and $m'_V(v)$. Similarly, for a given multi-commodity flow $\mathcal{MF}_T$ in decomposition tree T we can obtain the corresponding multi-commodity flow $\mathcal{MF} = m(\mathcal{MF}_T)$ in G by routing all the flow passing through each edge e_T through the path $m_E(e_T)$ in G.

The main contribution of Räecke [24] on embedding graph G into a convex combination of decomposition trees is the following theorem.

Theorem 6 ([24]) *For a graph G there exists a convex combination of $p = O(|E| \log n)$ decomposition trees $T_1, \ldots, T_p$ where each tree T_i has a multiplier λ_i with $\sum_{i=1}^p \lambda_i = 1$ such that the following holds. Suppose we are given a multi-commodity flow $\mathcal{MF}_i$ for each tree T_i with congestion at most C. Then if we map all the multi-commodity flows $\mathcal{MF}_i$ on G when scaled by multiplier λ_i, results in a multi-commodity flow $\mathcal{MF} = \sum_{i=1}^p \lambda_i m_{T_i}(\mathcal{MF}_i)$ in G with congestion at most $O(\log n \cdot C)$.*

Our goal is to use the above theorem to show by solving HGP efficiently in decomposition trees we can obtain an $O(\log n)$-approximation algorithm for HGP in G. Before we start we define a solution to the HGP problem in a decomposition tree. Note that there is a bijection between leaves of a decomposition tree and nodes of G. Therefore we are only interested in assigning leaves of a decomposition trees to the H-nodes.

Definition 10 *For a decomposition tree T, mirror function $\mathcal{P}_T$ is a solution to the HGP problem in T where for each $a_H^{(j)} \in V(H)$ we have $\mathcal{P}_T(a_H^{(j)})$ is a subset of leaves of T. The cost of solution $\mathcal{P}_T$ in T is*

$$C_T(\mathcal{P}_T) = \sum_{a_H^{(j)} \in V(H)} w_T(\text{CUT}_T(\mathcal{P}_T(a_H^{(j)}))) \cdot \frac{cm(j-1) - cm(j)}{2}.$$

Let $\mathcal{P}_T$ be a solution to the HGP in decomposition tree T assigning each of $\text{LEAVES}(H)$ to a subset of leaves of T. Similar to multi-commodity flows we use function m_V to map solution $\mathcal{P}_T$ to a solution in G. Let $\mathcal{P} = m(\mathcal{P}_T)$ be a solution for HGP in G where each $a_H \in V(H)$ is assigned to set $m(\mathcal{P}_T(a_H))$, more formally, $\mathcal{P} : V(H) \to 2^{V(G)}$ where $\mathcal{P}(a_H) = \{m_V(v_T) \in V(G) | v_T \in \mathcal{P}_T(a_H)\}$. It can be shown that the cost of a solution in decomposition tree T, $C_T(\mathcal{P}_T)$, is larger than its corresponding solution in G, $C(m(\mathcal{P}_T))$.

Theorem 7 *Let $T_1, \ldots, T_p$ where $p = O(|E| \log n)$ be the set of decomposition trees specified in Theorem 6. Let $\mathcal{P}_i^*$ be the optimum solution of HGP over the leaves of tree T_i and $\mathcal{P}^*$ be an optimal solution to the HGP problem in G. Then the cost of solution $m(\mathcal{P}_q^*)$ where $q = \arg\ min_i C_{T_i}(\mathcal{P}_i^*)$ is at most $\beta \cdot C(\mathcal{P}^*)$ where $\beta = O(\log n)$ in Theorem 6.*

We run the algorithm of Theorem 2 on each decomposition tree specified in Theorem 7 and select the solution with the best cost. Therefore, Theorem 1 which is our main result follows.

5. ACKNOWLEDGMENTS

The authors would like to thank anonymous reviewers for all their useful comments.

References

[1] K. Andreev and H. Räcke. Balanced graph partitioning. In *SPAA*, pages 120–124, 2004.

[2] M. Andrews, M.T. Hajiaghayi, H. Karloff, and A. Moitra. Capacitated metric labeling. In *SODA*, pages 976–995, 2011.

[3] S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings and graph partitioning. *J. ACM*, 56(2):5:1–5:37, 2009.

[4] Siew Yin Chan, Teck Chaw Ling, and Eric Aubanel. The impact of heterogeneous multi-core clusters on graph partitioning: an empirical study. *Cluster Computing*, 15(3):281–302, 2012.

[5] S. Chawla, R. Krauthgamer, R. Kumar, Y. Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. *Comput. Complex.*, 15(2):94–114, 2006.

[6] Jian Chen and Valerie E Taylor. Parapart: parallel mesh partitioning tool for distributed systems. *Concurrency: Practice and Experience*, 12(2-3):111–123, 2000.

[7] Karen Devine, Erik Boman, Robert Heaphy, Bruce Hendrickson, and Courtenay Vaughan. Zoltan data management services for parallel dynamic applications. *Computing in Science & Engineering*, 4(2):90–96, 2002.

[8] Guy Even, Joseph (Seffi) Naor, Satish Rao, and Baruch Schieber. Fast approximate graph partitioning algorithms. In *SODA*, pages 639–648, 1997.

[9] Uriel Feige and Robert Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM J. Comput.*, 31 (4):1090–1118, 2002.

[10] Andreas Emil Feldmann and Luca Foschini. Balanced Partitions of Trees and Applications. In *STACS*, volume 14, pages 100–111, 2012.

[11] Bugra Gedik, Henrique Andrade, Kun-Lung Wu, Philip S. Yu, and Myungcheol Doo. Spade: the system s declarative stream processing engine. In *SIGMOD*, pages 1123–1134, 2008.

[12] Lukasz Golab and Theodore Johnson. Data stream warehousing. In *SIGMOD*, pages 949–952, 2013.

[13] Dorit s. Hochbaum and David B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM J. Comput.*, 17(3):539–551, 1988.

[14] Subhash A. Khot and Nisheeth K. Vishnoi. The unique games conjecture, integrality gap for cut problems and embeddability of negative type metrics into ℓ 1. In *FOCS*, pages 53–62, 2005.

[15] Jon Kleinberg and Éva Tardos. Approximation algorithms for classification problems with pairwise relationships: metric labeling and markov random fields. *J. ACM*, 49(5):616–639, 2002.

[16] Robert Krauthgamer and Yuval Rabani. Improved lower bounds for embeddings into l1. In *SODA*, pages 1010–1017, 2006.

[17] Robert Krauthgamer, Joseph (Seffi) Naor, and Roy Schwartz. Partitioning graphs into balanced components. In *SODA*, pages 942–949, 2009.

[18] T. Leighton, F. Makedon, and S. Tragoudas. Approximation algorithms for vlsi partition problems. In *IEEE International Symposium on Circuits and Systems*, pages 2865–2868 vol.4, 1990.

[19] Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, November 1999.

[20] Irene Moulitsas and George Karypis. *Architecture aware partitioning algorithms*. Springer, 2008.

[21] J.S. Naor and R. Schwartz. Balanced metric labeling. In *STOC*, pages 582–591. ACM, 2005.

[22] François Pellegrini. Static mapping by dual recursive bipartitioning of process architecture graphs. In *Scalable High-Performance Computing Conference, 1994., Proceedings of the*, pages 486–493. IEEE, 1994.

[23] François Pellegrini and Jean Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *High-Performance Computing and Networking*, pages 493–498. Springer, 1996.

[24] H. Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *STOC*, pages 255–264. ACM, 2008.

[25] Horst D. Simon and Shang-Hua Teng. How good is recursive bisection? *SIAM J. Sci. Comput.*, 18(5):1436–1445, 1997.

[26] James D Teresco, Jamal Faik, and Joseph E Flaherty. Hierarchical partitioning and dynamic load balancing for scientific computation. In *Applied Parallel Computing. State of the Art in Scientific Computing*, pages 911–920. Springer, 2006.

[27] Jesper Larsson Traff. Implementing the mpi process topology mechanism. In *Supercomputing, ACM/IEEE 2002 Conference*, pages 28–28. IEEE, 2002.

[28] Chris Walshaw and Mark Cross. Parallel optimisation algorithms for multilevel mesh partitioning. *Parallel Computing*, 26(12):1635–1660, 2000.

[29] Chris Walshaw and Mark Cross. Multilevel mesh partitioning for heterogeneous communication networks. *Future generation computer systems*, 17(5):601–623, 2001.