Balanced Graph Partitioning

Konstantin Andreev* Harald Räcke †

ABSTRACT

In this paper we consider the problem of (k, ν)-balanced graph partitioning - dividing the vertices of a graph into k almost equal size components (each of size less than $\nu \cdot \frac{n}{k}$) so that the capacity of edges between different components is minimized. This problem is a natural generalization of several other problems such as minimum bisection, which is the $(2, 1)$-balanced partitioning problem. We present a bicriteria polynomial time approximation algorithm with an $O(\log^2 n)$-approximation for any constant $\nu > 1$. For $\nu = 1$ we show that no polytime approximation algorithm can guarantee a finite approximation ratio unless $P = NP$. Previous work has only considered the (k, ν)-balanced partitioning problem for $\nu \geq 2$.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Algorithms, Theory

Keywords

Graph Partitioning, Approximation Algorithms, Bicriteria Approximation

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, circuit testing, and sparse linear system solving.

The graph partitioning problem considered above 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 [9, 6]) and approximation algorithms (see [4, 5]).

If k is not constant the problem is much harder. 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 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 approach is sometimes called bicriteria-approximation, resource-augmentation, or pseudo-approximation in the literature.

To the best of our knowledge the 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. [2].

In this work we consider the $(k, 1 + \epsilon)$-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 $\epsilon = 0$. The latter result is in some sense the justification for looking at bicriteria approximation algorithms instead of "real" approximation algorithms.

1.1 Related Work

The first heuristics for minimum bisection were given by Kernighan and Lin [9] and subsequently improved in terms of running time by Fiduccia and Mattheyses [6].

The first non-trivial approximation algorithm for this problem is due to Saran and Vazirani [12] who obtained an approximation ratio of $n/2$. Subsequently the ratio was im-

*Mathematics Department, Carnegie-Mellon University, Pittsburgh PA 15213. Email: konst@cmu.edu

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

SPAA'04, June 27–30, 2004, Barcelona, Spain.

proved by Feige et al. [4] to $O(\sqrt{n}\cdot\text{polylog}(n))$. In their seminal paper [5] Feige and Krauthgamer were able to improve the ratio to the currently best known bound of $O(\log^2 n)$.

For some special graph classes better bounds are known. For planar graphs a bound of $O(\log n)$ is given in [5], and for graphs of minimum degree $\Omega(n)$, Arora et al. [1] presented a PTAS.

For general k some of the above algorithms can be extended. 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. Note however that these algorithms require a running time that is exponential in k.

The relaxed problem version was considered by Leighton et al. [10] and Simon and Teng [13]. 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. [3] have used the spreading metrics technique to improve this to $O(\log n \log\log n)$ and later ([2]) 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.

We do not think that the above solutions for the $(k,2)$-balanced partitioning problem can be easily extended to $(k,1+\epsilon)$-balanced partitioning problems with $\epsilon < 1$. 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 small pieces as well. From this the authors deduce 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.

Nevertheless 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. A partitioning of G into ℓ parts is a collection of ℓ disjoint subsets $V_1,\ldots,V_\ell$ that cover V, i.e., $V = V_1 \dot\cup \cdots \dot\cup V_\ell$. For a partitioning P we use $\text{cost}(P)$ to denote the capacity of edges *cut* by the partitioning, i.e., edges between different sets V_i.

We investigate the $(k,1+\epsilon)$-balanced partitioning problem, that is the problem of finding a minimum cost partitioning of G into k parts such that each part contains at most $(1+\epsilon)\frac{n}{k}$ 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 [2] show that an instance with $\epsilon > 1$ can be reduced to a problem instance with $\epsilon \leq 1$.

2. HARDNESS RESULTS FOR BALANCED PARTITIONING

It is well known already that the $(2,1)$-partitioning problem, in other words the minimum bisection problem, is NP-complete (see [7]). For this problem efficient approximation algorithms have been recently obtained in [5]. The following theorem shows that it is not possible to derive even 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, that is algorithms that are allowed to violate the balance constraint of the partitions.

THEOREM 2.1. *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 A such that $\frac{A}{4} < a_i < \frac{A}{2}$ and

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

The task is to decide if the numbers can be partitioned into triples such that each triple adds up to A. This problem is *strongly* NP-complete (see [7]). This means it is already NP-complete in the case that all numbers (a_i, A) are polynomially bounded.

Suppose that we have an approximation algorithm for $(k,1)$-balanced partition with finite approximation factor. We can use this algorithm to solve an instance of 3-Partition with polynomially bounded numbers, in the following manner.

We construct a graph G such that for each number a_i it contains a clique of size a_i. 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. □

Note that the above proof is only valid since the 3-Partition problem is strongly NP-complete, which means that the problem is still difficult if all numbers are bounded by some polynomial in the length of the input. (Otherwise the construction of the graph in the above proof would not be polynomial time.)

3. ALGORITHM AND ANALYSIS

In this section we prove the following theorem.

THEOREM 3.1. *There is a polynomial time algorithm for partitioning a graph $G = (V, E)$ into k disjoint components $V_1, \ldots, V_k \subset V$ such that each component contains at most $(1+\epsilon)\frac{n}{k}$ nodes, and cuts at most $O(\log^2 n/\epsilon^4) \cdot \mathrm{OPT}_k$.*

Our algorithm for solving the $(k, 1+\epsilon)$-balanced partitioning problem proceeds in two phases. In the first 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 directed 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. Let's set

$$\epsilon' := \frac{\epsilon/3}{1+\epsilon/3}.$$

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 $\epsilon' \cdot |V|$ 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 $\epsilon'/2$-balanced cut, i.e. every part contains at least $\epsilon'/2 \cdot |V|$ nodes, and that cuts a capacity of at most

$$O(S \cdot \log n/\epsilon^3),$$

where S denotes the total capacity of edges in an optimum ϵ'-balanced cut. This can be done in time $\tilde{O}(n^2/\epsilon)$ (see [11]). This separation algorithm gives a decomposition of height $O(\log n/\epsilon)$, since in each divide step the larger set is reduced by a factor of at least $(1-\epsilon'/2)$.

3.1 Coarse T-partitionings

To solve the $(k, 1+\epsilon)$-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 1. *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 $\frac{\epsilon n}{3k}$ nodes.*

Our algorithm computes a coarse T-partitioning P and then tries to transform such a partitioning into a $(k, 1+\epsilon)$-balanced partition by merging some of the subsets of P. However, the latter step is not always possible. Therefore we introduce a definition that captures whether a given T-partitioning P can be transformed into a $(k, 1+\epsilon)$-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 $(1+\epsilon)$*-feasible* if J can be partitioned into k distinct index sets $J_1, \ldots, J_k$ such that

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

where $(1+\epsilon/2)^{\lceil \log_{1+\epsilon/2} |V_j| \rceil}$ means that $|V_j|$ is rounded to the next power of $(1+\epsilon/2)$.

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

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

LEMMA 3.2. *There is a $(1+\epsilon)$-feasible coarse T-partitioning that cuts at most $O(\log^2 n/\epsilon^4) \cdot \mathrm{cost}(\mathrm{OPT}_k)$ 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. We successively transform OPT_k into a coarse T-partitioning that is $(1+\epsilon)$-feasible.

First we create a new partitioning OPT'_K that only contains subsets of T. This is done in the following manner. Initially OPT'_K is 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 our partitioning OPT'_k. *Almost monochromatically* means that at least a $\frac{1}{1+\epsilon/3}$-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 $\frac{1}{1+\epsilon/3}|V_x|$ 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 $\epsilon \leq 1$). Note that the sets that are taken by this algorithm are disjoint, i.e., OPT'_k is a partitioning of G in the end.

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 $\frac{\epsilon n}{3k}$ 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 $(1+\epsilon)$-feasible and that $\mathrm{cost}(\mathrm{OPT}''_k) \leq O(\log^2 n) \cdot \mathrm{cost}(\mathrm{OPT}_k)$.

CLAIM 3.3. OPT''_k *is* $(1+\epsilon)$*-feasible.*

PROOF. We first show that OPT'_k is $(1+\epsilon)$-feasible. We do this by merging the sets to a $(k, 1+\epsilon)$-balanced partition in the following way. Merge all sets of OPT'_k that have the same majority color into one partition. This gives k partitions. We have to show that no partition is too large with respect to the definition of $(1+\epsilon)$-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

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

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

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 $(1+\epsilon)\frac{n}{k}$ nodes. Again, rounding only increases this term by a factor of $(1+\epsilon/2)$, which proofs the claim. □

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

CLAIM 3.4. $\mathrm{cost}(\mathrm{OPT}''_k) \leq O(\log^2 n/\epsilon^4)\cdot \mathrm{cost}(\mathrm{OPT}_k)$.

PROOF. We will show that

$$\mathrm{cost}(\mathrm{OPT}'_k) \leq O(\log^2 n/\epsilon^4)\cdot \mathrm{cost}(\mathrm{OPT}_k) \ .$$

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 $\frac{1}{1+\epsilon/3}\cdot|V_{v_a}| = (1-\epsilon')|V_{v_a}|$ nodes because V_{v_a} is not colored almost monochromatically. We merge partitions of V_{v_a} into a set M in decreasing order of there size until M contains more than $\epsilon'\cdot|V_{v_a}|$ nodes. Since $\epsilon' < 1/3$ we have that $|M| \leq (1-\epsilon')|V_{v_a}|$. 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

$$\mathrm{cost}_{V_{v_a}}(\mathrm{OPT}'_k) \leq O(\log(n)/\epsilon^3)\cdot \mathrm{cost}_{V_{v_a}}(\mathrm{OPT}_k) \ ,$$

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 height(T) times for amortization we get that

$$\mathrm{cost}(\mathrm{OPT}'_k) \leq \mathrm{height}(T)\cdot O(\log n/\epsilon^3)\cdot \mathrm{cost}(\mathrm{OPT}_k) \\ \leq O(\log^2 n/\epsilon^4)\cdot \mathrm{cost}(\mathrm{OPT}_k) \ .$$

This completes the proof of the claim. □

Claims 3.3 and 3.4 give the lemma.

3.2 The algorithm

In this section we show how to find an optimal $(1+\epsilon)$-feasible coarse T-partitioning OPT_T. From Lemma 3.2 it follows that this partitioning gives a $(k, 1+\epsilon)$-partitioning with an approximation factor of $O(\log^2 n/\epsilon^4)$ w.r.t. the capacity of edges cut.

The main observation for this algorithm is that tree sets that are smaller than $\frac{\epsilon n}{3k}$ or larger than $(1+\epsilon)\frac{n}{k}$ 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 constrained (the partitition would have size larger than $(1+\epsilon)\frac{n}{k}$).

By removing all nodes from T that correspond to one of the above sets, we partition T into many small sub-trees $T_1, \ldots, T_\ell$. For each tree T_i we have

$$\mathrm{height}(T_i) \leq \log(\frac{1+\epsilon}{\epsilon/3})/\log(\frac{1}{1+\epsilon'}) = O(\frac{1}{\epsilon}\log(\frac{1}{\epsilon})) \ .$$

Furthermore, each tree contains at most $2^{O(1/\epsilon\log(1/\epsilon))}$ nodes. Note that both of these values are constant.

We can use a dynamic programming approach on these trees to find OPT_T in polynomial time, as follows. Suppose that the partitioning for the first sub-trees $T_1, \ldots, T_{i-1}$ is fixed. Then the optimum partitioning for sub-trees $T_i, \ldots, T_\ell$ only depends on the size of the sets that are chosen in sub-trees $T_1, \ldots, T_{i-1}$. Since we only check for $(1+\epsilon)$-feasibility of a partitioning we can round these sizes to powers of $(1+\epsilon/2)$.

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

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

Let $\mathrm{OPT}_T(T_i, \ldots, T_\ell | g_1, \ldots, g_{t+1})$ denote the cost of an optimal T-placement for sub-trees $T_i, \ldots, T_\ell$ if already g_s sets of rounded size r_s are used in the partitioining for $T_1, \ldots, T_{i-1}$. We want to derive a dynamic programming recurrence for this term. For this we need some notation. We say that $g_1, \ldots, g_{t+1}$ 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 $(1+\epsilon)\frac{n}{k}$.

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 [8]), 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 $\frac{1+\epsilon/2}{\epsilon/3}k = O(k/\epsilon)$ because each set has size at least $\frac{\epsilon n}{3k}$ and the total rounded size of all sets cannot exceed $(1+\epsilon/2)n$. The number of different sizes is $t+1$ which gives a running time of $O((\frac{k}{\epsilon})^{t+1})$.

Let for a T-partitioning P of a sub-tree, $x_s(P)$ denote the number of sets with rounded size r_s that are contained in P. The recurrence for $\mathrm{OPT}_T(T_i,\ldots,T_\ell|g_1,\ldots,g_{t+1})$ is as follows.

$$\mathrm{OPT}_T(T_i,\ldots,T_\ell|\vec{g}) = \min_{\substack{T\text{-part. } P \\ \text{for } T_i}} \{\mathrm{cost}(P) + \mathrm{OPT}_T(T_{i+1},\ldots,T_\ell|\vec{g})\} \ ,$$

if $g_1,\ldots,g_{t+1}$ is feasible. (Here we used the short-hand notation $\vec{g}$ to denote $g_1,\ldots,g_{t+1}$.) In the case that $g_1,\ldots,g_{t+1}$ is infeasible we define

$$\mathrm{OPT}_T(T_i,\ldots,T_\ell|g_1,\ldots,g_{t+1}) = \infty \ .$$

3.3 Running Time

Let's first calculate the running time of the dynamic programming. What time is needed for computing OPT_T with the above recurrence? The dynamic programming table has $O(\ell\cdot(k/\epsilon)^{t+1}) = O(n\cdot(k/\epsilon)^{t+1})$ entries because $g_i = O(k/\epsilon)$; otherwise the rounded size of sets used in $T_1,\ldots,T_{i-1}$ were larger than $(1+\epsilon/2)\cdot n$, and $\ell \le n$. For each entry the algorithm needs time $O((k/\epsilon)^{t+1})$ to decide whether $g_1,\ldots,g_{t+1}$ is feasible. To compute the minimum the algorithm can iterate over all partitionings of subtree T_i. There are at most $O(2^{2^{O(1/\epsilon\cdot\log(1/\epsilon))}})$ such partionings since there are only $2^{O(1/\epsilon\cdot\log(1/\epsilon))}$ nodes in a subtree. On the whole the dynamic programming algorithm has a running time of at most $O(n\cdot(k/\epsilon)^{O(\log 1/\epsilon)+1}\cdot 2^{2^{O(1/\epsilon\cdot\log(1/\epsilon))}})$ which is polynomial in n for constant ϵ.

The overall running time is the maximum of the separation algorithm $\tilde{O}(n^2/\epsilon)$ and the dynamic programming algorithm $O(n\cdot(k/\epsilon)^{O(\log 1/\epsilon)}\cdot 2^{2^{O(1/\epsilon\cdot\log(1/\epsilon))}})$. For example if $k = n/2$ and $\epsilon = 1/2$ then the overall running time will be $\tilde{O}(n^2)$.

4. CONCLUSIONS

We presented a polynomial time approximation algorithm for the (k,ν)-balanced partitioning problem that achieves an $O(\log^2 n)$-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 such that the dependence on $\frac{1}{\epsilon}$ is not that heavy. 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.

5. REFERENCES

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

[2] Even, Naor, Rao, and Schieber. Fast approximate graph partitioning algorithms. *SIAM Journal of Computing*, 28(6):2187–2214, 1999.

[3] Guy Even, Joseph Naor, Satish Rao, and Baruch Schieber. Divide-and-conquer approximation algorithms using spreading metrics. *Journal of the ACM(JACM)*, 47(4):585–616, 2000.

[4] U. Feige, R. Krauthgamer, and K. Nissim. Approximating the minimum bisection size. In *32nd Annual ACM Symposium on the Theory of Computing*, pages 530–536, 2000.

[5] Uriel Feige and Robert Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM Journal on Computing*, 31(4):1090–1118, 2002.

[6] C. M. Fiduccia and R. M. Mattheyses. A linear time heuristic for improving network partitions. In *19th IEEE Design Automation Conference*, pages 175–181, 1982.

[7] M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. W.H. Freeman and Co, San Francisco, CA, 1979.

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

[9] B.W. Kernighan and S. Lin. An efficient heuristic for partitioning graphs. *Bell Sys. Tech. Journal*, 49:291–308, 1970.

[10] T. Leighton, F. Makedon, and S. Tragoudas. Approximation algorithms for vlsi partition problems. In *IEEE International Symposium on Circuits and Systems, (ISCAS '90)*, volume 4, pages 2865–2868, 1990.

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

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

[13] Horst D. Simon and Shang-Hua Teng. How good is recursive bisection? *SIAM Journal on Scientific Computing*, 18(5):1436–1445, 1997.