Karlsruher Institut für Technologie

Bachelor thesis

Better Recursive Graph Bisection

Yani Kolev

Date: 2. November 2017

Supervisors: Prof. Dr. rer. nat. Peter Sanders,
Dr. rer. nat. Christian Schulz,
M.Sc. Sebastian Schlag

Institute of Theoretical Informatics, Algorithmics
Department of Informatics
Karlsruhe Institute of Technology

Abstract

Graph partitioning finds practical application in multiple fields of engineering ranging from load balancing to route planning to VLSI design. The graph partitioning problem consists of dividing the vertices of a given graph into k different blocks of almost equal size so that a certain objective function is optimised. One of the most common approaches to solving this problem is recursive bisection (RB). It starts by dividing the graph into two roughly equally sized blocks and subsequently recursively bisecting each of them. However, it has been shown that RB suffers from a lack of global knowledge when each recursive cut is made and from the necessarily very strict balance applied in the first few recursion levels.

In this thesis we present a new algorithm that aims to improve on RB's inherent weaknesses by computing a fixed number of cuts with larger imbalances in the first l levels of recursion, continuing each new recursion branch to its end, and subsequently picking the one with the best result. The idea is that this would allow the algorithm to initially cut through sparser areas of the graph than RB, thanks to the relaxed balance constraint. After we present and explain the inner workings of the algorithm, we show and analyse experimental results comparing our algorithm to RB. We always at least match RB's result and achieve an improvement of 4% in around 40% of the test cases at the cost of a running time slower by a few orders of magnitude.

Zusammenfassung

Graphpartitionierung wird in mehreren Feldern der Ingenieurwissenschaften, wie Lastverteilung, Routenplannung und VLSI Design, eingesetzt. Das Graphpartitionierungsproblem besteht darin die Knoten eines gegebenen Graphen auf k disjunkte Blöcke fast gleicher Größe zu verteilen, sodass eine Zielfunktion optimiert wird. Einer der populärsten Lösungsansätzen ist die rekursive Bisection (RB). Hier wird der Graph in zwei etwa gleichgrößen Blöcken halbiert, wonach der Algorithmus auf beiden rekursiv angewendet wird. Jedoch ist es bekannt, dass RB suboptimale Resultate produziert wegen der sehr strikten Balanceeinschränkung in den ersten Rekursionsebenen und mangelhaften Wissens über die globale Struktur des Graphen zum Zeitpunkt jedes individuellen Schnittes.

In dieser Arbeit präsentieren wir einen neuen Algorithmus, der versucht diese Schwächen zu beseitigen, indem er eine feste Anzahl Schnitten mit größeren Imbalancen in den ersten l Ebenen ausrechnet, jeden der resultierenden Rekursionsbäumen zum Ende führt und den Besten auswählt. Das erlaubt unserem Algorithmus am Anfang durch relativ dünnbesetzte Bereiche des Graphen zu schneiden, dank der relaxierten Balanceeinschränkung. Nachdem wir die Struktur und Arbeitsweise des Algorithmus vorgestellt haben, werden Experimente präsentiert, die ihn mit RB vergleichen. BRBs Resultate sind immer mindestens so gut wie diese von RB und eine Verbesserung von 4% ist in rund 40% der Testfälle vorhanden. Jedoch ist BRBs Laufzeit um einige Größenordnungen schlechter.

Acknowledgments

I would like to express my sincerest gratitude to all those involved in the writing of my thesis. Foremost I would like to thank my supervisors M. Sc. Sebastian Schlag and Dr. rer. nat. Christian Schulz for their excellent tutelage and invaluable advise. Furthermore I thank Prof. Dr. Henning Meyerhenke and his entire workgroup for introducing me to the graph partitioning problem. Special thanks go out to my colleagues Kolja Esders and Daniel Seemaier for showing me how to work on complex software projects. Finally, without the unwavering support of my family and friends none of this would have been possible.

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und keine anderen, als die angegebenen Quellen und Hilfsmittel benutzt, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet habe.

Karlsruhe, November 2, 2017
Yani Kolev

Contents

1 Introduction

1.1 Motivation

Graph partitioning is a pivotal technique in a number of extremely diverse fields - load distribution [17], VLSI design [3] or the study of political gerrymandering [5]. It consists of dividing the vertices of a given graph into k disjoint sets, called blocks, so that some objective function is optimised. The choice of function depends on the practical application. In this thesis we focus on the version of the problem that minimises the number of edges with end vertices in two different blocks, i.e. the total cut, while ensuring that each block's size does not exceed $(1+\epsilon)$ times the average block size, where ϵ is a predetermined parameter called balance.

One of the problems graph partitioning helps solve is route planning [8, 11]. Here the goal is to compute the optimal path between two vertices in a given network according to a predetermined metric, such as travel time or traffic congestion. Some modern route planning algorithms start by executing a pre-processing stage where the given road network is divided into several subgraphs, called cells, i.e. partitioning it. Each cell is then individually examined and the shortest path between each pair of boundary nodes is calculated. Finally, when a user query comes in, it is not the possibly extremely large original graph that has to be processed but a vastly smaller one - only the union of each node's cell and the newly calculated minimal distances between boundary nodes in all other cells play a role in determining the optimal path [8]. A better cell choice, or better partition, leads to an improved final result.

A very intuitive approach to solving the graph partitioning problem is divide-and-conquer. The premise is that instead of directly partitioning the graph into k different blocks, it would be easier to first bisect it into two blocks, minimising the cut between them. Subsequently each block can, in turn, be further bisected recursively.

1.2 Contribution

Even though RB is one of the more popular ways of solving the graph partitioning problem, it has been shown that it can produce results very far from the optimum [14, 19]. One of the reasons for this is the very small imbalance allowed in the first levels of recursion. This is necessary, since even a small imbalance in one of the first levels can lead to far greater imbalances further down the recursion tree. Take a graph that is to be partitioned into 32 blocks. The average block weight is then $0.03125 \cdot w(V)$. An imbalance of just 1%

on each level could potentially lead to a block of weight $(1.01 \cdot 0.5)^5 \cdot w(V) \geq 0.0328 \cdot w(V)$, producing a final imbalance of nearly 5%. This greatly restricts the number of possible cuts in the first recursion levels.

In this thesis we allow a larger imbalance in the first few recursion levels. The idea is that this would allow the algorithm to, at first, cut through sparser areas of the graph than it normally would, producing a better local solution. Subsequently the goal is no longer to partition each side into strictly $k/2$ blocks, so the parameter is adjusted accordingly after each bisection. Afterwards the two sides are rebalanced using a modified FM-Local-Search [9] in order to ensure they are both roughly of size p times the average block size, where p is a natural number greater than zero. Intuitively the change should be most noticeable in graphs with highly irregular structures such as social networks. We evaluate our algorithm experimentally on several well known and widely used graph instances. Finally, we provide an overview of the obtained results.

Our algorithm consistently at least matches RB's results and provides an improvement of the final cut of at least 4% in around 40% of test cases, sometimes reaching as high as 20%. However, due to the exponentially growing number of recursion trees that arise on each level, its running time is l orders of magnitude worse than that of RB.

1.3 Structure of the Thesis

The remainder of this work is organised as follows: in Chapter 2 we describe the notation, the exact definition of the version of the graph partitioning problem on which we focus, and the objective function of interest. We continue by giving a short summary of the $KaHIP$ suite, where our algorithm is implemented, and by outlining the FM heuristic, on which we base the rebalancing routine used in this work. Chapter 3 introduces RB and our improvements on it. It also explains in greater detail how the most important components of our algorithm function. Chapter 4 concentrates on giving a detailed explanation of the test environment, the instances used, and the results they provided. Finally, in Chapter 5 we discuss possible areas of interest for future research.

2 Preliminaries

In this chapter we introduce some common notations and important definitions. We formally define the graph partitioning problem and the balance constraint. We continue by clarifying the objective function used in this work. Afterwards we present related work and give an overview of the $KaHIP$ suite, where our algorithm was implemented.

2.1 Notations and Definitions

An unweighted undirected graph G can be described as the tuple (V, E) where V is the set of nodes and E is the set of edges. We use $n := |V|$ and $m := |E|$ as shorthands for their respective cardinalities. A simple graph contains no self loops or parallel edges, i.e. multiple edges between the same pair of vertices. Each edge is represented as the set $\{v, w\}$ where v and w are the end vertices. In this thesis we work exclusively with unweighted, undirected simple graphs. The neighbourhood of a vertex v is defined as $\{w \in V | \{v, w\} \in E\}$ and is denoted as $N(v)$. Note that while in the instances we use all vertices are of unit weight, in some steps of the algorithms presented new graphs are generated that do not possess this quality. Thus we denote the weight of a vertex v as $w(v)$. If $V' \subseteq V$, then $w(V')$ denotes the sum of the weights of the vertices in V', or just the number of vertices in V, provided they are all unweighted.

A k-way partition of a given graph $G = (V, E)$ consists of k sets of vertices $V_1, \ldots, V_k$, called blocks, such that $V_1 \cup \cdots \cup V_k = V$ and $V_1 \cap \cdots \cap V_k = \emptyset$ for $i, j \in \{1, \ldots, k\}$. A two-way partition is also called a bisection. We define the average block weight of a k-way partition, b_k, as $\lceil w(V)/k \rceil$. It is used to define the balance constraint: $\forall i \in \{1, \ldots, k\}$: $w(V_i) \leq (1 + \epsilon) \cdot b_k$, where ϵ is a predefined parameter. The k-way graph partitioning problem consists of creating a k-way partition of G which satisfies the balance constraint for some given value of ϵ and optimises some objective function.

Calculating a partition creates so-called border vertices and border edges. A vertex v is considered a border vertex if there exists an edge $\{v, w\}$ such that $v \in V_i$ and $w \in V_j$ with $i \neq j$ and $i, j \in \{1, \ldots, k\}$. An edge $\{v, w\}$ is considered a border edge if $v \in V_i$ and $w \in V_j$ with $i \neq j$ and $i, j \in \{1, \ldots, k\}$.

As stated above, graph partitioning optimises some objective function. In this thesis we focus on the total cut due to its simplicity and considerable correlation with numerous other metrics [7]. Given a partition $P = \{V_1, \ldots, V_k\}$ of a simple unweighted undirected graph, the number of border edges, or $c := |\{\{v, w\} \in E | v \in V_i \wedge w \in V_j \wedge i \neq j\}|$ is defined as the total cut.

2.2 Related Work

The decision problem corresponding to graph partitioning is NP-hard [13] and the instances it examines are often extremely large. Furthermore, even providing a solution quality guarantee is NP-hard in some cases [4]. Due to this and the problem's demonstrated ubiquity, numerous heuristic graph partitioning methods have been developed over the last several decades. These include spectral graph partitioning [10], branch-and-cut algorithms [6] as well as some genetic approaches [15]. An excellent and comprehensive overview of the recent developments in the field is given in [7]. In this section we introduce the Fiduccia-Matteyses improvement heuristic, variants of which are pivotal to our algorithm. We continue by giving a brief overview of the $KaHIP$ suite.

Fiduccia-Mattheyses Heuristic

The Fiduccia-Mattheyses heuristic [9] (FM), first introduced in the eighties, takes a given partition and improves it. Over the past several decades its numerous variations have become an integral part of many graph partitioning schemes.

It shares some of its core concepts with the well known Kernighan-Lin algorithm [14] (KL). First FM defines a gain function which denotes by how much the total cut would improve if a vertex is moved to the opposite block. Note that this value can be negative. Then it improves the given bisection by performing a number of passes over it. The vertices on each side are kept in two data structures consisting of arrays of linked lists, where the kth list contains the unmoved vertices with a gain of k. This allows us to easily maintain each array by moving a vertex to the appropriate list whenever its gain changes due to a neighbour being moved. The vertex with the highest gain, the movement of which would not violate the balance constraint, is selected and moved to the opposite block. This is one of the major differences to KL, as there instead of moving one vertex, the algorithm swaps a pair of vertices. Each vertex can only be moved once during a pass, after which the respective gains of its neighbours are recalculated. This is repeated until some stopping criterion is met, at which point the best partition encountered is selected. Then a new pass starts, until no further improvement is found. The worst case running time of a pass is linear in the size of the graph and in practice only a few passes are usually performed [9].

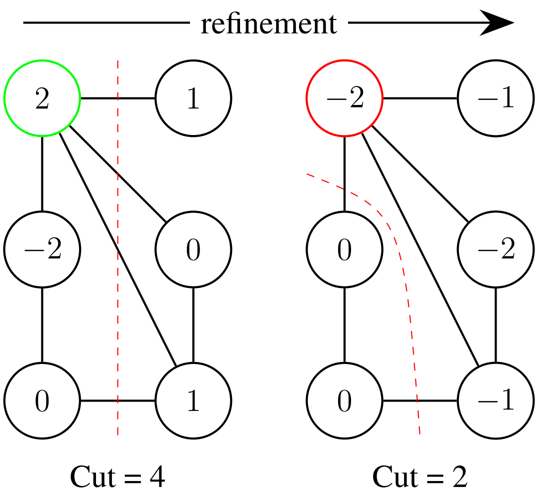


Figure 2.1: A single FM step

Figure 2.1 illustrates how a single vertex transfer from one side to the other works. In this example we omit the balance constraint for simplicity. It prohibits the vertex with the highest gain from being moved, if that would make the partition imbalanced. First we see the graph where each vertex is labeled with its respective gain. As vertices can only be moved once in a single pass, already moved vertices are painted red. We see that the algorithm selects the vertex with the highest gain from the two sides not yet moved. It is marked in green in Figure 2.1. This vertex is then moved to the opposite side, decreasing the total cut by 2. The next vertex that would be selected by FM is one of the vertices with gain 0. This is not a problem, as moving one of them could influence other vertices' gain, possibly increasing it above 0. Indeed, even moving a vertex with a negative gain is possible, since that allows the algorithm to climb out of local optima to some extent.

KaHIP

Developed at the KIT by Sanders, Schulz et.al. [18], $KaHIP$ (Karlsruhe High Quality Partitioning) is the graph partitioning framework where our algorithm is implemented. It includes $KaFFPa$ (Karlsruhe Fast Flow Partitioner), $KaFFPaE$ ($KaFFPaEvolutio-nary$) which is a parallel evolutionary algorithm that uses $KaFFPa$ to provide combine and mutation operations, as well as $KaBaPE$ which extends the evolutionary algorithm [2]. Some of these partitioning techniques are based on the multi-level graph partitioning paradigm. Algorithms which employ it are among the most successful heuristics in their field [12, 7].

The multi-level approach consist of three phases - coarsening, initial partitioning and uncoarsening. In the coarsening phase more and more details of the graph are omitted through iterated contractions. A contraction is the merging of some non-empty subset of a graph's vertices into a single new vertex with weight equal to the sum of the weights of the merged vertices. For every vertex $v \in V$ we denote the vertex to which it is contracted as $C(v)$. Let $\{v, w\} \in E$ be an edge. If $C(v) \neq C(w)$ a new edge is added between $C(v)$ and $C(w)$, else the edge is simply omitted in the coarser level. The weights of parallel edges are added together. The graph is iteratively contracted until some threshold is met. This results in a new far smaller graph with the same basic structure and core properties as the original one. Because of its smaller size, more expensive heuristics can be used in order to produce a better solution in the initial partitioning phase. In the uncoarsening phase the graph is gradually uncoarsened back to its original state.

$KaHIP$ further improves on this by employing iterated multi-level algorithms. They iterate through coarsening and uncoarsening phases based on different random seeds in order to furhter improve solution quality. We also took advantage of $KaHIP$'s implementation of the 2-way-FM heuristic. In $KaHIP$'s case only the border vertices are considered during each pass. A further improvement is the max-flow min-cut strategy. Here a flow problem is created around border vertices so that each $s-t$ cut represents a valid balanced partition and the optimal cut is then calculated. For further information on $KaHIP$, its graph partitioning capabilities and its preconfigurations, we refer to [18] and the $KaHIP$ user guide [2].

3 Better Recursive Bisection

In this chapter we present our new better recursive bisection graph partitioning algorithm (BRB). First we outline the original recursive bisection algorithm. Then we present the general idea behind BRB and explain which of RB's weaknesses it addresses. Afterwards we illustrate the workflow of each individual component with high level pseudocode, while simultaneously giving a more detailed overview of the critical or novel sections, accompanied by more detailed pseudocode.

3.1 Recursive Bisection

The well known recursive bisection algorithm is a typical way of solving the graph partitioning problem. It functions by recursively dividing the graph in two halves until k blocks are present. A pseudocode description is provided in Algorithm 1.

Algorithm 1: RECURSIVE BISECTION

```
Input: A graph G, integer k, and balance ϵ
if k > 1 then
    (V_1, V_2) ← bipartition(G, ϵ)
    recursiveBisection(V_1, ⌈k/2⌉, ϵ)
    recursiveBisection(V_2, ⌊k/2⌋, ϵ)
```

3.2 Overview of BRB

Our algorithm works as follows: for the first l levels of recursion it creates several different cuts, each one generated using a different imbalance. We call the set of these imbalances S. Then it calculates into how many blocks each side is to be partitioned in order to fulfil the balance constraint. Using a predetermined deviation of those, q, the block numbers that produce the best cut are selected and a modified version of FM is used to rebalance accordingly. The algorithm then calls itself recursively on each side with k equal to its respective number of blocks. This is done for all $|S|$ cuts. Each new recursion tree is then followed to its end and finally the best result is selected. An overview is provided in Algorithm 2.

Algorithm 2: BETTER RECURSIVE BISECTION

`Input:` A graph G, integer k, integer l, integer $level$, set of integers S, integer q, and balance ϵ

`if` $level \leq l$ `then`
    $(V_1, lhs_b, V_2, rhs_b) = tryMultipleImbalances(G, S, k, \epsilon)$
    $level \leftarrow level + 1$
    $betterRecursiveBisection(V_1, lhs_b, l, level, S, \epsilon)$
    $betterRecursiveBisection(V_2, rhs_b, l, level, S, \epsilon)$
`else`
    $recursiveBisection(G, k, \epsilon)$

3.3 Improvement on RB

The lack of global information at the time of each bisection makes it possible for RB to select a cut that, while perhaps better at the current level, greatly reduces the overall solution quality [14, 19]. This, coupled with the very strict balance constraint in the first recursion levels, makes it possible for RB to reach a solution arbitrarily far from the optimum [19]. A perfect demonstration of this is given in [19] - there a class of graphs is presented, for which RB always produces results very far from the optimum. This is still the case even when employing an optimal bipartitioning algorithm.

Consider the class of graphs shown in Figure 3.1. For clarity, in this example assume i and j are integers in $\{1, 2, 3, 4\}$. Let δ_i be real numbers so that:

(i) $-1/8 < \delta_i < 1/8$

(ii) $\sum_{i=1}^{4} \delta_i = 0$

(iii) $\delta_i \neq 0$

(iv) $\nexists i, j \in \{1, 2, 3, 4\} : \delta_i + \delta_j = 0$

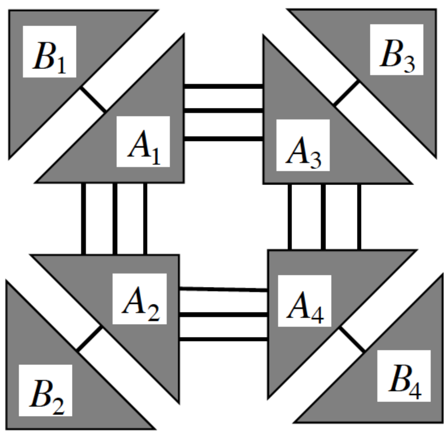


Figure 3.1: Example taken from [19]

Each graph in this class contains eight subgraphs A_i and B_i. Each A_i is connected to B_i with exactly one edge and to two of the other A_js with three edges each. Each A_i is of weight $(1/8+\delta_i)\cdot w(V)$ and each B_i is of weight $(1/8-\delta_i)\cdot w(V)$. This ensures each $A_i \cup B_i$ has a total weight of exactly $1/4 \cdot w(V)$.

If the graph is to be ideally partitioned, i.e with $\epsilon = 0$, the best solution for $k = 4$ has a total cut of 12 where each block $V_i = A_i \cup B_i$. However, an optimal bipartitioning algorithm would find the cut of weight 4 where $V_1 = \bigcup_{i=1}^{4} A_i$ and $V_2 = \bigcup_{i=1}^{4} B_i$ in its first level of recursion. This is a perfectly balanced partition because of (ii). Let us examine the recursive call performed on V_1. The goal is to divide this subgraph into two equally sized blocks of weight exactly $1/4 \cdot w(V)$. The algorithm can not simply pack together two of the As, A_i and A_j, into one block, since the weight of that block would be $(\delta_i + \delta_j + 1/4) \cdot w(V) \neq 1/4 \cdot w(V)$ because of (iv). Thus the algorithm would need to cut through at least one of the As, producing a final solution in $\Omega(n)$, in case the graph is sparse, or in $\Omega(n^2)$, in case it is dense, both of which are very far from the optimum of 12. The same would apply for a larger imbalance ϵ if the δ_is are chosen aptly.

3.4 Multiple Cuts on Each Level

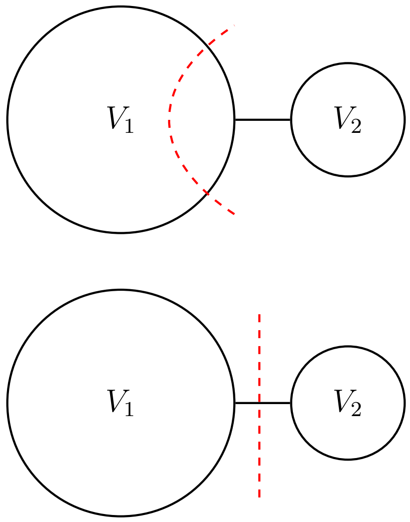


Figure 3.2: RB (above) vs. BRB (below)

Our solution to this problem is to allow a greater imbalance ϵ^* in the first few levels of recursion. This allows BRB to find the optimal cut between w.l.o.g. $A_1 \cup B_1$ and the rest of the graph. A simpler example is presented in Figure 3.2.It depicts a graph consisting of two relatively dense areas of vastly different sizes, connected by only a few edges. In this case RB would be forced to cut through the larger one, so as not to violate the balance constraint. One of our goals when developing BRB was to ensure that the beneficial small cut between the two areas is found and used, as illustrated in Figure 3.2. We achieve this by calculating multiple different cuts with different imbalances on the first l recursion levels, following each resulting recursion tree to its end, and picking the best one.

Producing a number of cuts based on different imbalances prevents the following issue: assume that in Figure 3.2 $w(V_1) = 0.95 \cdot w(V)$, $w(V_2) = 0.05 \cdot w(V)$ and $k = 4$. Then $b_4 = 0.25 \cdot w(V)$. If $\epsilon^* = 0.9$ the algorithm would find the optimal cut between V_1 and V_2. However, in order for our final partition not to violate the original balance constraint ϵ, V_1 would need to be partitioned into three blocks, meaning its weight should be almost $0.75 \cdot w(V)$. In contrast V_2 should not be partitioned any further, but its weight needs to be $0.25 \cdot w(V)$. So the two sides would be rebalanced using our modified FM, which would move vertices with a total weight $0.2 \cdot w(V) = 0.8 \cdot b_4$ from V_1 to V_2. This would effectively cut through the dense area V_1 and defeat the entire purpose of the relaxed balance constraint, as there would be no reason to assume that this new cut would be any better than RB's alternative. Examining other imbalances between 0 and 0.9 allows us to produce better cuts that are closer to a feasible state, i.e. a state where each side is of weight $p \cdot b$, where b is the global average block weight and p is some integer. A pseudocode description is provided in Algorithm 3.

Algorithm 3: TRYING MULTIPLE CUTS

`Input`: A graph G, integer set S, integer k, ϵ
`Output`: sides V_1 and V_2, number of blocks on each side lhs_b and rhs_b
`for` $\epsilon^* \in S$ `do`
    $(V_1, V_2) \leftarrow bipartition(G, \epsilon^*)$
    $(lhs_b, rhs_b) \leftarrow calculateBlocksOnEachSide(V_1, V_2, k)$
    $rebalanceSides(V_1, lhs_b, V_2, rhs_b)$
    $betterRecursiveBisection(V_1, lhs_b, l, S, \epsilon)$
    $betterRecursiveBisection(V_2, rhs_b, l, S, \epsilon)$
    $calculateCut(G)$
$(V_1, V_2) \leftarrow selectBestCut(G)$
$(lhs_b, rhs_b) \leftarrow calculateBlocksOnEachSide(V_1, V_2, k)$
`return` (V_1, lhs_b, V_2, rhs_b)

The reason we follow each recursion tree to its end is that, while severely increasing the algorithm's running time, this allows BRB greater flexibility when it comes to selecting the optimal cut on each level. The alternative - which would be simply picking the optimal cut at the current level - exhibits the same weakness as RB. In the example provided in Figure 3.1 the same cut would be selected at the first level as in the case of RB, regardless of how many different imbalances we test. This is the case since it has the minimal weight of any feasible cut. Without looking further down the recursion tree it is impossible to know whether or not it actually produces a better solution than the others.

3.5 Rebalancing

The rebalancing we use is based on FM. This method calculates the optimal size of each side and moves vertices accordingly, ensuring BRB is able to obtain a balanced partition in the end. First the ideal number of blocks, into which each side is to be partitioned, is calculated using the $calculateBlocksOnEachSide$ method. It functions as follows: first it divides $w(V_1)$ by b (the global average block weight) and rounds this to an integer. If this number is 0, it is set to 1, and if it is k, it is set to $k-1$, as partitioning one of the sides into 0 blocks would simply return the graph to its original state. Then the number of blocks on the right side is calculated as the difference between the total number of blocks and the number of blocks on the left. An overview is provided in Algorithm 4. Then the ideal weight of each side is calculated. These will be denoted as w_{lhs} and w_{rhs}. The formula used is $w_{lhs} = w(V_1 \cup V_2) \cdot lhs_b/(lhs_b + rhs_b)$. This works equivalently for the right side. Afterwards a modified version of FM is used. It works as follows: the vertex queue of each side is initialised only with its border vertices. Then the side that is above its target weight (and unless we are already at the optimal weights, there will always be a side that is too

Algorithm 4: CALCULATE BLOCKS ON EACH SIDE

`Input`: Graphs V_1 and V_2, integer k
`Output`: tuple of integers (lhs_b, rhs_b)

$lhs_b \leftarrow round(w(V_1)/b)$
`if` $lhs_b = 0$ `then`
  $lhs_b \leftarrow lhs_b + 1$
`if` $lhs_b = k$ `then`
  $lhs_b \leftarrow lhs_b - 1$
$rhs_b \leftarrow k - lhs_b$
`return` (lhs_b, rhs_b)

heavy) is selected and the node with the highest gain is moved to the opposite one. Note that this gain is often negative, which results in an overall worsening of the obtained cut, but gives the algorithm the ability to climb out of local minima to an extent. However, as this method presupposes the existence of border vertices, it runs into the problem of cuts of weight 0 in which case no border vertices exist.

Cuts of Weight 0

In connected graphs, such as the ones examined in this work, a single bisection cannot produce a cut of weight 0. Nevertheless, it is possible for such cuts to arise further down the recursion tree if somewhere higher up on the recursion tree one of the sides resulting from a bisection was itself disconnected. Since a 0-cut means there are no border edges and no border vertices, it would render two-way-FM useless - no gains would be calculated on either side because only border vertices are usually considered, and no vertices would be moved. We work around this issue by simply moving a single random vertex from the side that is too heavy to the side that is too light, thus creating a non-empty boundary, and continuing with the usual rebalancing routine. An example of the entire process is provided in Figure 3.3. The vertices on each side of the cut are labeled with their respective gain. Non-border vertices are left blank. When a vertex is moved it is painted red, as it can not be moved again during the same pass.

In the first figure we see the left side of weight 9 and a right side of weight 3. The target weight of each side is 6. This means that three nodes are to be moved from the left side to the right side. There are only two border vertices present. The one from the side above its target weight, in this case the left side, is selected and moved to the right side. It is painted green for clarity. In the second figure the left side has a weight of 8 and the right side - 4. Two more vertices need to be moved to the right side. However, there are no border vertices on the left side anymore. This is why a vertex is selected at random, painted here in blue, and moved to the side that is too light - the right side. Note that its gain has not yet been calculated, as it is not a border vertex.

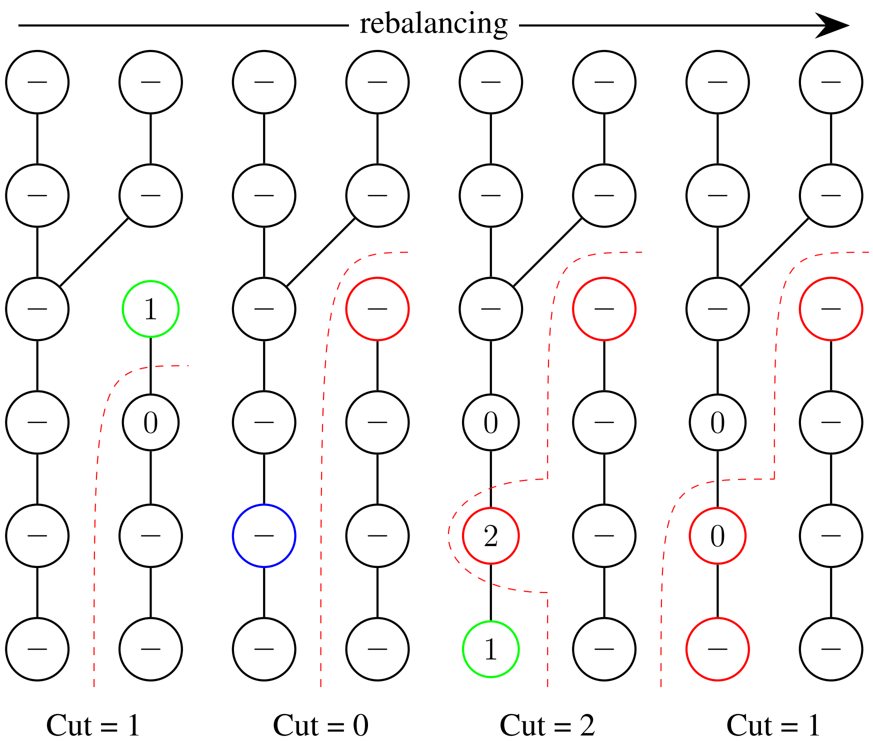


Figure 3.3: Rebalancing

The third figure shows the two sides of weights 7 and 5. One more vertex needs to be moved to the right side. We now see two border vertices that have not been moved on the left side with respective gains 0 and 1. The latter is selected and moved to the right side, ensuring the target weights of 6 are met. If we observe the final figure's right side, we can see how a cut of weight 0 can arise. A bipartition of it with $\epsilon \geq 0.25$ would split the nodes into two groups - the two on the left and the four on the right, producing a cut of weight 0.

3.6 Deviation Search

The $calculateBlocksOnEachSide$ method gives us the number of blocks into which it would be most intuitive to partition the two sides, but that does not guarantee that this is indeed the optimal solution. Figure 3.4 provides an example where deviating from those values produces a better local cut. Each vertex is labelled with its weight. Assume that $b = 4$. A high enough imbalance ϵ would find the cut of weight 1, drawn in black. Rebalancing, however, would calculate that the left side needs to be of weight $1 \cdot b$. So it would need to have weight 4, which is why a further vertex of weight 1 must be added to it, hence the red cut of weight 3 would be found. But if we deviate from the calculated number of blocks on each side by 1 the algorithm would try to rebalance the block to a weight of $1 \cdot b$ and $2 \cdot b$, i. e. 8, resulting in the smaller green cut of weight 2. In order to ensure such better cuts are

Algorithm 5: DEVIATION SEARCH

Input: Graph V_1, graph V_2, number of blocks on each side lhs_b and rhs_b

1 for $q^* \in \{0, \ldots, q\}$ do
2     $reset(V_1, V_2)$
3     $lhs_{est} \leftarrow lhs_b - q^*$
4     $rhs_{est} \leftarrow rhs_b + q^*$
5   if $lhs_{est} < 1 \quad || \quad rhs_{est} < 1$ then
6         $continue$
7     $w_1^* \leftarrow w(V_1 \cup V_2) \cdot (lhs_{est}/k)$
8     $w_2^* \leftarrow w(V_1 \cup V_2) - w_1^*$
9     $rebalance(V_1, w_1^*, V_2, w_2^*)$
10     $recordCut(V_1, V_2)$
11 $applyBestCut(V_1, V_2)$

always found we incorporate another tuning parameter into our program, q, which denotes by how much we allow the number of blocks on each side to deviate. During rebalancing the optimal weights of the two sides, w_{lhs} and w_{rhs}, are based on the number of blocks in each - lhs_b and rhs_b. Instead of only calculating a single version of w_{lhs} and w_{rhs}, several are calculated, ranging from ones based on $lhs_b - q$ and $rhs_b + q$ to $lhs_b + q$ and $rhs_b - q$. Algorithm 5 provides an overview.

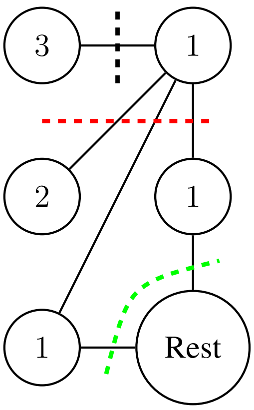


Figure 3.4: Deviation benefits

In some cases using the deviation would lead to one side having less than 1 or more than k blocks, which is why we safeguard against that with the if condition on line 4. RB uses a the same basic routine, just without looking for the best deviation. The $reset$ method brings the two sides back to their original state. The $rebalance$ method refers to the FM version described in the previous section. $RecordCut$ records the size of the cut so that $applyBestCut$ can later select the optimal one. It is also important to note that BRB no longer necessarily matches RB's result in each case if deviation search is used, as, due to the greedy selection of the ideal number of blocks on each side, it is possible that a different locally better cut is picked and the exact recursion tree that lead to RB's final partition is never replicated.

4 Experimental Evaluation

In this chapter we present the findings of the experimental evaluation of our new better recursive bisection algorithm. We start by describing the methodology behind these experiments. Then we outline the machines used for the experiments and continue by listing the available tuning parameters. Afterwards we present the examined instances and provide sources for them. In closing we compare BRB's performance to that of KaHIP's implementation of RB.

4.1 Methodology

We concentrate on two kinds of data - average values (for comparison tables) and plots that provide insight into the progress of solution quality (performance plots) [18, 16]. We test different algorithm configurations on multiple instances and for $k \in \{8, 16, 32, 64\}$. We use the $ecosocial$ preconfiguration of the $KAHIP$ suite, as it provides the best tradeoff between running time and solution quality [18]. Every algorithm is run on each instance-k pair three separate times, after which the arithmetic mean of the three results is calculated.

First we present comparison tables, which are constructed as follows: for each value of k the geometric mean of the arithmetic mean values for each instance is taken, as this allows smaller graphs to still have the same impact on the final result. Afterwards we present the performance plots. They are generated as follows: the best performing algorithm configuration is selected on an instance by instance basis. Then for every algorithm configuration the ratio between its solution and the best one is calculated and subtracted from 1. Afterwards the results for each configuration are sorted in a non-increasing order by this new ratio and are plotted. Hence the lower the point, the better the result. Each point on the plot represents the solution a certain algorithm provided for a single instance-k pair. For further information on the performance plots refer to [16].

4.2 Tuning Parameters

In this subsection we describe the tuning parameters of BRB. They have an effect on the running time as well as on the solution quality. The parameter $imbalancedBisectionEpsilon$ denotes the upper bound for the imbalances used in S. All other imbalances are generated by iteratively subtracting 0.1 from it until zero is reached. For example, an $imbalancedBi-sectionEpsilon$ of 0.2 would generate S as $\{0.2, 0.1, 0\}$. The value we always utilise is

0.9, hence it is not explicitly specified in each individual chart or plot. The next parameter - l - describes the greatest depth at which we call $tryMultipleImbalances$. We set it to $1, 2$ or 3. Finally, q is the maximum deviation from the desired number of blocks on each side to be checked in the rebalancing method - we set this to 0 or 1.

4.2.1 Instances

Our algorithm was tested on 27 graphs divided into three sets. The first consisted of the instances from Chris Walshaw's Graph Partitioning Archive [1]. The second was a collection of various social network graphs. All of them are listed in Table 4.1. The graphs rgg17 and rgg18 are random geometric graphs with 2^{17} and 2^{18} vertices. Other geometric instances are delaunay17 and delaunay18 - these are delaunay triangulations with 2^{17} and 2^{18} vertices respectively.

Graph	Nodes	Edges	Graph	Nodes	Edges
Walshaw Graphs			Social Network Graphs		
bcsstk29	13 992	302 748	p2p-Gnutella04	6 405	29 215
4elt	15 606	45 878	wordassociation-2011	10 617	63 788
fe_sphere	16 386	49 152	PGPgiantcompo	10 680	24 316
cti	16 840	48 232	as-22july06	22 963	48 436
memplus	17 758	54 196	soc-Slashdot0902	28 550	379 445
cs4	22 499	43 858	loc-brightkite	56 739	212 945
fe_pwt	36 519	144 794	enron	69 244	254 449
bcsstk32	44 609	985 046	finan512	74 752	261 120
fe_body	45 087	163 734	loc-gowalla	196 591	950 327
t60k	60 005	89 440	coAuthorsCiteseer	227 320	814 134
wing	62 032	121 544	wiki-Talk	232 314	1 458 806
fe_rotor	99 617	662 431			
Geometric Graphs					
rgg17	131 072	728 753	delaunay17	131 072	393 176
rgg18	262 144	1 547 283	delaunay18	262 144	786 396

Figure 4.1: Instances used in experiments

4.3 Environment

We use two machines for our experiments. Machine A uses gcc version 4.8.5 and machine B uses 4.8.4. Machine A runs Ubuntu 14.04 LTS, has two Intel Xeon E5-2670 v3

- 2.3 GHz 12-core CPUs and 128 GIB DDR4-RAM. Machine B runs Ubuntu 12.04, possesses four AMD Opteron 6168 1.9 Ghz 12-core CPUs and 256 GB RAM.

4.4 Comparison to RB

We compare BRB's results to those of RB. Tables 4.1 through 4.8 provide a comparison of the geometric means of important metrics used to describe an algorithm's efficiency between RB and BRB. The best results in the cut and time columns are bolded.

Param.	Avg. Cut	Best Cut	Worst Cut	Avg.Bal.	Worst Bal.	Time (s)
RB	7 707	7 411	8 054	1,01	1,01	**1,51**
$l = 1$	7 420	7 248	7 623	1,01	1,01	22,27
$l = 2$	7 268	7 113	7 433	1,01	1,01	172,48
$l = 3$	**7 220**	**7 072**	**7 381**	1,01	1,01	887,14

Table 4.1: $k = 8$, $q = 0$, Machine B

Param.	Avg. Cut	Best Cut	Worst Cut	Avg.Bal.	Worst Bal.	Time (s)
RB	7 707	7 411	8 054	1,01	1,01	**3,50**
$l = 1$	7 467	7 309	7 631	1,01	1,01	52,88
$l = 2$	7 430	7 239	7 626	1,01	1,01	428,14
$l = 3$	**7 300**	**7 140**	**7 478**	1,01	1,01	2 488,76

Table 4.2: $k = 8$, $q = 1$, Machine A

Param.	Avg. Cut	Best Cut	Worst Cut	Avg.Bal.	Worst Bal.	Time (s)
RB	11 242	10 983	11 500	1,01	1,02	**2,06**
$l = 1$	11 012	10 856	11 153	1,01	1,01	27,86
$l = 2$	10 871	10 726	10 998	1,01	1,01	231,23
$l = 3$	**10 674**	**10 571**	**10 784**	1,01	1,02	1 690,57

Table 4.3: $k = 16$, $q = 0$, Machine B

Param.	Avg. Cut	Best Cut	Worst Cut	Avg.Bal.	Worst Bal.	Time (s)
RB	11 242	10 983	11 500	1,01	1,02	**4,40**
$l = 1$	11 065	10 878	11 234	1,01	1,01	61,49
$l = 2$	10 929	10 801	11 063	1,01	1,01	522,14
$l = 3$	**10 821**	**10 661**	**10 985**	1,01	1,01	3 986,73

Table 4.4: $k = 16$, $q = 1$, Machine A

Param.	Avg. Cut	Best Cut	Worst Cut	Avg.Bal.	Worst Bal.	Time (s)
RB	15 840	15 572	16 181	1,02	1,02	**2,41**
$l = 1$	15 596	15 433	15 759	1,02	1,02	30,92
$l = 2$	15 386	15 240	15 523	1,02	1,02	268,44
$l = 3$	**15 161**	**15 046**	**15 279**	1,02	1,02	2 145,98

Table 4.5: $k = 32$, $q = 0$, Machine B

Param.	Avg. Cut	Best Cut	Worst Cut	Avg.Bal.	Worst Bal.	Time (s)
RB	15 840	15 572	16 181	1,02	1,02	**5,25**
$l = 1$	15 616	15 447	15 797	1,02	1,02	69,34
$l = 2$	15 440	15 286	15 581	1,02	1,02	603,60
$l = 3$	**15 266**	**15 158**	**15 368**	1,01	1,02	4 910,29

Table 4.6: $k = 32$, $q = 1$, Machine A

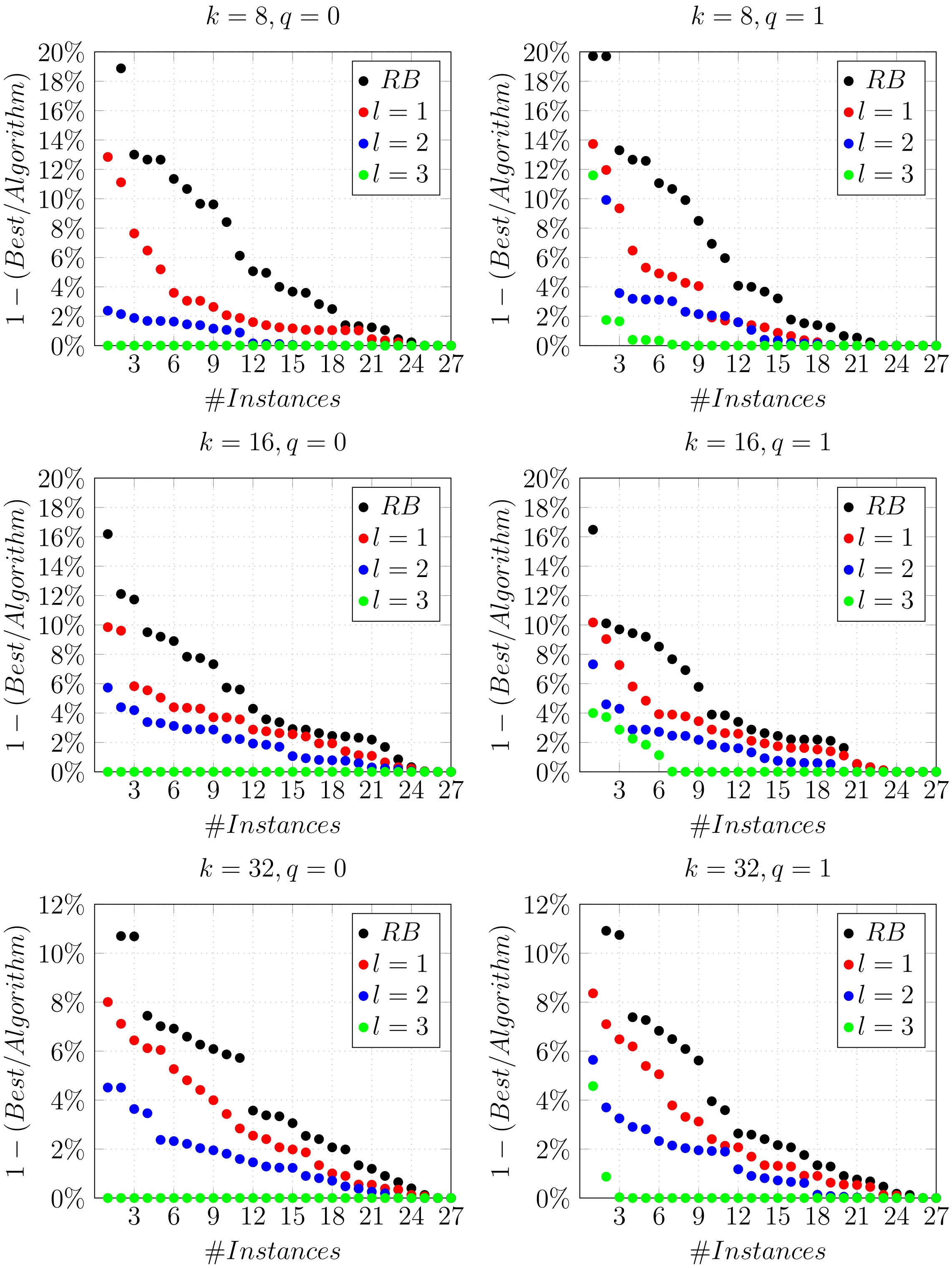
Param.	Avg. Cut	Best Cut	Worst Cut	Avg.Bal.	Worst Bal.	Time (s)
RB	22 305	22 037	22 591	1,02	1,02	**6,40**
$l = 1$	22 005	21 847	22 177	1,02	1,02	94,61
$l = 2$	21 781	21 651	21 926	1,02	1,03	690,23
$l = 3$	**21 488**	**21 390**	**21 600**	1,02	1,03	5 499,01

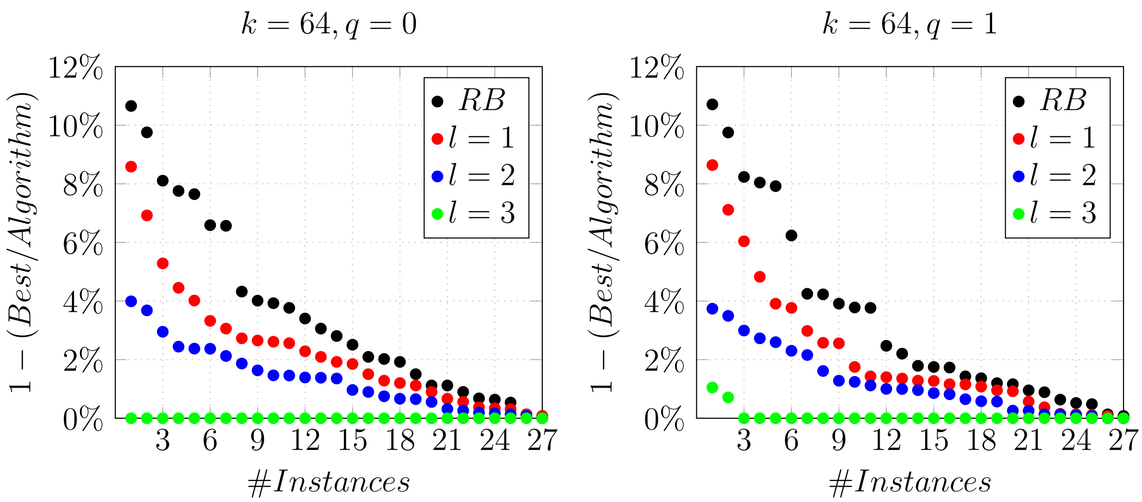
Table 4.7: $k = 64$, $q = 0$, Machine B

Param.	Avg. Cut	Best Cut	Worst Cut	Avg.Bal.	Worst Bal.	Time (s)
RB	22 305	22 037	22 591	1,02	1,02	**6,40**
$l = 1$	22 029	21 850	22 218	1,02	1,02	95,56
$l = 2$	21 819	21 697	21 955	1,02	1,03	702,11
$l = 3$	**21 568**	**21 460**	**21 685**	1,02	1,02	5 667,18

Table 4.8: $k = 64$, $q = 1$, Machine B

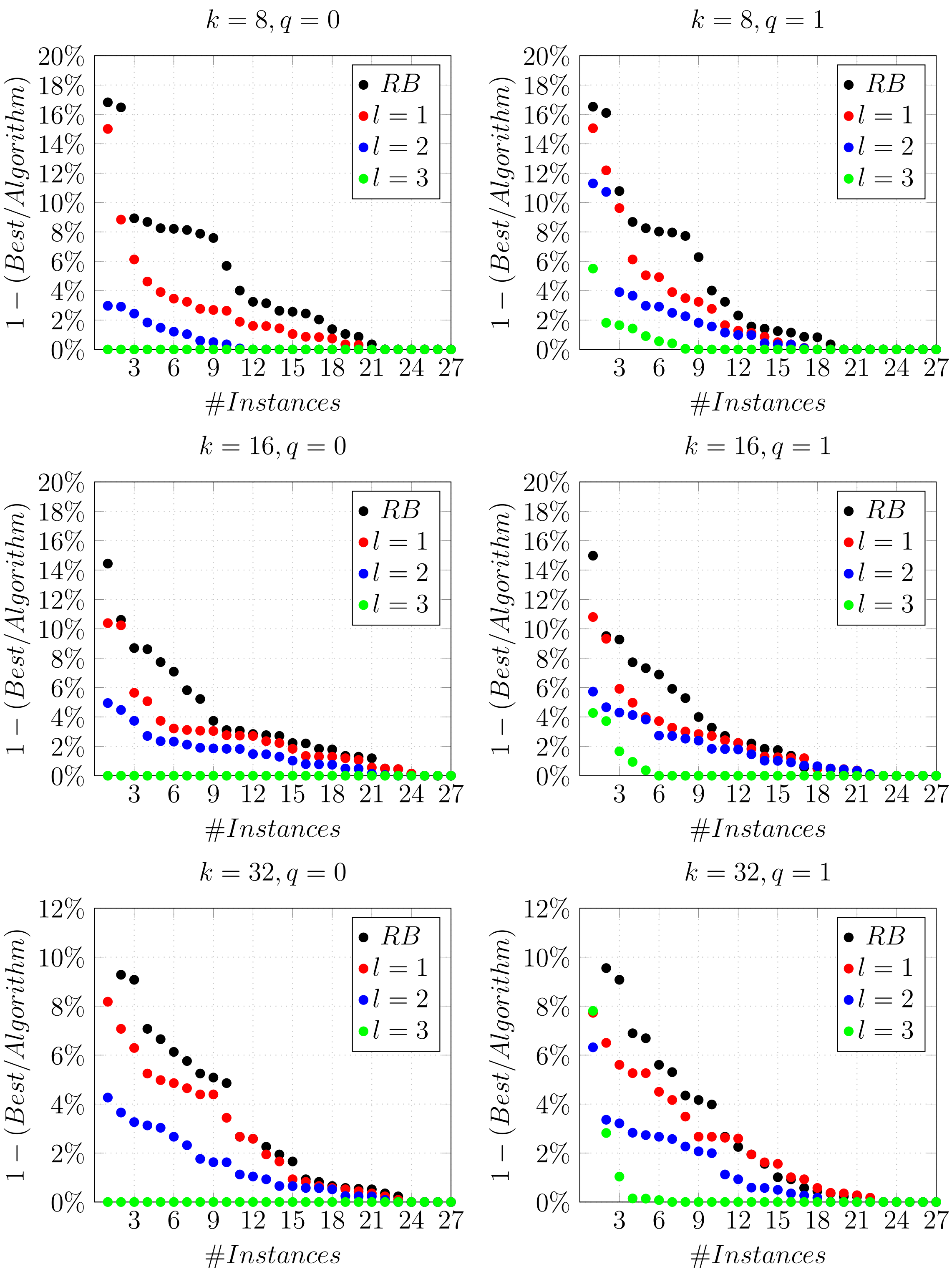
These results underline that BRB does not violate the balance constraint substantially more often than RB. However, in order to better visualise the large amount of test data, some which can be found in Appendix A, the following performance plots are provided. First we present plots for the average cut calculated by the algorithms.


k = 8, q = 0
k = 8, q = 1
k = 16, q = 0
k = 16, q = 1
k = 32, q = 0
k = 32, q = 1
1 − (Best/Algorithm)
#Instances
RB
l = 1
l = 2
l = 3
0% 2% 4% 6% 8% 10% 12% 14% 16% 18% 20%
0% 2% 4% 6% 8% 10% 12%
3 6 9 12 15 18 21 24 27

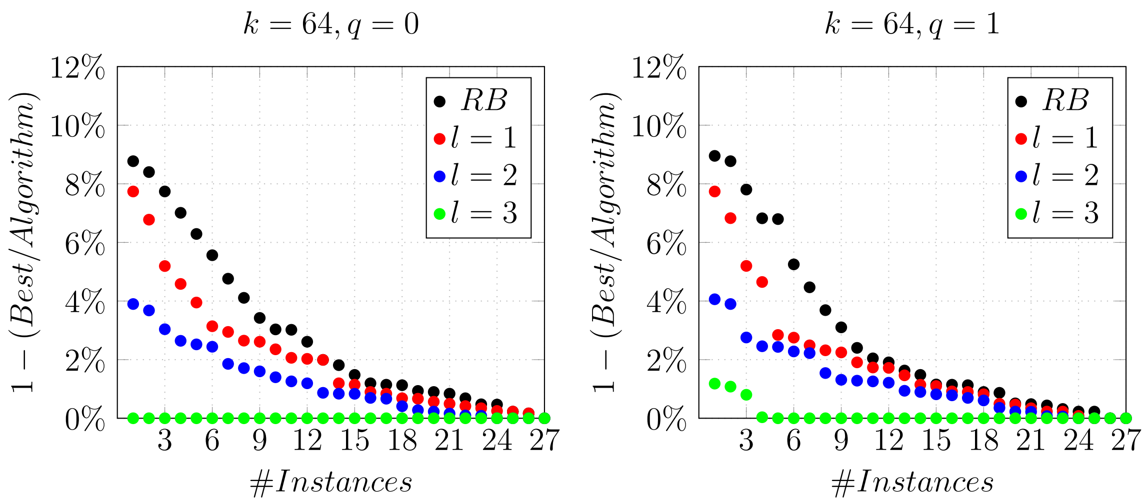


The results presented here clearly demonstrate that with $q = 0$ BRB always matches RB's result. The difference was most noticeable for large social networks with $k = 8$ and $q = 0$. For example, $wiki - Talk$ saw an improvement of 20% and $loc - gowalla$ - 10%. BRB achieves a better average cut by at least in 4% in around half the cases for $k \leq 32$. The same can be said for around a third of cases for $k = 64$. It is important to note that for random geometric graphs, of which two are present in the test suite, and for Delaunay triangulations, of which we test two, no improvement was achieved. This is the case because their highly regular structure means that, like with grid graphs, the choice of individual cut does not affect the final result as all cuts that do not violate the balance constraint are very close in weight. Furthermore, the results suggest that in general the larger the value of k, the smaller the improvement BRB yields. We believe this is the case because the values of l we tested were not sufficiently big. For example, if $k = 8$ RB has a recursion depth of 3. With $l = 3$ BRB examines multiple cuts on all of those levels. However, for $k = 64$, RB reaches a depth of 6. Here even with $l = 3$ we examine multiple cuts on only half of RB's recursion levels.

The next plots provide an overview of the best cut calculated by the algorithms. The improvement, although still present, is less noticeable, as a change of 4% is only seen in around a third of cases. Moreover, the improvement is below 3% in more than half of cases. Again BRB yielded no improvement on the geometric graphs and large social networks saw their results change the most, as for example $wiki - Talk$'s best cut improved by around 17% and $loc - gowalla$'s - by 9% for $k = 8$ and $q = 0$.



$k = 8, q = 0$
$k = 8, q = 1$
$k = 16, q = 0$
$k = 16, q = 1$
$k = 32, q = 0$
$k = 32, q = 1$
$1 - (Best/Algorithm)$
#Instances
RB
$l = 1$
$l = 2$
$l = 3$
0% 2% 4% 6% 8% 10% 12% 14% 16% 18% 20%
3 6 9 12 15 18 21 24 27



It is also important to note that the running time of BRB is much worse than RB's. As seen in Tables 4.1 through 4.8, each new recursion level increases the running time by a factor of around $|S| \cdot (2 \cdot q + 1)$. This is the case because on each level BRB examines $|S|$ individual cuts and follows their recursion trees to their end. Furthermore, every individual cut is rebalanced a maximum of $2 \cdot q + 1$ times, in order to test the deviation in each direction.

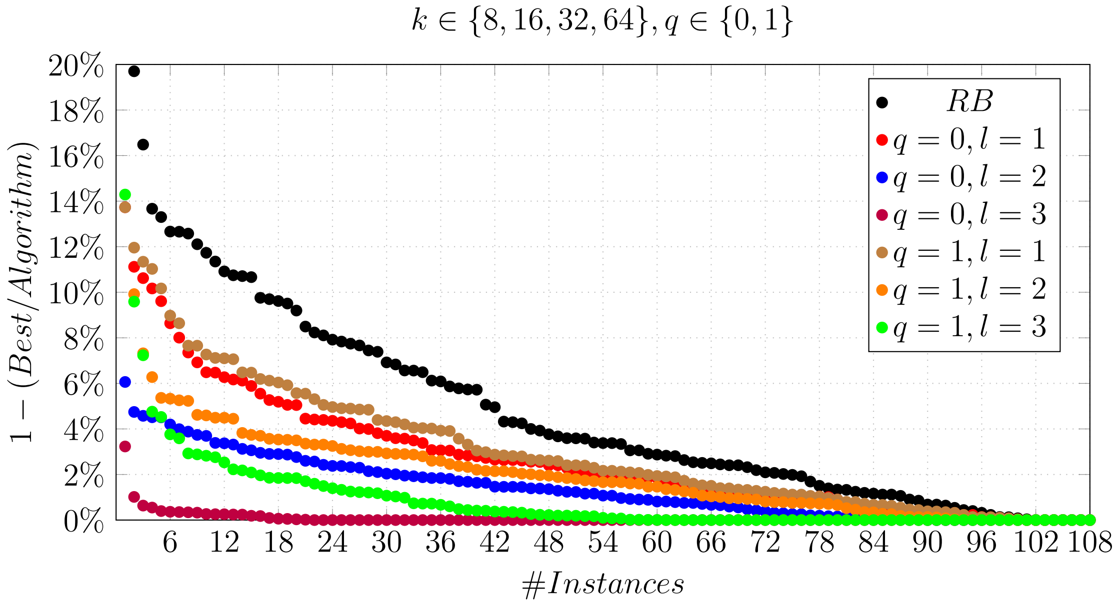


Figure 4.2: Average cut comparison with different tuning parameters

As is evident from Figure 4.2, BRB's best preconfiguration regarding the average cut is $l = 3, q = 0$. It achieves an improvement of at least 4% in around 40% of cases, a ratio which rises to almost 50% if we exclude the geometric instances where no improvement

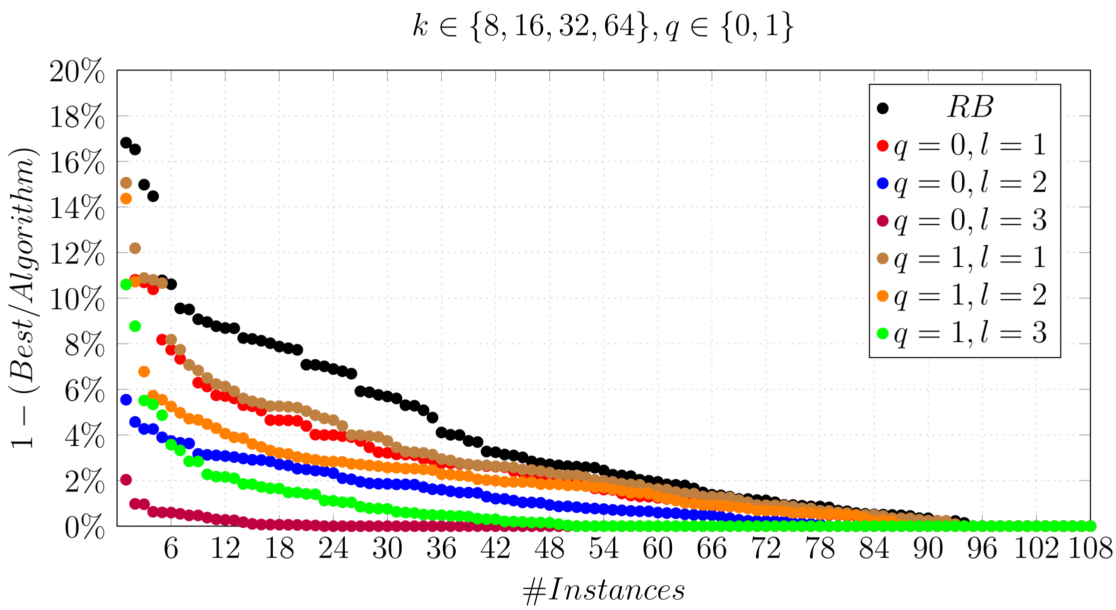


Figure 4.3: Best cut comparison with different tuning parameters

is to be expected. These results also indicate that setting the deviation parameter q to a value greater than 0 worsens the result in almost all cases due to its greedy nature, while simultaneously increasing the running time by a factor close to 3. Furthermore, it only provides an improvement of at least 1% in an extremely small number of cases - less than 1%.

Figure 4.3 illustrates that again the $l = 3, q = 0$ configuration provides the best results regarding the minimal cut obtained on a certain instance-k pair. However, an improvement of 4% is only seen in around a third of cases here. Indeed, for about a quarter of all cases the improvement is less than 1%. It is also important to note that setting the deviation parameter q to 1 once again worsens the overall result, thus reinforcing the suspicion that a greedy cut selection at any stage of the algorithm leads to poorer performance.

5 Discussion

5.1 Conclusion

This thesis presents a new variant of the recursive bisection algorithm. We propose a novel approach, namely examining multiple cuts on each recursion level, following each new recursion tree to its end, and choosing the best one and implemented it in the $KaHIP$ framework.

We investigate the performance of our new algorithm in comparison to the already existing basic recursive bisection. We continue by fine tuning the algorithm's parameters. Our results show BRB performs best for smaller values of k on large social networks. BRB produces an improvement of around 4% in ca. 40% of test cases when it comes to the average cut and improves only about a third of test cases by 4% when it comes to the minimal cut. However, this comes at the cost of a running time several orders of magnitude worse than RB's. Furthermore, the tuning parameter q, responsible for greedily rebalancing each individual bisection cut, not only increases the running time by a factor of $2 \cdot q + 1$, but also worsens the result in the vast majority of test cases.

5.2 Future Work

These findings suggest that there is progress to be made in a recursive bisection model with a larger initial imbalance. An interesting direction of investigation would be the imbalance selected at each recursion level. Developing a heuristic so that only the likely optimal value for it is tested, instead of the many more our algorithm examines, would reduce the running time by likely as many as l orders of magnitude, possibly coming very close to the running time of RB.

Furthermore, the ability to accurately foretell what value of q would be optimal for each individual cut, further reducing the running time by a factor ≥ 2, would be a great benefit. This, however, would require global knowledge at the time of each bisection, making it far less feasible than other research directions. It would also be beneficial to find concrete parameter configurations well suited to specific graph families.

A Detailed Experimental Results

A.1 K = 8

Graph	Param.	Avg. Cut	Best Cut	Worst Cut	Avg.Bal.	Worst Bal.	Time (s)
			Walshaw Graphs				
bcsstk29	RB	16 787	15 868	18 112	1,01	1,01	0,82
	$l = 1$	14 971	14 854	15 170	1,01	1,01	9,04
	$l = 2$	14 833	14 691	15 092	1,01	1,01	103,79
	$l = 3$	14 661	14 618	14 716	1,01	1,01	430,29
4elt	RB	647	618	690	1,01	1,01	0,26
	$l = 1$	618	604	633	1,01	1,01	4,35
	$l = 2$	591	584	595	1,00	1,00	31,11
	$l = 3$	578	567	584	1,00	1,01	165,47
fe_sphere	RB	1 461	1 371	1 518	1,01	1,01	0,27
	$l = 1$	1 311	1 302	1 317	1,00	1,00	3,79
	$l = 2$	1 293	1 286	1 296	1,00	1,00	32,58
	$l = 3$	1 271	1 267	1 274	1,00	1,00	200,42
cti	RB	2 345	2 203	2 454	1,01	1,01	0,32
	$l = 1$	2 183	2 165	2 203	1,01	1,01	4,41
	$l = 2$	1 949	1 886	1 997	1,01	1,01	36,45
	$l = 3$	1 902	1 840	1 972	1,01	1,01	173,59
memplus	RB	13 389	13 196	13 708	1,01	1,01	0,34
	$l = 1$	13 195	13 124	13 264	1,02	1,03	5,08
	$l = 2$	13 076	13 014	13 183	1,01	1,03	33,88
	$l = 3$	13 056	13 014	13 132	1,01	1,03	210,41
cs4	RB	1 773	1 730	1 815	1,00	1,01	0,43
	$l = 1$	1 773	1 730	1 815	1,00	1,01	7,12
	$l = 2$	1 750	1 730	1 790	1,00	1,00	53,32
	$l = 3$	1 748	1 724	1 790	1,01	1,01	290,94

Table A.1: $k = 8$, $q = 0$, Machine B

Graph	Param.	Avg. Cut	Best Cut	Worst Cut	Avg.Bal.	Worst Bal.	Time (s)
Walshaw Graphs							
pwt	RB	1 605	1 522	1 671	1,01	1,01	0,62
	$l = 1$	1 486	1 461	1 514	1,01	1,01	7,87
	$l = 2$	1 471	1 461	1 477	1,00	1,01	57,73
	$l = 3$	1 470	1 461	1 475	1,00	1,01	173,03
bcsstk32	RB	23 337	22 804	23 703	1,01	1,01	2,41
	$l = 1$	23 337	22 804	23 703	1,01	1,01	56,00
	$l = 2$	22 700	22 204	23 093	1,01	1,01	448,10
	$l = 3$	22 499	22 204	22 736	1,03	1,08	2 265,65
fe_body	RB	1 261	1 125	1 404	1,01	1,01	0,78
	$l = 1$	1 136	1 099	1 171	1,03	1,05	8,88
	$l = 2$	1 102	1 061	1 133	1,01	1,01	76,54
	$l = 3$	1 102	1 061	1 133	1,01	1,01	413,04
t60k	RB	556	498	625	1,01	1,01	0,91
	$l = 1$	556	498	625	1,01	1,01	13,93
	$l = 2$	556	498	625	1,01	1,01	102,09
	$l = 3$	556	498	625	1,01	1,01	466,97
wing	RB	3 130	3 122	3 137	1,00	1,00	1,41
	$l = 1$	3 064	2 932	3 137	1,00	1,01	21,89
	$l = 2$	3 015	2 851	3 137	1,00	1,01	178,27
	$l = 3$	3 015	2 851	3 137	1,00	1,01	973,86
rotor	RB	14 560	13 898	15 726	1,01	1,01	4,56
	$l = 1$	13 931	13 837	14 057	1,01	1,01	82,19
	$l = 2$	13 870	13 758	14 037	1,00	1,01	623,67
	$l = 3$	13 669	13 615	13 711	1,01	1,01	3 448,35
Social Network Graphs							
p2p-Gnutella04	RB	15 637	15 345	15 936	1,01	1,01	0,45
	$l = 1$	15 484	15 345	15 558	1,01	1,01	5,18
	$l = 2$	15 430	15 345	15 506	1,00	1,00	37,31
	$l = 3$	15 430	15 345	15 506	1,00	1,00	221,50
word-association-2011	RB	28 062	27 674	28 528	1,01	1,01	0,55
	$l = 1$	27 557	27 080	27 916	1,01	1,01	6,70
	$l = 2$	27 269	26 998	27 406	1,01	1,01	49,45
	$l = 3$	27 269	26 998	27 406	1,01	1,01	307,06

Table A.2: $k = 8$, $q = 0$, Machine B

Graph	Param.	Avg. Cut	Best Cut	Worst Cut	Avg.Bal.	Worst Bal.	Time (s)
			Social Network Graphs				
PGPgiant-compo	RB	1 067	1 045	1 094	1,01	1,01	0,17
	$l = 1$	1 067	1 045	1 094	1,01	1,01	2,49
	$l = 2$	1 056	1 036	1 087	1,01	1,01	18,29
	$l = 3$	1 056	1 036	1 087	1,01	1,01	108,72
as-22july06	RB	11 810	11 422	12 381	1,01	1,01	0,82
	$l = 1$	11 329	11 291	11 353	1,01	1,01	12,81
	$l = 2$	11 212	11 129	11 317	1,01	1,01	108,98
	$l = 3$	11 212	11 129	11 317	1,01	1,01	650,86
soc-Slashdot-0902	RB	224 536	216 607	237 725	1,01	1,01	3,27
	$l = 1$	201 421	200 478	202 931	1,01	1,01	43,36
	$l = 2$	199 272	198 991	199 679	1,01	1,01	344,29
	$l = 3$	199 053	198 991	199 154	1,01	1,01	2 279,64
loc-brightkite	RB	51 370	50 186	52 482	1,01	1,01	1,87
	$l = 1$	50 146	49 541	50 711	1,01	1,01	25,80
	$l = 2$	49 636	49 517	49 849	1,01	1,01	186,65
	$l = 3$	48 825	48 609	49 067	1,01	1,01	1 146,96
enron	RB	53 998	51 476	56 459	1,01	1,01	1,53
	$l = 1$	52 817	51 426	54 508	1,01	1,01	22,82
	$l = 2$	49 471	46 913	51 013	1,01	1,01	176,73
	$l = 3$	48 783	46 881	50 487	1,01	1,01	1 163,85
finan512	RB	675	648	729	1,00	1,00	1,30
	$l = 1$	648	648	648	1,00	1,00	20,32
	$l = 2$	648	648	648	1,00	1,00	156,11
	$l = 3$	648	648	648	1,00	1,00	557,85
loc-gowalla	RB	200 376	196 968	202 893	1,02	1,02	9,56
	$l = 1$	191 028	188 144	193 672	1,01	1,01	172,17
	$l = 2$	184 207	183 004	185 028	1,01	1,01	1 070,63
	$l = 3$	181 109	180 793	181 341	1,01	1,01	5 917,97
coAuthors-Citeseer	RB	49 085	48 451	49 459	1,01	1,01	7,51
	$l = 1$	49 085	48 451	49 459	1,01	1,01	131,79
	$l = 2$	48 869	47 943	49 346	1,01	1,01	888,06
	$l = 3$	48 869	47 943	49 346	1,01	1,01	4 772,32
wiki-Talk	RB	503 217	477 251	527 752	1,02	1,02	238,45
	$l = 1$	451 287	416 213	516 260	1,01	1,01	2 069,32
	$l = 2$	408 799	399 358	416 213	1,01	1,01	10 486,50
	$l = 3$	401 112	396 980	406 997	1,01	1,01	35 528,70

Table A.3: $k = 8$, $q = 0$, Machine B

Graph	Param.	Avg. Cut	Best Cut	Worst Cut	Avg.Bal.	Worst Bal.	Time (s)
				Geometric Graphs			
rgg17	RB	2 288	2 189	2 460	1,01	1,01	3,53
	$l = 1$	2 288	2 189	2 460	1,01	1,01	46,51
	$l = 2$	2 288	2 189	2 460	1,01	1,01	468,29
	$l = 3$	2 288	2 189	2 460	1,01	1,01	2 289,94
delaunay17	RB	2 520	2 498	2 559	1,01	1,01	2,43
	$l = 1$	2 520	2 498	2 559	1,01	1,01	50,31
	$l = 2$	2 516	2 491	2 559	1,01	1,01	430,40
	$l = 3$	2 489	2 417	2 559	1,01	1,01	1 796,36
rgg18	RB	3 573	3 412	3 843	1,01	1,01	9,74
	$l = 1$	3 573	3 412	3 843	1,01	1,01	115,05
	$l = 2$	3 573	3 412	3 843	1,01	1,01	1 006,69
	$l = 3$	3 573	3 412	3 843	1,01	1,01	4 674,62
delaunay18	RB	3 523	3 444	3 606	1,01	1,01	6,36
	$l = 1$	3 523	3 444	3 606	1,01	1,01	145,03
	$l = 2$	3 515	3 444	3 606	1,01	1,01	1 200,93
	$l = 3$	3 515	3 444	3 606	1,01	1,01	6 495,50

Table A.4: $k = 8$, $q = 0$, Machine B

A.2 K = 64

Graph	Param.	Avg. Cut	Best Cut	Worst Cut	Avg.Bal.	Worst Bal.	Time (s)
			Walshaw Graphs				
bcsstk29	RB	65 847	64 907	66 619	1,02	1,02	4,27
	$l = 1$	62 827	62 189	63 554	1,01	1,01	46,90
	$l = 2$	62 057	61 498	62 633	1,01	1,01	455,41
	$l = 3$	60 739	60 357	61 124	1,01	1,01	4 127,38
4elt	RB	3 137	3 114	3 165	1,01	1,01	1,22
	$l = 1$	3 017	3 011	3 030	1,02	1,05	16,67
	$l = 2$	2 971	2 963	2 985	1,01	1,02	138,98
	$l = 3$	2 883	2 873	2 889	1,02	1,04	1 200,33
fe_sphere	RB	4 330	4 287	4 363	1,01	1,01	1,22
	$l = 1$	4 277	4 243	4 300	1,01	1,01	15,34
	$l = 2$	4 218	4 216	4 220	1,01	1,01	139,18
	$l = 3$	4 160	4 157	4 166	1,01	1,01	1 240,99
cti	RB	7 182	7 059	7 250	1,04	1,05	1,54
	$l = 1$	6 963	6 936	7 008	1,02	1,02	18,92
	$l = 2$	6 751	6 713	6 793	1,01	1,02	169,78
	$l = 3$	6 481	6 466	6 508	1,01	1,02	1 483,80
memplus	RB	18 012	17 577	18 529	1,03	1,03	1,25
	$l = 1$	17 774	17 577	17 888	1,03	1,04	19,17
	$l = 2$	17 658	17 457	17 828	1,03	1,04	126,73
	$l = 3$	17 560	17 457	17 694	1,03	1,04	1 026,80
cs4	RB	4 827	4 797	4 877	1,01	1,01	1,91
	$l = 1$	4 827	4 797	4 877	1,01	1,01	26,26
	$l = 2$	4 804	4 794	4 821	1,01	1,01	223,52
	$l = 3$	4 773	4 754	4 795	1,01	1,01	1 864,36
pwt	RB	9 578	9 555	9 600	1,01	1,02	2,73
	$l = 1$	9 414	9 395	9 449	1,01	1,01	29,33
	$l = 2$	9 293	9 248	9 349	1,01	1,01	265,33
	$l = 3$	9 164	9 100	9 210	1,01	1,02	2 347,24
bcsstk32	RB	114 389	112 285	115 832	1,02	1,02	12,24
	$l = 1$	111 535	110 988	112 197	1,02	1,02	184,95
	$l = 2$	108 287	108 080	108 477	1,02	1,02	1 670,81
	$l = 3$	105 641	105 224	106 062	1,01	1,02	14 217,60

Table A.5: $k = 64$, $q = 0$, Machine B

Graph	Param.	Avg. Cut	Best Cut	Worst Cut	Avg.Bal.	Worst Bal.	Time (s)
Walshaw Graphs							
fe_body	RB	5 599	5 403	5 784	1,03	1,04	3,76
	$l = 1$	5 450	5 394	5 504	1,03	1,05	37,15
	$l = 2$	5 359	5 315	5 394	1,06	1,09	310,52
	$l = 3$	5 231	5 181	5 268	1,07	1,10	2 736,76
t60k	RB	2 501	2 463	2 568	1,03	1,06	3,67
	$l = 1$	2 501	2 463	2 568	1,03	1,06	47,73
	$l = 2$	2 501	2 463	2 568	1,03	1,06	411,90
	$l = 3$	2 497	2 463	2 558	1,03	1,06	3 652,48
wing	RB	9 354	9 262	9 467	1,01	1,01	5,47
	$l = 1$	9 354	9 262	9 467	1,01	1,01	83,00
	$l = 2$	9 340	9 220	9 467	1,01	1,01	679,33
	$l = 3$	9 213	9 156	9 314	1,01	1,01	5 525,04
rotor	RB	52 907	52 589	53 396	1,02	1,03	15,57
	$l = 1$	52 907	52 589	53 396	1,02	1,03	231,77
	$l = 2$	52 535	52 497	52 590	1,02	1,03	1 975,28
	$l = 3$	51 289	51 216	51 414	1,03	1,06	15 483,80
Social Network Graphs							
p2p-Gnutella04	RB	20 123	20 101	20 168	1,01	1,02	1,68
	$l = 1$	20 078	20 057	20 091	1,01	1,01	19,16
	$l = 2$	20 054	20 024	20 072	1,01	1,01	140,95
	$l = 3$	20 015	20 007	20 024	1,01	1,01	1 157,67
word-association-2011	RB	38 936	38 803	39 161	1,01	1,02	2,13
	$l = 1$	38 804	38 779	38 830	1,01	1,02	40,93
	$l = 2$	38 795	38 777	38 830	1,02	1,02	215,84
	$l = 3$	38 672	38 618	38 716	1,02	1,02	1 694,20
PGPgiant-compo	RB	3 115	3 081	3 174	1,02	1,03	0,80
	$l = 1$	3 090	3 060	3 119	1,03	1,05	11,47
	$l = 2$	3 059	3 040	3 090	1,03	1,05	88,04
	$l = 3$	3 009	2 988	3 023	1,04	1,05	759,85
as-22july06	RB	20 769	20 637	20 885	1,02	1,02	4,54
	$l = 1$	20 485	20 399	20 556	1,02	1,02	60,62
	$l = 2$	20 485	20 399	20 556	1,02	1,02	497,47
	$l = 3$	20 349	20 263	20 393	1,02	1,03	4 512,81

Table A.6: $k = 64$, $q = 0$, Machine B

Graph	Param.	Avg. Cut	Best Cut	Worst Cut	Avg.Bal.	Worst Bal.	Time (s)
Walshaw Graphs							
soc-Slashdot-0902	RB	303 174	301 872	304 895	1,03	1,03	9,93
	$l = 1$	301 470	300 754	302 128	1,02	1,04	115,48
	$l = 2$	300 288	299 888	300 490	1,01	1,02	874,88
	$l = 3$	299 783	299 057	300 421	1,01	1,02	6 946,35
loc-brightkite	RB	77 250	76 259	77 970	1,02	1,02	7,66
	$l = 1$	77 250	76 259	77 970	1,02	1,02	474,44
	$l = 2$	76 503	76 259	76 953	1,02	1,02	1 131,36
	$l = 3$	75 764	75 620	75 979	1,02	1,02	6 544,44
enron	RB	102 875	101 976	103 449	1,03	1,03	7,05
	$l = 1$	101 343	101 164	101 463	1,02	1,02	104,54
	$l = 2$	100 626	99 741	101 281	1,05	1,10	765,93
	$l = 3$	98 747	98 485	99 083	1,03	1,04	6 638,68
finan512	RB	12 669	12 324	13 017	1,01	1,01	6,29
	$l = 1$	12 382	12 186	12 556	1,01	1,01	79,15
	$l = 2$	11 751	11 699	11 844	1,01	1,02	690,11
	$l = 3$	11 319	11 243	11 395	1,01	1,01	5 846,15
loc-gowalla	RB	339 685	333 368	345 802	1,03	1,03	38,25
	$l = 1$	336 378	333 368	338 048	1,02	1,03	1 564,13
	$l = 2$	335 038	333 368	336 405	1,02	1,03	4 937,20
	$l = 3$	330 150	329 390	331 536	1,02	1,03	30 033,10
coAuthors-Citeseer	RB	73 319	73 054	73 556	1,02	1,02	27,78
	$l = 1$	73 319	73 054	73 556	1,02	1,02	958,72
	$l = 2$	73 319	73 054	73 556	1,02	1,02	3 748,08
	$l = 3$	73 261	72 888	73 549	1,02	1,02	23 700,20
wiki-Talk	RB	$1 \cdot 10^{+06}$	$1 \cdot 10^{+06}$	$1 \cdot 10^{+06}$	1,03	1,03	743,30
	$l = 1$	984 919	980 325	989 849	1,03	1,03	4 894,33
	$l = 2$	969 691	961 020	976 980	1,02	1,03	26 682,40
	$l = 3$	962 414	960 107	964 992	1,02	1,03	120 520,00

Table A.7: $k = 64$, $q = 0$, Machine B

Graph	Param.	Avg. Cut	Best Cut	Worst Cut	Avg.Bal.	Worst Bal.	Time (s)
			Geometric Graphs				
rgg17	RB	8 850	8 582	9 165	1,01	1,02	15,49
	$l = 1$	8 627	8 497	8 803	1,01	1,02	170,06
	$l = 2$	8 517	8 455	8 582	1,01	1,01	1 755,00
	$l = 3$	8 517	8 455	8 582	1,01	1,01	15 486,30
delaunay17	RB	9 484	9 437	9 512	1,01	1,01	12,23
	$l = 1$	9 484	9 437	9 512	1,01	1,01	177,63
	$l = 2$	9 424	9 330	9 504	1,01	1,01	1 547,29
	$l = 3$	9 286	9 249	9 351	1,02	1,02	12 411,30
rgg18	RB	13 582	13 561	13 600	1,02	1,02	37,17
	$l = 1$	13 547	13 454	13 600	1,02	1,02	421,35
	$l = 2$	13 531	13 408	13 600	1,02	1,02	4 010,83
	$l = 3$	13 496	13 408	13 586	1,02	1,02	33 933,80
delaunay18	RB	13 301	13 151	13 541	1,01	1,02	26,36
	$l = 1$	13 301	13 151	13 541	1,01	1,02	459,12
	$l = 2$	13 301	13 150	13 541	1,01	1,02	3 720,86
	$l = 3$	13 182	13 129	13 247	1,02	1,02	29 376,80

Table A.8: $k = 64$, $q = 0$, Machine B

Bibliography

[1] Chris walshaw's graph partitioning archive. http://chriswalshaw.co.uk/partition/. Accessed: 24/08/2017.

[2] Kahip v2.00 – karlsruhe high quality partitioning user guide. http://algo2.iti.kit.edu/schulz/software_releases/kahipv2.00.pdf/. Accessed: 07/09/2017.

[3] C. J. Alpert and A. B. Kahng. Recent directions in netlist partitioning: A survey. *Integration, the VLSI Journal*, 19(1-2):1–81, 1995.

[4] K. Andreev and H. Räcke. Balanced graph partitioning. *Theory of Computing Systems*, 39(6):929–939, 2006.

[5] N. Apollonio, R.I. Becker, I. Lari, F. Ricca, and B. Simeonec. Bicolored graph partitioning, or: gerrymandering at its worst. *Discrete Applied Mathematics*, 157:3601–3614, 2009.

[6] M. Armbruster, M. Fuügenschuh, C. Helmberg, and A. Martin. A comparative study of linear and semidefinite branch-and-cut methods for solving the minimum graph bisection problem. In *13th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, volume 5035 of LNCS, pages 112 – 124. Springer, 2008.

[7] Aydin Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. abs/1311.3144, 2013.

[8] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck. Customizable route planning. In *Proceedings of the 10th International Symposium on Experimental Algorithms*, volume 6630 of LCNS, pages 376 – 387. Springer, 2011.

[9] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Conference on Design Automation*, pages 175 – 181, 1982.

[10] M. Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Mathematical Journal*, 25(4):619–633, 1975.

[11] I. C. M. Flinsenberg. *Route planning algorithms for car navigation*. PhD thesis, Eindhoven University of Technology, January 2004.

[12] Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin-Cummings Publishing Co. Inc., Redwood City, CA, USA, 1994.

[13] L. Hyafil and R. Rivest. *Graph partitioning and constructing optimal decision trees are polynomial complete problems*. IRIA-Laboria, Rocquencourt, France, 1973.

[14] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(1):291–307, 1970.

[15] Jin Kim, Inwook Hwang, Yong-Hyuk Kim, and Byung Ro Moon. Distributed evolutionary graph partitioning. In *12th Workshop on Algorithm Engineering and Experimentation (ALENEX)*, pages 16 – 29, 2012.

[16] Sebastian Schlag, Vitali Henne, Tobias Heuer, Henning Meyerhenke, Peter Sanders, and Christian Schulz. k-way hypergraph partitioning via n-level recursive bisection. -, 2016.

[17] K. Schloegel, G. Karypis, and V. Kumar. Graph partitioning for high performance scientific simulations. *The Sourcebook of Parallel Computing*, pages 491–541, 2003.

[18] Christian Schulz. *High Quality Graph Partitioning*. PhD thesis, Karlsruhe Institute of Thechnology, 2013.

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