Multiply Balanced $k-$ Partitioning

Amihood Amir[1,2,⋆], Jessica Ficler[1], Robert Krauthgamer[3⋆⋆], Liam Roditty[1], and Oren Sar Shalom[1]

[1] Department of Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel.
E-mail: amir@cs.biu.ac.il | jessica.ficler@gmail.com | liamr@cs.biu.ac.il | oren.sarshalom@gmail.com
[2] Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218.
[3] Weizmann Institute of Science, Rehovot, Israel. E-mail: robert.krauthgamer@weizmann.ac.il

Abstract. The problem of partitioning an edge-capacitated graph on n vertices into k balanced parts has been amply researched. Motivated by applications such as load balancing in distributed systems and market segmentation in social networks, we propose a new variant of the problem, called Multiply Balanced k Partitioning, where the vertex-partition must be balanced under d vertex-weight functions simultaneously.

We design bicriteria approximation algorithms for this problem, i.e., they partition the vertices into up to k parts that are nearly balanced simultaneously for all weight functions, and their approximation factor for the capacity of cut edges matches the bounds known for a single weight function times d. For the case where $d = 2$, for vertex weights that are integers bounded by a polynomial in n and any fixed $\epsilon > 0$, we obtain a $(2+\epsilon,\ O(\sqrt{\log n \log k}))$-bicriteria approximation, namely, we partition the graph into parts whose weight is at most $2+\epsilon$ times that of a perfectly balanced part (simultaneously for both weight functions), and whose cut capacity is $O(\sqrt{\log n \log k}) \cdot \mathrm{OPT}$. For unbounded (exponential) vertex weights, we achieve approximation $(3,\ O(\log n))$.

Our algorithm generalizes to d weight functions as follows: For vertex weights that are integers bounded by a polynomial in n and any fixed $\epsilon > 0$, we obtain a $(2d+\epsilon,\ O(d\sqrt{\log n \log k}))$-bicriteria approximation. For unbounded (exponential) vertex weights, we achieve approximation $(2d+1,\ O(d \log n))$.

1 Introduction

In the k-BALANCED PARTITIONING problem (aka MINIMUM k-PARTITIONING) the input is an edge-capacitated graph and an integer k, and the goal is to partition the graph vertices into k parts of equal size, so as to minimize the total capacity of the cut edges (edges connecting vertices in different parts). The problem has

⋆ Partly supported by NSF grant CCR-09-04581, ISF grant 347/09, and BSF grant 2008217.

⋆⋆ Work supported in part by a US-Israel BSF grant #2010418, ISF grant 897/13, and by the Citi Foundation.

many applications, ranging from parallel computing and VLSI design to social networks, as we discuss further below. The above problem is known to be NP-hard. Even the special case where $k = 2$ (called minimum bisection) is already NP-hard [7] and several approximation algorithms were designed [13,5,2,17]. For constant k the polynomial-time algorithm of MacGregor [14] can solve the problem on trees. However, if k is not constant the problem is hard to approximate within any finite factor [1]. Several heuristics were proposed, see e.g. [10,16,15,9] but they do not guarantee any upper bounds on the cut capacity. It is therefore common to consider a bicriteria approximation, which relaxes the balance constraint.

Formally, let $G = (V, E)$ be a graph of n vertices. In a (k, ν)-*balanced partition*, the vertex set V is partitioned into at most k parts, each of size at most $\nu n/k$, and the cut capacity is compared against an optimal (minimum cut capacity) perfectly balanced k-partition [12,18,4,1,11,6]. In the *weighted* version, every vertex $v \in V$ has a weight $w(v) \geq 0$, and now in a (k, ν)-balanced partitioning, there are at most k parts, and the total weight of every part is at most $\nu W/k$, where W is the total weight of all the vertices. Let us emphasize that we always consider graphs with edge capacities; the terms weighted or unweighted graphs refer only to vertex weights. Throughout, we assume that there exists a perfectly balanced k-partition, e.g., in the unweighted version this means that k divides n.

Definition 1. *An algorithm for* k-BALANCED PARTITIONING *is said to give a* (ν, α)-bicriteria approximation *if it finds a* (k, ν)*-balanced partition whose cut capacity is at most* $\alpha\,\mathrm{OPT}$*, where* OPT *is the cut capacity of an optimal perfectly balanced* k*-partition.*

The k-BALANCED PARTITIONING problem has numerous applications. Specifically, in parallel computing, each vertex typically represents a task, its weight represents the amount of processing time needed for that task, and edges represent the communication costs. In this example, k is the number of available processors. However, this formulation does not support the case where we want to distribute the load by two parameters, for example processing time and memory. A similar unsolved problem arises in social networks and marketing: vertices represent people, edges are the strength of the relationship between two people, and each person has a value (potential revenue) for a marketing campaign. The goal is to partition the people into k groups, such that there will be the least connection between the groups, and the groups are balanced both by their size and by their total marketing value.

Definition 2. *In the* DOUBLY BALANCED k-PARTITIONING *problem, the input is a graph* $G = (V, E, w_1, w_2, c)$ *and an integer* k*, where* $w_1, w_2 : V \to \mathbb{R}_{\geq 0}$ *are the vertex-weight functions and* $c : E \to \mathbb{R}_{\geq 0}$ *is the edge capacity function. The goal is to find a partition of the graph into at most* k *parts that are balanced by both weight functions, so as to minimize the total capacity of the cut between the different parts.*

We emphasize that k-BALANCED PARTITIONING refers to the case where there is a single vertex-weight function, in contrast to DOUBLY BALANCED k-PARTITIONING. This is true even when one of the two vertex-weight functions above is constant (aka uniform weights), which means balancing with respect to the sizes (cardinalities) of the parts.

The problem can be generalized to d vertex-weight functions:

Definition 3. *In the* MULTIPLY BALANCED k PARTITIONING *problem, the input is a graph* $G = (V, E, w_1, w_2, ..., w_d, c)$ *and an integer* k*, where* $w_1, ..., w_d : V \to \mathbb{R}_{\geq 0}$ *are the vertex-weight functions and* $c : E \to \mathbb{R}_{\geq 0}$ *is the edge capacity function. The goal is to find a partition of the graph into at most* k *parts that are balanced by all* d *weight functions, so as to minimize the total capacity of the cut between the different parts.*

2 Bicriteria Approximations and Our Results

The DOUBLY BALANCED k-PARTITIONING problem is hard to approximate within any finite factor, simply because setting $w_2(v) = 0$ (or $w_2(v) = w_1(v)$) for all $v \in V$ yields the k-BALANCED PARTITIONING problem as a special case. We therefore aim at a bicriteria approximation for the problem. Throughout, for $S \subseteq V$, $1 \leq j \leq 2$, and a vertex-weight function w, we define $w(S) := \sum_{v \in S} w_j(v)$, and let $W := w(V)$ denote the total weight of all the vertices.

Definition 4. *A partition* $\{P_i\}$ *of* V *is called* (k, ν)-doubly balanced *if it has at most* k *parts, and for each part* P_i *and each* $j = 1, 2$ *it hold that* $w_j(P_i) \leq \nu\, w_j(V)/k$.

Before defining the precise guarantees of our algorithm, we need to understand the criteria that we are trying to approximate. In the unweighted version, balanced partition asserts that every part is of size at most $\lceil \frac{n}{k} \rceil$, which guarantees that there always exists a perfectly balanced partition. In the weighted version, this might not be possible at all. For example, if the graph has a single vertex whose weight exceeds that of all other vertices together, then obviously there is no perfectly balanced partition. Therefore, in all existing algorithms there is an implicit assumption that there exists a perfectly balanced partition (which we are trying to approximate).

In DOUBLY BALANCED k-PARTITIONING, we could assume the existence of a perfectly doubly balanced partition as well. However, this might be an unreasonable assumption in many applications, and thus we weaken the requirement — we only assume that there is a perfectly balanced partition for each weight separately, but not necessarily together.

Definition 5. *A* (ν, α)-bicriteria approximation *for the* DOUBLY BALANCED k-PARTITIONING *problem finds a* (k, ν)*-doubly balanced partition, whose cut capacity is at most* $\alpha\,\mathrm{OPT}$*, where* OPT *is the maximum of the two optimal* $(k, 1)$*-balanced partitions.*

Throughout, the term OPT in the context of Doubly Balanced k-Partitioning refers to the above value. Notice that the cut capacity of a perfectly k-doubly balanced partition (if it exists) might be substantially larger than each of the k-balanced partitions. Nevertheless, because we relax the balance constraints, we require our algorithm to return a near k-doubly balanced partition whose cut capacity is comparable to the larger of the two different partitions.

Definition 6. *A partition $\{P_i\}$ of the vertices is called* (k^+, ν)-balanced *if $w(P_i) \leq \nu W/k$ for every part P_i.*

Notice that in a (k^+, ν)-balanced partition, unlike a (k, ν)-balanced partition, there can be more than k parts.

Definition 7. *An algorithm for* k-BALANCED PARTITIONING *is said to give a* $(\nu, \alpha)^+$-bicriteria approximation *if it finds a (k^+, ν)-balanced partition whose cut capacity is at most $\alpha\,\mathrm{OPT}$, where* OPT *is the cut capacity of an optimal perfectly balanced k-partition.*

Notice that because every (k, ν)-balanced partition is also a (k^+, ν)-balanced partition, then every (ν, α)-bicriteria approximation algorithm is also a $(\nu, \alpha)^+$-bicriteria approximation.

2.1 Our Results

Theorem 1. *The Multiply Balanced k Partitioning problem admits a polynomial-time (ν, α)-bicriteria approximation, according to the following table:*

vertex-weight functions	$\boldsymbol{\nu}$	$\boldsymbol{\alpha}$
polynomial	$2d + \epsilon$	$O(d\sqrt{\log n \log k})$
arbitrary (exponential)	$2d + 1$	$O(d \log n)$

In particular, for the Doubly Balanced k-Partitioning problem, where $d = 2$ we have:

vertex-weight functions	$\boldsymbol{\nu}$	$\boldsymbol{\alpha}$
polynomial	$2 + \epsilon$	$O(\sqrt{\log n \log k})$
arbitrary (exponential)	3	$O(\log n)$

When we say that the weights are polynomial we mean that the length necessary to encode each weight is poly-logarithmic in n.

We now provide a high-level overview of the algorithm for the Doubly Balanced k-Partitioning problem. The full details are presented in Section 3. Let A be a $(\nu, \alpha)^+$-bicriteria approximation algorithm for the k-BALANCED PARTITIONING problem.

1. *Partition Stage*: Divide the vertices into some number of parts t, with cut capacity at most $\alpha\,\mathrm{OPT}$, and the respective weight of each part is bounded by $\nu W_1/k$ and $\nu W_2/k$ (simultaneously). If $t \leq k$ then the balance requirements are met. Otherwise proceed to stage 2.

2. *Union Stage*: Combine these t parts into k parts carefully, so that each part S has weights $w_1(S) \leq (1+\nu)W_1/k$ and $w_2(S) \leq (1+\nu)W_2/k$. This new partition meets the same approximation factor for the cut capacity, because combining parts can only decrease the capacity of the cut.

We present two different algorithms, each based on a different k-balanced partition approximation algorithm, to achieve the two bounds stated in Theorem 1. We first present the special case $d = 2$ in Section 3, and then prove its generalization to d weight functions in Section 4.

2.2 Polynomial Weights

We now show how it is possible to extend the approximation ratio for k-BALANCED PARTITIONING to hold also for weighted graphs.

Andreev and Räcke [1] showed a $(1+\epsilon, \log^{1.5} n)$ bicriteria approximation for any constant $\epsilon > 0$. Their work balances the graph with respect to the sizes of the parts, but can be extended in a straightforward manner to the case where the vertices of the graph have polynomial weights and the goal is to balance the weight among the parts.

Theorem 2. *Every (ν, α)-bicriteria approximation algorithm $\mathcal{A}$ for the k-balanced partitioning problem in unweighted graphs can be used also in (polynomially) weighted graphs with the same approximation factors.*

Proof. Will appear at the full version.

If the running time of the unweighted version algorithm is $f(n)$, where n is the number of vertices, then the modified running time would be $f(W)$, where W is the total weight. If the (integer) weights of the vertices are polynomial in n, then the algorithm runs in polynomial time as well. Since the length necessary to encode each weight is polylogarithmic in n, then it guarantees that the total weight is polynomial.

For any fixed $0 < \epsilon < 1$, Feldman and Foschini [6] presented a $(1+\epsilon, O(\log n))$ bicriteria approximation for unweighted graphs. Krauthgamer, Naor and Schwartz [11] presented a $(2, O(\sqrt{\log n \log k}))$ bicriteria approximation algorithm. Their algorithm can be considered as a $(1+\epsilon, O(\sqrt{\log n \log k}))^+$ bicriteria approximation algorithm, since during its main procedure it finds a $(k^+, 1+\epsilon)$-balanced partition. As explained above, we can modify this algorithm to support weighted graphs.

2.3 Unrestricted Weights

To our knowledge, the only algorithm that achieves a bicriteria approximation for graphs with exponential weight function is that of Even, Naor, Rao and Schieber [4]. Their algorithm uses an algorithm for the $\rho-$separator problem in order to achieve a $(2, \log n)$ bicriteria approximation with an exponential weight function.

3 Bicriteria Approximation Algorithm for $d = 2$

Let $\mathcal{A}$ be a $(\nu, \alpha)^+$-bicriteria approximation algorithm for the k-balanced partitioning problem. For convenience sake, we will normalize the weights such that for every $v \in V$ we have $w_j(v) \leftarrow w_j(v) \cdot \frac{k}{w_j(V)}$, where $j = 1, 2$. From the definition of (k, ν)-doubly balanced partition each part is of weight at most $\frac{\nu w_j(V)}{k}$, thus after the normalization each part is of weight at most $\frac{\nu w_j(V)}{k} \cdot \frac{k}{w_j(V)} = \nu$. Moreover, after the normalization $w_j(V) = k$.

The algorithm works as follows. First, partition G using algorithm $\mathcal{A}$ with respect to weight function w_1. Let $\mathcal{P} = \{P_1, P_2, \ldots, P_{\ell_1}\}$ be the resulting partition. It holds for every $P \in \mathcal{P}$, that $w_1(P) \leq \nu$. Let $\mathcal{P}^> = \{P \mid P \in \mathcal{P}\ , w_2(P) > \nu\}$. In case that $\mathcal{P}^> = \emptyset$, then if $\ell_1 \leq k$ then we have at most k parts and each part satisfies the balance condition with respect to both w_1 and w_2. Let OPT_j be the cut capacity of an optimal perfectly balanced k-partition with respect to weight function $w_j,\ j = 1, 2$. The cut capacity of partition $\mathcal{P} \leq \alpha\, \mathrm{OPT}_1 \leq \alpha\, \mathrm{OPT}$.

In case that $\mathcal{P}^> \neq \emptyset$ we partition G using algorithm $\mathcal{A}$ with respect to weight function w_2. Let $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_{\ell_2}\}$ be the resulting partition.

Fix a part $P \in \mathcal{P}^>$. Let $R_i(P) = Q_i \cap P$, where $Q_i \in \mathcal{Q}$ and $1 \leq i \leq \ell_2$. As $w_2(Q_i) \leq \nu$ it follows that $w_2(R_i(P)) \leq \nu$ for every $1 \leq i \leq \ell_2$.

Consider now the partition $\mathcal{R}$ that is composed of all the parts that are in $\mathcal{P} \setminus \mathcal{P}^>$ and the parts $R_i(P) = Q_i \cap P$, where $1 \leq i \leq \ell_2$, for every $P \in \mathcal{P}^>$. Each part of this partition has weight at most ν with respect to w_1 and to w_2. The cut capacity of this partition is at most $\alpha\, \mathrm{OPT}_1 + \alpha\, \mathrm{OPT}_2 \leq O(\alpha) \cdot \mathrm{OPT}$.

The only problem with the partition $\mathcal{R}$ is that the number of its parts might be as large at $\ell_1 \cdot \ell_2$ and this may be larger than k.

In subsection 3.1 we describe a process that takes as an input this partition and combines parts of it until it reaches a final partition with at most k parts each of weight at most $1 + \nu$. As the the final partition is obtained only by combining parts of the input partition, its cut capacity cannot exceed the cut capacity of the input partition. In subsection 3.2 we show lower bounds for the method of subsection 3.1.

3.1 Combining Partitions via Bounded Pair Scheduling

Let $\mathcal{R} = \{R_1, R_2, \ldots, R_{\ell_3}\}$. Each $R_i \in \mathcal{R}$ is represented by a pair of coordinates (x_i, y_i), where $x_i = w_1(R_i)$, $y_i = w_2(R_i)$ and $1 \leq i \leq \ell_3$. Moreover, $0 \leq x_i, y_i < \nu$ and $\sum_{i=1}^{\ell_3} x_i = \sum_{i=1}^{\ell_3} y_i = k$. In case that $\ell_3 \leq k$ then the partition has all the desired properties. Hence, we assume that $\ell_3 > k$. This problem resembles a known NP-hard problem, called Vector Scheduling with 2 dimensions. Formally:

Definition 8. (Vector Scheduling) *We are given a set J of n rational d-dimensional vectors $p_1, \ldots, p_n$ from $[0, \infty)^d$ and a number m. A valid solution is a partition of J into m sets $A_1, \ldots, A_m$. The objective is to minimize $max_{1 \leq i \leq m} \|\bar{A}_i\|_\infty$ where $\bar{A}_i = \sum_{j \in A_i} p_j$ is the sum of the vectors in A_i.*

When d is constant, [8] shows a $(d+1)$ approximation, and a later work [3] gives a PTAS for the problem.

Our problem is a special case of the VS, namely with $d = 2$. However, the existing algorithms approximate the objective with respect to the optimal solution that can be achieved for a specific instance. We need to design an approximation algorithm that bounds the maximal objective for a *family* of instances, and not just for a specific instance. The family of input instances are the vectors p_i such that $\|p_i\|_\infty < \nu$ and for all $1 \leq j \leq d$, $\sum_{i=1}^{n} p_i^j = k$, where p^j is the j'th element of vector p.

Formally, we need to solve the following problem:

Definition 9. (Bounded Pair Scheduling)

INPUT: A number k and a set $\mathcal{R}$ of $n > k$ elements, such that each element is a pair (x, y) that holds $0 \leq x, y < \nu$, and $\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i = k$.

OUTPUT: A partition of $\mathcal{R}$ into a set of k elements $R_1, \ldots, R_k$, such that for all $i = 1, \ldots, k$ and $R_i = (x^{(i)}, y^{(i)})$, it holds that $0 \leq x^{(i)} \leq 1 + \nu$ and $0 \leq y^{(i)} \leq 1 + \nu$.

The algorithm below solves the bounded pair scheduling problem.

Consider now the following sets of elements:

1. $S = \{(x, y) \mid x < 1, y < 1\}$
2. $A = \{(x, y) \mid 1 \leq x < 1 + \nu,\ 1 \leq y < 1 + \nu\}$
3. $B_x = \{(x, y) \mid 1 \leq x < \nu,\ y < 1\}$
4. $B_y = \{(x, y) \mid x < 1,\ 1 \leq y < \nu\}$
5. $C_x = \{(x, y) \mid \nu \leq x < 1 + \nu,\ y < 1\}$
6. $C_y = \{(x, y) \mid x < 1,\ \nu \leq y < 1 + \nu\}$

Elements in A are balanced, and the minimum weight in each coordinate exceeds 1, therefore, if all our elements were of type A we would be done - there are no more than k balanced elements.

The B elements are "almost" balanced and the union of every element in B_x with a element in B_y is a element in A.

The elements in C are not balanced and can not be trivially combined with any other elements. The main effort of our algorithm is dealing with these elements.

The S elements are the ones which present a difficulty since both their coordinates are not bounded below. Thus there may be a very large number of them. However, they do give us the necessary maneuverability in the combining process.

The auxiliary sets span the input set $\mathcal{R}$, and because their criteria are exclusive, every element in $\mathcal{R}$ fits to exactly one of these sets. We begin by dividing the input set to the appropriate auxiliary sets. Clearly, the C sets remain empty at this stage. As we show next, the algorithm iteratively combines elements. The meaning of combining elements R_i and R_j is creating a new element $(x_i + x_j, y_i + y_j)$ instead of them and assigning it to the appropriate set.

As long as there are two elements $R_i, R_j \in S$ such that $x_i + x_j < 1$ and $y_i + y_j < 1$ we pick such two elements R_i and R_j and combine them. At the end of this stage it is guaranteed for every $R_i, R_j \in S$ that either $x_i + x_j \geq 1$ or $y_i + y_j \geq 1$.

Next, as long as there is a pair R_i and $R_j \in S$ such that $x_i + x_j \geq 1$ and $y_i + y_j \geq 1$ then it also holds for such a pair that $x_i + x_j \leq 2 < 1 + \nu$ because $x_i < 1$ and $x_j < 1$. Similarly, $y_i + y_j \leq 2 < 1 + \nu$. We combine such a pair to R_{ij}, and add it to A.

At the end of this stage it is guaranteed that for every $R_i, R_j \in S$ either $x_i + x_j \geq 1$ and $y_i + y_j < 1$ or $y_i + y_j \geq 1$ and $x_i + x_j < 1$.

Lemma 1. *At this stage of the algorithm, for every pair $R_i, R_j \in S$ it holds that $(x_i + x_j, y_i + y_j)$ fits to one of the elements B_x, B_y, C_x and C_y.*

Proof. Will appear at the full version.

The algorithm proceeds as follows. We iteratively choose a pair $R_i, R_j \in S$ that minimizes $max\{x_i + x_j, y_i + y_j\}$. Since we choose the pair that minimizes the maximum of the two coordinates it is guaranteed that all the pairs that their combination is either in B_x or in B_y will be chosen before all the pairs that their combination is either in C_x or in C_y. As long as there is a pair whose combination belongs to B_x (or B_y), we combine it.

If we reach to a point that B_x and B_y are not empty and there is no longer a pair of elements that its combination belong to either B_x or B_y we do the following. As long as both B_x and B_y are not empty we combine an arbitrary pair $R_i \in B_x$ and $R_j \in B_y$. Notice that the combined element belongs to A because $1 \leq x_i + x_j < 1 + \nu$ as $1 \leq x_i < \nu$ and $x_j < 1$, and similarly, $1 \leq y_i + y_j < 1 + \nu$. After that, at most one of B_x and B_y is not empty. Assume that one of them is not empty, and wlog let it be B_x.

Consider the following state of the algorithm: the sets B_y, C_x and C_y are empty and the sets B_x and S are not empty. We now distinguish between two cases. The case that there is at most one $R \in S$ such that $w_1(R) < w_2(R)$ and the case that there is more than one such element in S. For the first case we prove:

Lemma 2. *If B_y and C_y are empty elements, and there is at most one $R \in S$ such that $w_1(R) < w_2(R)$, then there is a way to combine the elements of S so that the total number of different elements is at most k and every element is of weight at most $1 + \nu$.*

Proof. Will appear at the full version.

It stems from the lemma above that if we are in the case that there is at most one $R \in S$ such that $w_1(R) < w_2(R)$ then we can reach the desired partition. Thus, we assume now that there are at least two elements $R_j, R_q \in S$ such that $w_1(R_j) < w_2(R_j)$ and $w_1(R_q) < w_2(R_q)$. We choose an arbitrary element $R_i \in B_x$. We know that $x_i < \nu$, hence $x_i + x_j + x_q < 1 + \nu$. So even if we combine

R_i with R_j and R_q the x-coordinate is in the right range for A. The only question is if the y-coordinate fits. If $y_i + y_j \geq 1$, then we combine R_i and R_j as both y_i and y_j are less than 1 we can remove them and add their combination to A. If $y_i + y_j < 1$ then combine the elements R_i, R_j, R_q and add them to A, because $y_i + y_j < 1$ and $y_q < 1$ then $y_i + y_j + y_q < 2 < 1 + \nu$, and because $y_j + y_q \geq 1$ by our assumption. We continue with this process until B_x gets empty.

Now both B_x and B_y are empty, and the next pair $R_i, R_j \in S$ that minimizes $max\{x_i + x_j, y_i + y_j\}$ fits into C_x or C_y. Assume, wlog it belongs to C_x. There are two possible cases:

There is a third element $R_q \in S$ such that $y_i + y_q \geq 1$ or $y_j + y_q \geq 1$. Assume, wlog, that $y_i + y_q \geq 1$, which leads to $x_i + x_q < 1$ and therefore $x_i + x_q + x_j < 2 < 1 + \nu$. Additionally, $y_i + y_j < 1,\ y_i + y_q \geq 1$ therefore, $1 \leq y_i + y_j + y_q < 2 < 1 + \nu$. We can remove these three elements from S and add their combination to A.

If there is no such third element it holds for each element $R_q \in S$ that $y_i + y_q < 1$ and $y_j + y_q < 1$. We show that in such a case there is at most one element $R_q \in S$ such that $x_q < y_q$. Assume for the sake of contradiction that there are two elements $R_q, R_t \in S$ such that $x_q < y_q$ and $x_t < y_t$. We know that either $x_i + x_q \geq x_i + x_j$ or $y_i + y_q \geq x_i + x_j$, because otherwise a different pair of elements would have obtain the minimum of the maximum. Because $y_i + y_q < 1$ then $x_i + x_q > x_i + x_j \geq \nu$. Also, $x_q \geq x_i$ because otherwise a different pair of elements would have obtain the minimum of the maximum. From the same considerations $x_q \geq x_j$. Recall that $x_i + x_j \geq \nu$, thus, $x_q \geq \frac{\nu}{2}$. Recall that by our assumption $y_q > x_q$, thus $y_q > \frac{\nu}{2}$. In the same way we can show that $x_t \geq \frac{\nu}{2}$,$y_t > \frac{\nu}{2}$, and therefore the combination of R_q and R_t belongs to A, which contradicts the fact that no combination of any pairs in S belongs to A. Therefore, the conditions of the Lemma 2 are satisfied and we can apply it. After each time the algorithm combines two elements, it checks whether we are left with exactly k pairs, and if so it stops and outputs the result. Therefore we do not explicitly mention this check in the algorithm itself.

3.2 Lower bounds

This section considers the tightness of the bounds of the union stage. For a given ν, the partition stage produces many parts such that each part R has weights $w_1(R) \leq (1 + \nu)W_1/k$ and $w_2(R) \leq (1 + \nu)W_2/k$. It was shown in [4] 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 values of ν at most 2. Therefore we can express ν as $1 + \epsilon$ when $0 < \epsilon \leq 1$.

Lemma 3. *There exist an input to the Bounded Pair Scheduling problem that can not be combined to k parts without exceeding* $2 + \frac{2\epsilon}{2+\epsilon}$.

Proof. Our example consists of two types of elements: type $A = (1 + \epsilon, 0)$ and type B, which will be defined later. First we set up an input with s elements

whose total weight is (s, s). We use only a single element of type A, so we are left with $s-1$ type B elements. The total value of all of the y's is s and only the $s-1$ type B elements contribute to this value. Therefore the y value of each such element is $\frac{s}{s-1}$. The total value of all of the x's is s, the type A element contributes $1+\epsilon$ to this sum, so the y value of each of the type B elements is $\frac{s-(1+\epsilon)}{s-1}$. We call this the *basic structure*. The basic structure will be replicated many times, as we'll show later.

Since type A elements get the maximal value $(1+\epsilon)$, combining two such elements yields a high value. Therefore, we need to balance between the combined value of A and B compared to the combined value of two B's. Combining a type A with a type B pair yields x value $1+\epsilon+\frac{s-(1+\epsilon)}{s-1}$ and y value $\frac{s}{s-1}$, i.e. the x value is greater.

Combining two type B pairs yields x value $2(\frac{s-(1+\epsilon)}{s-1})$ and y value $2\frac{s}{s-1}$, i.e. the y value is greater.

If we want to balance the x and y values, we would need to have $1+\epsilon+\frac{s-(1+\epsilon)}{s-1}$ equals to $\frac{2s}{s-1}$. For this to happen compute s as a function of ϵ:

$1+\epsilon+\frac{s-(1+\epsilon)}{s-1}=\frac{2s}{s-1}$

$s=2+\frac{2}{\epsilon}$.

We can assume that s is an integer. Otherwise we can represent s as a ratio of two integers $s=\frac{n}{d}$, and replicate each of the elements d times.

The y value of type B elements $=\frac{s}{s-1}=\frac{2+\frac{2}{\epsilon}}{2+\frac{2}{\epsilon}-1}=\frac{2+2\epsilon}{2+\epsilon}=1+\frac{\epsilon}{2+\epsilon}<1+\epsilon$.

The x value of type B elements $=\frac{s-(1+\epsilon)}{s-1}=1-\frac{\epsilon}{s-1}=1-\frac{\epsilon}{1+\frac{2}{\epsilon}}=1-\frac{\epsilon^2}{2+\epsilon}<1+\epsilon$. Therefore type B elements do not reach the $1+\epsilon$ threshold and are valid elements.

The above scenario is not interesting since the total value of W_1 and W_2 equals to the number of parts, so no parts should be combined. Therefore we tweak the example. Modify the basic structure as follows: For each basic structure subtract an infinitesimally small value $\delta \ll \epsilon$ from each of the coordinates of type B elements. This decreases the total value of each coordinate by $(s-1)\delta$. Now we will replicate the whole set $\frac{s-(s-1)\delta}{(s-1)\delta}=\frac{s}{(s-1)\delta}-1$ times. This leaves enough free space for an additional basic structure.

At this point we have to combine at least two elements. If we combine two type A elements, their x value will be $2+2\epsilon$. If we combine two type B elements, their y value will be $2+\frac{2\epsilon}{2+\epsilon}-2\delta\approx 2+\frac{2\epsilon}{2+\epsilon}$. The last possible combination is combining a type A element with a type B. The x value will be $(1+\epsilon)+(1-\frac{\epsilon^2}{2+\epsilon})-\delta=2+\frac{2\epsilon+\epsilon^2-\epsilon^2}{2+\epsilon}-\delta=2+\frac{2\epsilon}{2+\epsilon}-\delta\approx 2+\frac{2\epsilon}{2+\epsilon}$. Therefore no matter which pair we decide to combine, we get a value, as k goes to infinity of $2+\frac{2\epsilon}{2+\epsilon}$. □

Conclusion: Our algorithm is within $1+\frac{\epsilon^2}{4(1+\epsilon)}$ of the optimal.

Proof. Our algorithm achieves $1+\nu=2+\epsilon$, which is within $\frac{2+\epsilon}{2+\frac{2\epsilon}{2+\epsilon}}=\frac{2+\epsilon}{\frac{4+2\epsilon+2\epsilon}{2+\epsilon}}=\frac{4+4\epsilon+\epsilon^2}{4+4\epsilon}=1+\frac{\epsilon^2}{4(1+\epsilon)}$ of the example. □

4 Generalization to d Weight Functions

The important observation is that the algorithm of Subsection 3.1 can be viewed as a subroutine whose input is a partition of the vertices into k subsets, each having weight bounded by ν, and another partition into k subsets, each having weight bounded by ν. The result of the subroutine is a partition into k subsets, each having weight bounded by $\nu + 1$. Call this subroutine COMBINE. As presented, the sum of the weights in each of the two coordinates is bounded by k. We need to use the subroutine in a more general fashion, where the sum of the weights of each coordinate is bounded by mk, for a parameter m. The necessary change to COMBINE is in the definitions of the A, B, and C sets. It now becomes:

1. $S = \{(x, y) \mid x < m, y < m\}$
2. $A = \{(x, y) \mid m \leq x < m + \nu,\ m \leq y < m + \nu\}$
3. $B_x = \{(x, y) \mid m \leq x < \nu,\ y < m\}$
4. $B_y = \{(x, y) \mid x < m,\ m \leq y < \nu\}$
5. $C_x = \{(x, y) \mid \nu \leq x < m + \nu,\ y < m\}$
6. $C_y = \{(x, y) \mid x < m,\ \nu \leq y < m + \nu\}$

The result of the subroutine is a partition of the vertices into k subsets, each having weight bounded by $\nu + m$. Observe also that the cut capacity of the partition after subroutine COMBINE is bounded by the sum of the two initial cut capacities.

Assume now that we have d weight functions. Assume also that d is a power of 2. Using COMBINE we can construct $\frac{d}{2}$ partitions of the graph vertices, each into k subsets, and each subset having weight bounded by $1 + \nu$, where in the i-th partition the weights considered are w_{2i-1} and w_{2i}. We prepare these $\frac{d}{2}$ partitions for the next iteration, by considering partition i as having weight function $w_{1,i} = \max(w_{2i-1}, w_{2i})$.

We can now do the same process, but we now have only $\frac{d}{2}$ partitions. Use COMBINE to produce $\frac{d}{4}$ partitions of the graph vertices, each into k subsets, and each subset having weight bounded by $2 + \nu$. Again, we prepare these $\frac{d}{4}$ partitions for the next iteration, by considering partition i as having weight function $w_{2,i} = \max(w_{1,2i-1}, w_{1,2i})$.

After $\lceil \log d \rceil$ iterations, we have a partition into k subsets, each having weight bounded by $2^{\lceil \log d \rceil} - 1 + \nu \leq 2d - 1 + \nu$ in every weight function.

Since we employ subroutine COMBINE $\lceil \log d \rceil$ times, the final cut capacity as a result of our algorithm is the cut capacity resulting from a single partitioning multiplied by $O(d)$.

References

1. K. Andreev and H. Racke. Balanced graph partitioning. *Theory of Computing Systems*, 39(6):929–939, 2006.
2. S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings, and graph partitionings. In *36th Annual Symposium on the Theory of Computing*, pages 222–231, May 2004.

3. C. Chekuri and S. Khanna. On multidimensional packing problems. *SIAM Journal on Computing*, 33(4):837–851, 2004.
4. G. Even, J. Naor, S. Rao, and B. Schieber. Fast approximate graph partitioning algorithms. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 639–648. ACM, New York, 1997.
5. U. Feige and R. Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM J. Comput.*, 31(4):1090–1118, 2002.
6. A. E. Feldmann and L. Foschini. Balanced partitions of trees and applications. In *29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*, volume 14, pages 100–111, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
7. M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoret. Comput. Sci.*, 1(3):237–267, 1976.
8. M. N. Garofalakis and Y. E. Ioannidis. Parallel query scheduling and optimization with time-and space-shared resources. *SORT*, 1(T2):T3, 1997.
9. B. Hendrickson and R. W. Leland. A multi-level algorithm for partitioning graphs. *SC*, 95:28, 1995.
10. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.
11. R. Krauthgamer, J. S. Naor, and R. Schwartz. Partitioning graphs into balanced components. In *20th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 942–949. SIAM, 2009.
12. F. Leighton, F. Makedon, and S. Tragoudas. Approximation algorithms for VLSI partition problems. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 2865–2868. IEEE Computer Society Press, 1990.
13. T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.
14. R. MacGregor. *On Partitioning a Graph: A Theoretical and Empirical Study*. Memorandum UCB/ERL-M. University of California, Berkeley, 1978.
15. S. B. Patkar and H. Narayanan. An efficient practical heuristic for good ratiocut partitioning. In *16th International Conference on VLSI Design*, pages 64–69. IEEE, 2003.
16. D. Portugal and R. Rocha. Partitioning generic graphs into k regions. In *6th Iberian Congress on Numerical Methods in Engineering (CMNE2011)*, Coimbra, Portugal, Jun. 2011.
17. H. Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *40th Annual ACM Symposium on Theory of Computing*, pages 255–264. ACM, 2008.
18. H. D. Simon and S. Teng. How good is recursive bisection? *SIAM J. Sci. Comput.*, 18(5):1436–1445, 1997.