Balanced Graph Partitioning

Konstantin Andreev* Harald Räcke †

Abstract

We consider the problem of partitioning a graph into k components of roughly equal size while minimizing the capacity of the edges between different components of the cut. In particular we require that for a parameter $\nu \geq 1$, no component contains more than $\nu \cdot \frac{n}{k}$ of the graph vertices.

For $k = 2$ and $\nu = 1$ this problem is equivalent to the well known Minimum Bisection Problem for which an approximation algorithm with a polylogarithmic approximation guarantee has been presented in [FK02]. For arbitrary k and $\nu \geq 2$ a bicriteria approximation ratio of $O(\log n)$ was obtained by [ENRS99] using the spreading metrics technique.

We present a bicriteria approximation algorithm that for any constant $\nu > 1$ runs in polynomial time and guarantees an approximation ratio of $O(\log^{1.5} n)$ (for a precise statement of the main result see Theorem 6). For $\nu = 1$ and $k \geq 3$ we show that no polynomial time approximation algorithm can guarantee a finite approximation ratio unless $P = NP$.

1 Introduction

The problem of partitioning graphs into equal sized components while minimizing the number of edges between different components is extremely important in parallel computing. For example, parallelizing many applications involves the problem of assigning data or processes evenly to processors, while minimizing the communication.

This issue can be posed as the following graph partitioning problem. Given a graph $G = (V, E)$, where nodes represent data or tasks and edges represent communication, the goal is to divide V into equal sized parts $V_1, \ldots, V_k$ while minimizing the capacity of edges cut. Here k denotes the number of processors of the parallel machine on which the application is to be executed. Apart from this example, graph partitioning algorithms also play an important role in areas such as VLSI layout, sparse linear system solving and circuit testing and simulation.

The above graph partitioning problem is already NP-complete for the case $k = 2$, which is also called the Minimum Bisection problem. Due to the importance of the problem many effort has been made to develop efficient heuristics (see [KL70, FM82]) and approximation algorithms (see [FKN00, FK02]).

If k is not constant the problem is hard to approximate within any finite factor as we will show in Theorem 1. Therefore the following version of the problem with relaxed balance constraint was introduced. In the (k, ν)-balanced partitioning problem the task is to partition the graph into k pieces

*Mathematics Department, Carnegie-Mellon University, Pittsburgh PA 15213. This work was in part supported by the NSF under grants CCR-0085982 and CCR-0122581. Email: konst@cmu.edu

†Computer Science Department, Carnegie-Mellon University, Pittsburgh PA 15213. This work was supported by the NSF under grants CCR-0085982 and CCR-0122581. Email: harry@cs.cmu.edu

of size at most $\nu\frac{n}{k}$, while minimizing the capacity of edges between different partitions. However, the performance of a partitioning algorithm is compared to the optimum algorithm that has to create partitions of equal size and is thus more restricted. This modeling approach is sometimes called bicriteria-approximation, resource-augmentation, or pseudo-approximation in the literature.

Previous work about the (k,ν)-balanced partitioning problem only focuses on the case $\nu \geq 2$. In this case it is possible to get an approximation ratio of $O(\log n)$ as shown by Even et al. [ENRS99].

In this work we consider the $(k, 1+\varepsilon)$-balanced partitioning problem for arbitrary constants ε. We show that we can get a polylogarithmic approximation w.r.t. the capacity of edges between different partitions. Furthermore, we show that it is not possible to obtain even a finite approximation factor for the case $\varepsilon = 0$. The latter result is in some sense the justification for looking at bicriteria approximation algorithms instead of true approximation algorithms.

1.1 Related Work

The first heuristics for minimum bisection were given by Kernighan and Lin [KL70] and subsequently improved in terms of running time by Fiduccia and Mattheyses [FM82].

If the balance requirement of the minimum bisection problem is relaxed, one gets the well known graph separator problem. Since the seminal paper of Lipton and Tarjan [LT79] on graph separators in planar graphs, this subject has been intensively studied. For a comprehensive survey of major results in the area the reader is refered to the book "Graph separators with applications" by Rosenberg and Heath [RH01].

The first non-trivial approximation algorithm for minimum bisection is due to Saran and Vazirani [SV95] who obtained an approximation ratio of $n/2$. Subsequently the ratio was improved by Feige et al. [FKN00] to $O(\sqrt{n} \cdot \mathrm{polylog}(n))$. In their seminal paper Feige and Krauthgamer were able to improve the ratio to $O(\log^2 n)$ ([FK02]). An important part of their algorithm is the computation of the so called *amortized cuts*. They show that an approximation algorithm for sparsest cut can be transformed into an approximation algorithm for amortized cuts. Recently, Arora, Rao and Vazirani [ARV04] gave an improved approximation for sparsest cut with an approximation ratio of $O(\sqrt{\log n})$. Combining the Feige-Krauthgamer algorithm with this algorithm gives an $O(\log^{1.5} n)$ approximation to Minimum Bisection.

For some special graph classes better bounds are known. For example for planar graphs the algorithm by Feige and Krauthgamer achieves an approximation ratio of $O(\log n)$. In dense graphs (graphs of minimum degree $\Omega(n)$) Arora et al. [AKK95] gave a PTAS.

Some of the above algorithms can be extended to larger values of k. The algorithm by Saran and Vazirani gives an upper bound of $\frac{k-1}{k}n$ for the $(k,1)$-balanced partitioning problem. Feige and Krauthgamer extend their algorithm to give an approximation ratio of $O(\log^2 n)$ for general k ($O(\log^{1.5} n)$ when using the result of [ARV04]). Note however that this algorithm requires a running time that is exponential in k.

The relaxed version of the k partitioning problem for general graphs was considered by Leighton et al. [LMT90] and Simon and Teng [ST97]. Their solutions are based on recursive partitioning the graph with approximate separators, yielding an approximation factor of $O(\log n \log k)$. Even et al. [ENRS00] have used the spreading metrics technique to improve this to $O(\log n \log\log n)$ and later ([ENRS99]) to $O(\log n)$. In the latter paper it is also shown that any (k,ν)-balanced partitioning problem with $\nu > 2$ can be reduced to a (k',ν')-balanced partitioning problem with $k' \leq k$ and $\nu' \leq 2$, i.e., that it is only necessary to analyze the problem for small values of ν.

The above solutions for the $(k,2)$-balanced partitioning problem cannot be easily extended to $(k,1+\varepsilon)$-balanced partitioning problems with $\varepsilon < 1$. The reason is that the above solutions rely mainly on partitioning the graph into pieces of size less than n/k, while cutting as few edges, as possible. Obviously, an optimal algorithm for the $(k,1)$-balanced partitioning problem has to partition the graph into pieces of size at most n/k as well.

From this it can be deduced that their algorithm cuts only a small factor more edges than the optimal $(k,1)$-balanced partitioning. Since pieces of size less than n/k can be packed into k partitions such that no partition receives more than $2n/k$ nodes, they get a good solution to the $(k,2)$-balanced partitioning problem.

However, if we require that the balance-constraint is relaxed by less than a factor of 2, we have to carefully choose the piece-sizes into which the graph is cut since we have to be able to pack these pieces into k partitions of size at most $\nu\frac{n}{k}$. Therefore it is necessary to combine the graph partitioning algorithm with an algorithm for scheduling the resulting pieces into partitions of small size.

1.2 Basic Notations and Definitions

Throughout this paper $G=(V,E)$ denotes the input graph and $n=|V|$ denotes the number of nodes of G. We call P a partitioning of G into ℓ parts if it consists of a collection of ℓ disjoint subsets $V_1,\ldots,V_\ell$ that cover V, i.e. $V=V_1\dot\cup\cdots\dot\cup V_\ell$. We use $\text{cost}(P)$ to denote the capacity of edges *cut* by the partitioning, i.e. edges between different subsets V_i. We call the subsets V_i, partitions of G to distinguish them from other subsets of V that are not necessarily part of a partitioning.

We investigate the $(k,1+\varepsilon)$-balanced partitioning problem, that is the problem of finding a minimum cost partitioning of G into k parts such that each part contains at least $(1+\varepsilon)\frac{n}{k}$ nodes.

We seek a bicriteria approximation algorithm for this problem; we compare the cost of an approximation algorithm to the cost of the optimum $(k,1)$-balanced partitioning algorithm that has to output parts of exactly equal size and is thus more restricted. We call the result of the latter algorithm the optimum $(k,1)$-balanced partitioning and denote it with OPT_k.

Throughout the paper we assume that the unbalance factor ε is smaller than 1. This does not simplify the problem because the results of [ENRS99] show that an instance with $\varepsilon > 1$ can be reduced to a problem instance with $\varepsilon \leq 1$.

2 Hardness Results for Balanced Partitioning

It is already well known that the $(2,1)$-partitioning problem, in other words the Minimum Bisection problem, is NP-complete (see [GJ79]). As mentioned previously, Krauthgamer and Feige gave an efficient approximation algorithm for the Minimum Bisection problem (see [FK02]). We show in the following Theorem 1 that the general problem of partitioning a graph into k equal parts cannot be solved in polynomial time if k is not constant. Furthermore the theorem shows that it is not even possible to derive a finite approximation factor for the general problem where k is not a constant. This is the main motivation for analyzing bicriteria approximation algorithms for this case, i.e. algorithms that are allowed to violate the balance constraint of the partitions.

Theorem 1 *For $k \geq 3$ the $(k,1)$-balanced partitioning problem has no polynomial time approximation algorithm with finite approximation factor unless P=NP.*

Proof. We use a reduction from the 3-Partition problem defined as follows. We are given $n = 3k$ integers $a_1, a_2, \ldots, a_n$ and a threshold S such that $\frac{S}{4} < a_i < \frac{S}{2}$ and

$$\sum_{i=1}^{n} a_i = kS.$$

The task is to decide if the numbers can be partitioned into triples such that each triple adds up to S. This problem is *strongly* NP-complete (see [GJ79]), i.e. it is already NP-complete if all number are bounded by some polynomial in the length of the input.

The reduction is as follows. Assume that we have an approximation for $(k,1)$-balanced partition with finite approximation factor, and further we are given an instance of 3-Partition in which all numbers (i.e. the integers a_i and the threshold S) are polynomially bounded. We can use the $(k,1)$-balanced partition algorithm to solve the 3-Partition instance in the following way.

We construct a graph G such that for each number a_i it contains a clique of size a_i. Note that this graph is of polynomial size since the integers are polynomially bounded. If the 3-Partition instance can be solved, the $(k,1)$-balanced partitioning problem in G can be solved without cutting any edge. If the 3-Partition instance cannot be solved, the optimum $(k,1)$-balanced partitioning in G will cut at least one edge. An approximation algorithm with finite approximation factor has to differentiate between these two cases. Hence, it can solve the 3-Partition problem which is a contradiction under the assumption that P≠NP. ■

3 Algorithm and Analysis

3.1 Decomposing the graph

Our algorithm for solving the $(k, 1+\varepsilon)$-balanced partitioning problem proceeds in two phases. In a first preprocessing phase the graph is decomposed via a recursive decomposition scheme. This means we use a separation algorithm that gets a part of the graph as input, and outputs two non-empty subsets of this part, for recursively decomposing the graph. To each such recursive decomposition corresponds a binary decomposition tree T. Each node v_t of T corresponds to a part $V_{v_t} \subset V$ obtained during the decomposition as follows. The root of the tree corresponds to the node set V of the input graph G, the leaves correspond to individual vertices of G, and the two direct descendants of a node v_t correspond to the two subparts obtained when applying the separation algorithm to V_{v_t}.

The separation algorithm is as follows. We set $\varepsilon' := \frac{\varepsilon}{3}/(1+\frac{\varepsilon}{3})$. In each divide step the algorithm tries to find a minimum ε'-balanced separator, that is a partition of an input set V into two parts with size more than $\varepsilon' \cdot |V|$ that cuts a minimum number of edges. However, we allow our algorithm to relax the balance constraint. We use an approximation algorithm that returns an $\varepsilon'/2$-balanced cut, i.e. every part contains at least $\varepsilon'/2 \cdot |V|$ nodes, and that cuts a capacity of at most

$$O(S \cdot \sqrt{\log n}/\varepsilon),$$

where S denotes the total capacity of edges in an optimum ε'-balanced cut. In [LR99] it has been shown how to construct approximate balanced separators using an approximation algorithm for sparsest cut. Combining this algorithm with the recent approximation algorithm for the sparsest cut problem by Arora, Rao and Vazirani [ARV04] gives a polynomial time algorithm for computing the separators needed in our divide steps. Since each divide step reduces the larger side by a factor of at least $(1-\varepsilon'/2)$ the resulting decomposition tree has height $O(\log n/\varepsilon)$.

3.2 Coarse T-partitionings

To solve the $(k, 1+\varepsilon)$-balanced partitioning problem efficiently, we do not look at all possible partitionings, but we reduce the search space by only considering partitionings with a special structure that is induced by the decomposition tree T. In order to describe this structure we introduce the following definition.

Definition 2 *We call a partitioning a* T-partitioning *if it only contains subsets of T, i.e., subsets that occurred during the recursive decomposition process and that therefore correspond to some node of T. We call a T-partitioning* coarse *if it does not contain any small subsets. More precisely it does not contain subsets V_{v_t} for which the parent subset V_{v_p} (with v_p parent of v_t in the tree) contains less than $\frac{\varepsilon n}{3k}$ nodes.*

Our algorithm first computes a coarse T-partitioning P in which no partition is too large and that does not cut too many edges in the graph. However P may contain too many (i.e., more than k) partitions. In a second step the algorithm tries to transform P into a $(k, 1+\varepsilon)$-balanced partition by merging some of the subsets of P to reduce the number of partitions.

However, this latter step may not always be possible without violating the balance constraint. Therefore we introduce a definition that captures whether a given T-partitioning P can be transformed into a $(k, 1+\varepsilon)$-balanced partition. Let J denote the index set of sets in P, i.e., $P = \{V_j, j \in J\}$. We say that P is $(1+\varepsilon)$-*feasible* if J can be partitioned into k distinct index sets $J_1, \ldots, J_k$ such that

$$\forall \ell \in \{1, \ldots, k\}: \quad \sum_{j \in J_\ell} (1+\varepsilon/2)^{\lceil \log_{1+\varepsilon/2} |V_j| \rceil} \leq (1+\varepsilon) \cdot \frac{n}{k} \ .$$

Note that the expression $(1+\varepsilon/2)^{\lceil \log_{1+\varepsilon/2} |V_j| \rceil}$ rounds $|V_j|$ to the next power of $(1+\varepsilon/2)$.

The above definition is not straightforward in the sense that we do not simply require that $\forall \ell \in \{1, \ldots, k\}: \sum_{j \in J_\ell} |V_j| \leq (1+\varepsilon) \cdot \frac{n}{k}$, which means that the total size of all sets in a part is smaller than $(1+\varepsilon) \cdot \frac{n}{k}$. The reason for the rounding to powers of $(1+\varepsilon/2)$ is that this definition of $(1+\varepsilon)$-feasibility will allow us to check this property in polynomial time, whereas for the straightforward definition the decision whether a coarse T-partitioning is $(1+\varepsilon)$-feasible is NP-complete.

The following lemma shows that we only have to consider feasible coarse T-partitionings in order to solve the $(k, 1+\varepsilon)$-balanced partitioning problem with a polylogarithmic approximation ratio.

Lemma 3 *There exists a $(1+\varepsilon)$-feasible coarse T-partitioning that cuts at most $O(\log^{1.5} n/\varepsilon^2) \cdot \mathrm{cost}(\mathrm{OPT}_k)$ edges, where OPT_k denotes the optimum (exact) $(k,1)$-balanced partitioning.*

Proof. Think of the optimum partitioning OPT_k as a coloring of the graph with k distinct colors, where each color is used for exactly n/k nodes. We successively transform OPT_k into a coarse T-partitioning that is $(1+\varepsilon)$-feasible.

First we create a new partitioning OPT'_k that only contains subsets of T. This is done by the following algorithm.

Let $\mathscr{S}$ denote a collection of subsets of V that is initially empty. As long as there exists a node v of G that is not yet contained in a set of OPT'_k we traverse the path from the root of T to the leaf that represents v. We take the first set on this path that is colored "*almost monochromatically*" according to the coloring induced by OPT_k, and add it to the collection $\mathscr{S}$. *Almost monochromatically* means that at least a $1/(1+\varepsilon/3)$-fraction of the nodes of the set are colored the same. More formally, a set

$V_x \subset V$ is colored *almost monochromatically* if there is a color c such that at least $\frac{1}{1+\varepsilon/3}|V_x|$ nodes of V_x are colored with c. We call c the *majority color* of V_x since more than half of the nodes of V_x are colored with c (recall that we assumed $\varepsilon \leq 1$). Note that the sets that are taken by this algorithm are disjoint, i.e. $\mathscr{S}$ is a collection of disjoint subsets of V that upon the termination of the algorithm covers all nodes of V, i.e. $\mathscr{S}$ is a partitioning. We denote this final result of the algorithm with OPT'_k.

Now, we modify OPT'_k to get a coarse T-partitioning OPT''_k that contains no small sets. Initially, OPT''_k is equal to OPT'_k. We will successively remove small sets from OPT''_k in the following way. Suppose there is a set $V_{v_t} \in \mathrm{OPT}''_k$ such that the parent set V_{v_p} contains less than $\frac{\varepsilon n}{3k}$ nodes. The nodes in V_{v_p} are all contained in sets of OPT''_k that correspond to descendants of v_p in T. We remove all these sets from OPT''_k and add V_{v_p} instead. OPT''_k is still a T-partitioning after this step; therefore continuing in this manner gives a T-partitioning that contains no small sets, in other words a coarse T-partitioning.

In order to prove the lemma we now show that OPT''_k is $(1+\varepsilon)$-feasible and that $\mathrm{cost}(\mathrm{OPT}''_k) \leq O(\log^{1.5} n) \cdot \mathrm{cost}(\mathrm{OPT}_k)$.

Claim 4 OPT''_k *is* $(1+\varepsilon)$*-feasible.*

Proof. We first show that OPT'_k is $(1+\varepsilon)$-feasible. We do this by merging the sets to a $(k, 1+\varepsilon)$-balanced partition in the following way. Merge all sets of OPT'_k that have the same majority color into one partition. This gives at most k partitions. We have to show that no partition is too large with respect to the definition of $(1+\varepsilon)$-feasibility. Fix a partition and let J denote the index set of the sets in this partition. Further, let M_j denote the number of nodes in a set V_j, $j \in J$ that are colored with the majority color. We have

$$\sum_{j \in J} |V_j| = (1+\varepsilon/3) \cdot \sum_{j \in J} \frac{1}{1+\varepsilon/3} |V_j| \leq (1+\varepsilon/3) \cdot \sum_{j \in J} M_j = (1+\varepsilon/3) \cdot \frac{n}{k}$$

where the last equality holds since in OPT_k any color is used by exactly $\frac{n}{k}$ vertices. In the definition of $(1+\varepsilon)$-feasibility the sizes of subsets are rounded to powers of $(1+\varepsilon/2)$. This can increase the above sum only by a factor of $(1+\varepsilon/2)$. Hence, it is smaller than $(1+\varepsilon/2) \cdot (1+\varepsilon/3) \cdot \frac{n}{k} \leq (1+\varepsilon) \cdot \frac{n}{k}$.

Now we will show that OPT''_k is feasible as well. For this we have to decide in which partitions to put the sets of OPT''_k that are not contained in OPT'_k (all other sets remain in their partition). We do this greedily. For a given set that is not yet assigned we choose a partition that still contains less than n/k nodes and assign the set to this partition. It is guaranteed that we find a partition with less than n/k nodes since the total number of nodes is n and there are k partitions. In this manner no partition receives more than $(1+\varepsilon/3)\frac{n}{k}$ nodes, because all relevant sets have size at most $\frac{\varepsilon n}{3k}$. Again, rounding only increases this term by a factor of $(1+\varepsilon/2)$, which proves the claim. ■

The following claim shows that OPT''_k approximates well the optimum solution w.r.t. the capacity of edges cut.

Claim 5 $\mathrm{cost}(\mathrm{OPT}''_k) \leq O(\log^{1.5} n/\varepsilon^2) \cdot \mathrm{cost}(\mathrm{OPT}_k)$.

Proof. We show that $\mathrm{cost}(\mathrm{OPT}'_k) \leq O(\log^{1.5} n/\varepsilon^2) \cdot \mathrm{cost}(\mathrm{OPT}_k)$. This is sufficient since OPT''_k is obtained from OPT'_k by merging sets, which gives $\mathrm{cost}(\mathrm{OPT}''_k) \leq \mathrm{cost}(\mathrm{OPT}'_k)$.

The cost of OPT'_k is the capacity of all edges between different sets of the partitioning. A convenient interpretation of this cost is that it is created by the ancestors of sets in OPT'_k, in the following way. Suppose that there is an ancestor set V_{v_a} for some set V_{v_t} of OPT'_k (i.e., v_a is ancestor of v_t in the tree). We say that the cost created by V_{v_a} is the capacity of edges that are cut in the divide step of the recursive decomposition that divided V_{v_a}. In this way we have assigned the cost of OPT'_k to the ancestor sets in T.

Now, we amortize the cost created by an ancestor set V_{v_a} against the capacity of edges that are cut by the optimum partitioning OPT_k within V_{v_a}. Consider the partitioning of V_{v_a} induced by the optimum coloring of OPT_k. No partition has more than $\frac{1}{1+\varepsilon/3} \cdot |V_{v_a}| = (1-\varepsilon')|V_{v_a}|$ nodes because V_{v_a} is not colored almost monochromatically (here we used the definition of ε' as introduced in Section 3.1). We merge partitions of V_{v_a} into a set M in decreasing order of their sizes until M contains more than $\varepsilon' \cdot |V_{v_a}|$ nodes. Since $\varepsilon' < 1/3$ we have that $|M| \leq (1-\varepsilon')|V_{v_a}|$. This means M induces an ε'-balanced cut for V_{v_a}. OPT_k cuts all edges within this cut. Since, the divide step for V_{v_a} uses an approximation to the optimum ε'-balanced cut we get

$$\mathrm{cost}_{V_{v_a}}(\mathrm{OPT}'_k) \leq O(\sqrt{\log(n)}/\varepsilon) \cdot \mathrm{cost}_{V_{v_a}}(\mathrm{OPT}_k) \ ,$$

where $\mathrm{cost}_{V_{v_a}}(\mathrm{OPT}'_k)$ denotes the cost of OPT'_k created by V_{v_a}, and $\mathrm{cost}_{V_{v_a}}(\mathrm{OPT}_k)$ denotes the capacity of edges cut by the optimum partitioning within V_{v_a}.

Since, a single edge of G is at most used $\mathrm{height}(T)$ times for amortization we get that

$$\begin{aligned}\mathrm{cost}(\mathrm{OPT}'_k) &\leq \mathrm{height}(T) \cdot O(\sqrt{\log n}/\varepsilon) \cdot \mathrm{cost}(\mathrm{OPT}_k)\\ &\leq O(\log^{1.5} n/\varepsilon^2) \cdot \mathrm{cost}(\mathrm{OPT}_k) \ .\end{aligned}$$

This completes the proof of the claim. ■

Using Claims 4 and 5 proves the lemma. ■

3.3 The algorithm

In this section we show how to find an optimal $(1+\varepsilon)$-feasible coarse T-partitioning OPT_T. From Lemma 3 it follows that this partitioning gives a $(k, 1+\varepsilon)$-partitioning with an approximation factor of $O(\log^{1.5} n/\varepsilon^2)$ w.r.t. the capacity of edges cut.

The main observation for this algorithm is that tree sets that are smaller than $\frac{\varepsilon n}{3k}$ or larger than $(1+\varepsilon)\frac{n}{k}$ do not have to be considered. The first type of set cannot be contained in OPT_T because of the definition of a coarse T-partitioning. The second type of set cannot be contained since then OPT_T would not be feasible, because such a large set cannot be assigned to a partition without exceeding the balance constraint (the partition would have size larger than $(1+\varepsilon)\frac{n}{k}$).

In the following we remove all nodes from T that correspond to sets of size smaller than $\frac{\varepsilon n}{3k}$, and we mark all nodes that correspond to sets larger than $(1+\varepsilon)\frac{n}{k}$ as *invalid*. We can use a dynamic programming approach to compute the optimum T-partitioning OPT_T. Let for a node $v \in T'$ (where T' denotes the pruned tree), T_v denote the sub-tree rooted at r and let G_v denote the sub-graph of G induced by the set V_v (the set corresponding to node v). The optimum partitioning for $G \setminus G_v$ only depends on the sizes of sets used for partioning T_v. As described above we can assume that these set-sizes are from the interval $(\frac{\varepsilon n}{3k}, (1+\varepsilon)\frac{n}{k})$. Furthermore, since we are only interested in $(1+\varepsilon)$-feasibility we can round these sizes to powers of $(1+\varepsilon/2)$.

Let t denote the number of $(1+\varepsilon/2)$-powers in the interval $(\frac{\varepsilon n}{3k},(1+\varepsilon)\frac{n}{k})$ (i.e., $t=O(\frac{1}{\varepsilon}\log(\frac{1}{\varepsilon}))$). The powers of $(1+\varepsilon/2)$ partition this interval into $t+1$ sub-intervals. A subset whose size is in the s-th interval has rounded size

$$r_s := (1+\varepsilon/2)^{\lceil \log_{1+\varepsilon/2}\frac{\varepsilon n}{3k}\rceil}\cdot(1+\varepsilon/2)^{s-1} \ .$$

Now, let for a partitioning P of G_v, $\vec{g}(P)=(g_1,\ldots,g_{t+1})$ denote a size-vector in which the s-th entry g_s denotes the number of sets for rounded size r_s that are used in the partitioning P. We call $\vec{g}(P)$ the size-vector of the partitioning P. We define $\mathrm{OPT}_{T_v}(\vec{g}_{\max})$ as the cost of an optimum partitioning of the sub-graph G_v such that the size-vector of this partitioning is dominated by $\vec{g}_{\max}$. Formally

$$\mathrm{OPT}_{T_v}(\vec{g}_{\max}) = \begin{cases} \infty & \text{if } \forall P\ \vec{g}(P) > \vec{g}_{\max} \\ \min\limits_{\substack{T\text{-part. } P \text{ for } T_v\\ \vec{g}(P)\le \vec{g}_{\max}}} \{\mathrm{cost}(P)\} & \text{otherwise} \end{cases}$$

The cost OPT_T of the optimum $(1+\varepsilon)$-feasible coarse T-partitioning for G is then

$$\mathrm{OPT}_T = \min\{\mathrm{OPT}_T(\vec{g}) \mid \vec{g} \text{ is feasible}\} \ , \tag{1}$$

where we say that a size-vector $\vec{g}=(g_1,\ldots,g_{t+1})$ is feasible if it is possible to pack g_s items of size r_s, $s\in\{1,\ldots,t+1\}$ into k bins of size $(1+\varepsilon)\frac{n}{k}$.

We can compute the value $\mathrm{OPT}_{T_v}(\vec{g}_{\max})$ for every sub-tree and for every possible vector $\vec{g}_{\max}$ via dynamic programming in the following way. Fix a tree node v. We define $r(v)\in\{1,\ldots,t+1\}\cup\{\mathsf{invalid}\}$ to be the rounded size of the set V_v corresponding to v (if $|V_v|\in(\frac{\varepsilon n}{3k},(1+\varepsilon)\frac{n}{k})$) or to be $\mathsf{invalid}$. Furthermore we use $\mathrm{out}(V_v)$ to denote the number of edges leaving V_v, and we use v_l and v_r to denote the children of v in the pruned tree T' (if v is not a leaf node). The dynamic programming recurrence for $\mathrm{OPT}_{T_v}(\vec{g}_{\max})$ is as follows.

$$\mathrm{OPT}_{T_v}(\vec{g}_{\max}) = \begin{cases} \infty & \text{if } \vec{g}_{\max}=(0,\ldots,0) \\ \mathrm{out}(V_v) & \text{if } r_s \neq \mathsf{invalid} \text{ and } e_{r_s}\le \vec{g}_{\max} \\ \min\limits_{\substack{\vec{g}_{\mathrm{l}},\vec{g}_{\mathrm{r}}\\ \vec{g}_{\mathrm{l}}+\vec{g}_{\mathrm{r}}=\vec{g}_{\max}}} \{\mathrm{OPT}_{T_{v_{\mathrm{l}}}}(\vec{g}_{\mathrm{l}})+\mathrm{OPT}_{T_{v_{\mathrm{r}}}}(\vec{g}_{\mathrm{r}})\} & \text{otherwise} \end{cases}$$

3.4 Running Time

The running time for computing the value OPT_T with the above recurrence is as follows. We start with a bound on the number of entries in the dynamic programming table for $\mathrm{OPT}_{T_v}(\vec{g}_{\max})$. Firstly, there are $O(n)$ nodes in the tree (i.e., $O(n)$ different values for v). Secondly, the overall number of pieces is bounded by $\frac{1+\varepsilon/2}{\varepsilon/3}k\le 3(1+\varepsilon)k/\varepsilon$ because each set has size at least $\frac{\varepsilon n}{3k}$ and the total rounded size of all sets cannot exceed $(1+\varepsilon/2)n$. Therefore, each entry in a vector $\vec{g}_{\max}$ is bounded by $3(1+\varepsilon)k/\varepsilon$, which gives a bound of $O((3(1+\varepsilon)k/\varepsilon)^{t+1})$ on the number of different vectors to consider. Overall the total number of entries is $O(n\cdot(3(1+\varepsilon)k/\varepsilon)^{t+1})$.

For each entry there are at most $(3(1+\varepsilon)k/\varepsilon)^{t+1}$ possibilities to choose the vectors $\vec{g}_{\mathrm{l}}$ and $\vec{g}_{\mathrm{r}}$. Hence, the running time for computing one entry is $O((3(1+\varepsilon)k/\varepsilon)^{t+1})$. The running time for all entries is $O(n\cdot(3(1+\varepsilon)k/\varepsilon)^{2(t+1)})$.

After computing all entries in the table the algorithm has to compute the minimum in Equation 1. For this the algorithm has to compute whether a vector $\vec{g}$ is feasible. This is the well known bin packing problem which, in general, is NP-complete but can be solved in polynomial time for our case because we only have items of a constant number of different sizes due to the rounding. Solving such a bin packing problem needs time $O(p^{2s})$ (see [HS88]), where p denotes the number of pieces and s denotes the number of different sizes. In our case the number of pieces is bounded by $3(1+\varepsilon)k/\varepsilon$ and the number of different sizes is $t+1$. This gives a running time of $O((3(1+\varepsilon)k/\varepsilon)^{2(t+1)})$ for checking the feasibility of a vector $\vec{g}$. Doing this for all vectors results in time $O((3(1+\varepsilon)k/\varepsilon)^{3(t+1)})$ for evaluating the minimum in Equation 1. The total running time for the dynamic programming is bounded by $O(n\cdot(3(1+\varepsilon)k/\varepsilon)^{3(t+1)}) = O(n\cdot(k/\varepsilon)^{O(\frac{1}{\varepsilon}\log(\frac{1}{\varepsilon}))})$.

The overall running time is the maximum of the $(\varepsilon, 1-\varepsilon)$-separation algorithm, which is polynomial time ($O(\max\{\log n, \log\frac{1}{\varepsilon}\}\cdot(n\frac{1}{\varepsilon^2}+n^{4.5}))$) and will use semidefinite programming as a subroutine, and the dynamic programming algorithm $O(n\cdot(k/\varepsilon)^{O(\frac{1}{\varepsilon}\log(\frac{1}{\varepsilon}))})$.

The following theorem summarizes the algorithmic result from this section

Theorem 6 *For the $(k, 1+\varepsilon)$-balanced graph partitioning problem the above algorithm finds a partition that cuts at most $O(\log^{1.5} n/\varepsilon^2)\cdot\mathrm{cost}(\mathrm{OPT}_k)$ edges, where OPT_k denotes the optimum (exact) $(k,1)$-balanced partitioning and runs in time polynomial in n and k, but exponential in $1/\varepsilon$.*

4 Conclusions

We presented a polynomial time approximation algorithm for the (k,ν)-balanced partitioning problem that achieves an $O(\log^{1.5} n)$-approximation ratio w.r.t. the capacity of edges between different partitions. The algorithm extends in a straightforward manner to the case where the nodes of the graphs are weighted and the goal is to balance the weight among the partitions (this works if the node weights are polynomially bounded).

It seems a challenging task to improve the running time of the algorithm so that the dependence on $\frac{1}{\varepsilon}$ is reduced. Another interesting problem is to generalize the problem to the case where the different partitions are required to have different size. This could be used to model parallel scheduling on different machines, by making the partitions for fast processors larger. Finally, it is worth mentioning that the current hardness results do not rule out an approximation algorithm for which the approximation factor does not depend on $1/\varepsilon$ and only the running time depends on epsilon. Therefore it is an interesting open problem whether the dependency on $1/\varepsilon$ in the approximation ratio can be reduced or completely removed.

References

[AKK95] Sanjeev Arora, David R. Karger, and Marek Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. In *Proceedings of the 27th ACM Symposium on Theory of Computing (STOC)*, pages 284–293, 1995.

[ARV04] Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings, and graph partitionings. In *Proceedings of the 36th ACM Symposium on Theory of Computing (STOC)*, pages 222–231, 2004.

[ENRS99] Guy Even, Joseph (Seffi) Naor, Satish B. Rao, and Baruch Schieber. Fast approximate graph partitioning algorithms. *SIAM Journal on Computing*, 28(6):2187–2214, 1999. Also in *Proc. 8th SODA*, 1997, pp. 639–648.

[ENRS00] Guy Even, Joseph (Seffi) Naor, Satish Rao, and Baruch Schieber. Divide-and-conquer approximation algorithms via spreading metrics. *Journal of the ACM*, 47(4):585–616, 2000. Also in *Proc. 36th FOCS*, 1995, pp. 62–71.

[FK02] Uriel Feige and Robert Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM Journal on Computing*, 31(4):1090–1118, 2002. Also in *Proc. 32nd STOC*, 2000, pp. 530–536.

[FKN00] Uriel Feige, Robert Krauthgamer, and Kobbi Nissim. Approximating the minimum bisection size. In *Proceedings of the 32nd ACM Symposium on Theory of Computing (STOC)*, pages 530–536, 2000.

[FM82] Charles M. Fiduccia and Robert M. Mattheyses. A linear time heuristic for improving network partitions. In *Proceedings of the 19th IEEE Design Automation Conference*, pages 175–181, 1982.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Co., San Francisco, Calif., 1979.

[HS88] Dorit S. Hochbaum and David B. Shmoys. A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. *SIAM Journal on Computing*, 17(3):539–551, 1988.

[KL70] Brian W. Kernigham and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–308, 1970.

[LMT90] Frank Thomson Leighton, Fillia Makedon, and Spyros Tragoudas. Approximation algorithms for VLSI partition problems. In *Proceedings of the IEEE International Symposium on Circuits and Systems, (ISCAS)*, pages 2865–2868, 1990.

[LR99] Frank Thomson Leighton and Satish B. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, 1999.

[LT79] Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[RH01] Arnold L. Rosenberg and Lenwood S. Heath. *Graph separators, with applications*. Kluwer Academic Publishers, 2001.

[ST97] Aravind Srinivasan and Chung-Piaw Teo. A constant-factor approximation algorithm for packet routing and balancing local vs. global criteria. In *Proceedings of the 29th ACM Symposium on Theory of Computing (STOC)*, pages 636–643, 1997.

[SV95] Huzur Saran and Vijay V. Vazirani. Finding k-cuts within twice the optimal. *SIAM Journal on Computing*, 24(1):101–108, 1995.