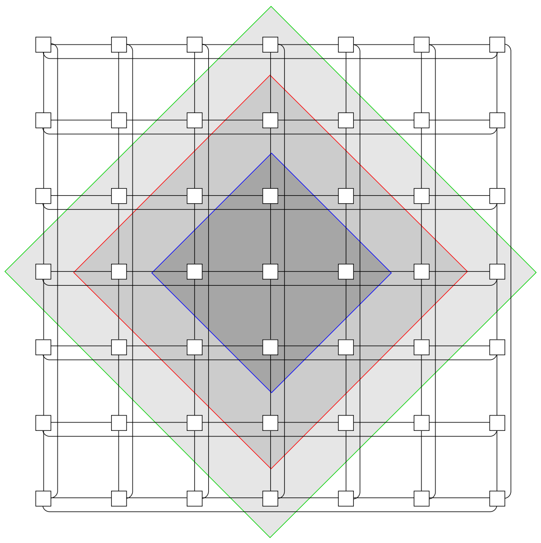
Lower Bounds and Exact Algorithms for the Graph Partitioning Problem using Multicommodity Flows



Dissertation

von

Norbert Sensen

Schriftliche Arbeit zur Erlangung des Grades
eines Doktors der Naturwissenschaften

Universität Paderborn,
Fakultät für Elektrotechnik, Informatik und Mathematik
Institut für Informatik

Paderborn, 15. April 2003

Danksagungen

Zu aller erst möchte ich mich bei Prof. Dr. Burkhard Monien bedanken. Durch seine Unterstützung und Kommentare an den richten Stellen wurde ich bei der Erstellung der vorliegenden Arbeit immer wieder gefordert und motiviert. Zusätzlich hatte ich als wissenschaftlicher Mitarbeiter seiner Arbeitsgruppe jederzeit den Freiraum, der für die Erstellung einer solchen Arbeit nötig ist.

Desweiteren möchte ich mich bei allen aktuellen und ehemaligen Mitgliedern der Arbeitsgruppe bedanken. Es herrschte immer ein sehr freundschaftliches und produktives Arbeitsklima, so dass ich einerseits bei inhaltlichen Fragestellungen immer einen Diskussionspartner hatte und andererseits auch Freundschaften entstehen konnten. Namentlich nennen möchte ich an dieser Stelle Meinolf Sellmann, Torsten Fahle, Robert Preis, Robert Elsässer, Manuel Rode und Thomas Decker.

Zu Dank verpflichtet bin ich auch Geraldine Brehony, die mit viel Geduld alle meine typisch deutschen Sprachfehler aus der Arbeit heraussuchte und korrigierte.

Besonders zu Dank verpflichtet bin ich meiner Frau Britta, die mich immer unterstütze und den Rückhalt gab, der während der Erstellung einer solchenArbeit notwendig ist. An dieser Stelle möchte ich auch meine Eltern danken, die mir die Möglichkeit gaben, Informatik zu studieren, und die mich jederzeit unterstützten.

Norbert

— DANKE —

Contents

List of Figures, Tables and Algorithms

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Graph partitioning problems occur in a wide range of applications. The task in hand is to divide the set of vertices of a graph into a given number of parts such that the number of edges with endpoints in different sets is minimized.

Applications

The efficient use of parallel or *distributed processor systems* is a major application of graph partitioning problems. The work which has to be computed is often subdivided into a number of small tasks which are of different difficulty. These tasks have to communicate with each other in order to be solvable. This can be modeled as a graph: the vertices model the tasks of the graphs with weights corresponding to their difficulty and the edges model the required communication between the tasks. Now, in order to have an efficient parallelization, the tasks have to be assigned to the different processors. This assignment corresponds to a graph partitioning problem since on the one hand every processor should get the same amount of tasks and on the other hand communication between different processors is expensive. A interesting survey on this application of graph partitioning is given in [HK00].

Another very important application of graph partitioning problems arises within the area of *Finite Element Methods* (FEM). There, numerical simulations are computed for problems such as crash-simulations, computational fluid dynamics, weather forecasts or earthquake simulations. For the efficient computation of these simulations parallel systems are needed. The single elements have to be assigned to the different processors. Every processor should get an equal part of the elements. However dependencies between the elements exist and dependencies of elements on different processors are expensive. A recent survey on this application is given in [SKK00].

Within the area of *integrated circuit layout* several hundred thousand transistors have to be placed with their connections. These placements imply a partitioning of the transistors and of course it is important that there are relatively few connections between the different parts of the circuit. Therefore, among many other problems, typical graph partitioning problems have to be solved. A survey on this topic is given in [Len90].

One last application of graph partitioning problems which we want to mention here is the computation of *sparse-matrix orderings.* If large sparse systems of linear equations (e.g. linear programs) have to be solved, the computation of fill-reducing orderings speeds-up the calculation. Finding optimal orderings is an NP-hard problem, so heuristics are used. One very promising heuristic is based on graph partitioning: The sparse matrix is regarded as the adjacency matrix of a graph. Therefore, the ordering of the rows such that adjacent rows stay together can be carried out using graph partitioning. More details are given in [Gup97] for example.

Complexity

Due to the importance of graph partitioning problems, they were extensively examined. Unfortunately, even simplest versions are NP-hard: The problem of the division of the vertices into an arbitrary number of sets with at most M vertices per set is NP-hard [HR73], even when $M = 3$. If $M = 2$, the problem is equivalent to the maximum matching problem and can be solved in time $O(\sqrt{n}m)$ where n is the number of vertices and m is the number of edges [MV80]. At the other end of graph partitioning problems, there is the graph bisection problem, which involves the problem of dividing the vertices into two equally sized sets. This is also NP-hard[GJS76]. It is shown in [BCLS87] that this also holds for regular graphs. Furthermore, the division of the vertices into two sets where each set has at most αn vertices, with $\alpha \in [0.5, 1)$ constant, is also NP-hard. The two-partitioning becomes solvable in polynomial time, if one set has at most k vertices, for any constant k.

It is an open question if the graph bisection problem is NP-hard or solvable in polynomial time if the graph G is planar. It is also unknown if there is a PTAS for the graph bisection problem. The best approximation algorithm provides us with a ratio of $O(\log^2 n)$ from the optimum [FK02].

Solutions

Due to the NP-hardness of graph partitioning problems on the one hand and its importance on the other hand, a lot of work has been carried out on the development of effective and fast heuristical algorithms for solving the problem. A survey of different methods is given in [Fjä98]. Furthermore, there are a number of libraries which heuristically solve graph partitioning problems, e.g. Chaco [HL94, HL95b, HL95a], JOS-

TLE [WCE95, WCM00, WCE97], METIS [KK98a, KK98b], PARTY [PD97, Pre98] or SCOTCH [PR96].

In comparison with these heuristical approaches, relatively little work has been done in order to exactly solve graph partitioning problems. In Section 1.5 we give an overview of the most important or successful approaches. Due to the NP-hardness of the problem, the exact methods can only handle relatively small graphs, while the heuristics deal with thousands or even millions of vertices. Nevertheless, the exact methods and lower bounds are very important for the verification of heuristics. They can be used in applications with a small number of vertices and finally, their development gives an insight into the difficulties of the problem.

1.2 Overview of this work

The main theme of this work is the presentation of new lower bounds on graph partitioning problems and their use inside an algorithm which exactly solves graph partitioning problems. These new lower bounds are based on multicommodity-flows and can be viewed as a generalization of the known Leighton-bound. In the remainder of this chapter, we will formally define graph partitioning problems and present the best known lower bounds and their exact approaches.

In Chapter 2, we will present the new bounds and our experiments which will show that these bounds are often superior when compared to other lower bounds. In Chapter 3 we will provide theoretical analyses of the new bounds. I.e. we will show that symmetries inside the graphs can be utilized when calculating the bounds. Furthermore, we will present the bounds for several well known graphs, e.g. tori and hypercubes. This will show that the bounds are not only useful for practical computations but that they can also be used for theoretical analyses.

In Chapter 4, we will present three different methods for the computation of the lower bounds. We will show that a new approximation algorithm is superior to linear programming and cost-decomposition-based algorithms. Finally, in Chapter 5, we will present a branch&bound algorithm which is based on the new lower bounds. Experiments will show that this algorithm is often superior to other algorithms and that we were able to solve problems quickly which were, to our knowledge, practically unsolvable until now.

1.3 Definitions

The main topic of this work is the graph partitioning problem. In literature there are many different definitions of this problem, so in the following we will present the definition which we are working on.

The most well-known version of the graph partitioning problem is the simple graph bisection problem: Given a graph G, its set of vertices has to be partitioned into two equally sized sets such that the number of edges which are adjacent to vertices in different sets is minimized. This minimum number of edges is called the bisection width of the graph. More formally:

Definition 1.1
Let $G = (V,E)$ be an undirected graph with vertices V and edges E. The **graph bisection problem bisGP=(G)** *is the problem of calculating the* **bisection width** $bw(G)$ *with*

$$bw(G) := \min_{U \subseteq V, |U| = \lceil \frac{|V|}{2} \rceil} |\{\{v,w\} \in E | v \in U, w \notin U\}|.$$

If vertex- and edge-weights are given, i.e. $G = (V,E,g,f)$ with $g : V \to \mathbb{R}_{\geq 0}$ and $f : E \to \mathbb{R}_{\geq 0}$, we use $N := \sum_{v \in V} g(v)$ and $N_1 := \sum_{v \in V_1} g(v)$ and the definition can be generalized as follows:

$$bw(G) := \min_{V_1 \subseteq V; N_1 = \lceil \frac{N}{2} \rceil} \sum_{\{v,w\} \in E; v \in V_1 \wedge w \notin V_1} f(\{v,w\})$$

If vertex-weights are given, the situation can occur that no bisection which fulfills the balance-constraint $N_1 = \lceil \frac{N}{2} \rceil$ exists. In practice, a less strict balance-constraint is often given in this case. These cases are covered in this work by the general graph partitioning problem which will be defined in Definition 1.3.

The next more general version of the graph partitioning problem is the k-partitioning problem. I.e. the set of vertices is not only partitioned into two sets but into k sets:

Definition 1.2
Let $G = (V,E)$ be an undirected graph, $k \in \mathbb{N}$ and Π is the set of all correct partitions, i.e. $\pi \in \Pi :\Leftrightarrow \pi : V \to \{1,\ldots k\}$ with $V_i := \{v \in V | \pi(v) = i\}$ and $\forall i : |V_i| \leq \lceil \frac{|V|}{k} \rceil$. We use

$$\forall \pi \in \Pi : \quad cut(\pi, G) := |\{\{v,w\} \in E | \pi(v) \neq \pi(w)\}|$$

and

$$cut\text{-}size(G,k) := \min_{\pi \in \Pi} cut(\pi, G).$$

The **k-partitioning problem kGP=(G,k)** *is the problem of calculating the cut-size(G,k).*

If vertex- and edge-weights are given, i.e. $G = (V,E,g,f)$ with $g : V \to \mathbb{R}_{\geq 0}$ and $f : E \to \mathbb{R}_{\geq 0}$, we use $N := \sum_{v \in V} g(v)$ and $\forall i : N_i := \sum_{v \in V_i} g(v)$ and the definition can be generalized to the balance constraint $\forall i : N_i \leq \lceil \frac{N}{k} \rceil$ and

$$cut(\pi, G) := \sum_{\{v,w\} \in E; \pi(v) \neq \pi(w)} f(\{v,w\}).$$

We must point out that a different balance-constraint of the k-partitioning problem is often used. I.e. instead of $\forall i : |V_i| \leq \lceil \frac{|V|}{k} \rceil$ the constraint $\forall i, j : |V_i| - |V_j| \leq 1$ is used. However in our opinion the balance-constraint that we use is more practical, since in a lot of applications the costs are determined by the biggest partition only, for example in the case of load-balancing. Of course, applications can also be found where the other balance-constraint is more realistic.

Finally, the most general version which we deal with allows more unbalanced partitions by giving a maximal size M to every partition:

Definition 1.3
Let $G = (V, E)$ be an undirected graph, $k \in \mathbb{N}$ and $M \in \mathbb{R}_{\geq 0}$. Let Π be the set of all correct partitions, i.e. $\pi \in \Pi \;:\Leftrightarrow\; \pi : V \to \{1, \ldots k\}$ with $V_i := \{v \in V | \pi(v) = i\}$ and $\forall i : |V_i| \leq M$. We use

$$\forall \pi \in \Pi : \quad cut(\pi, G) := |\{\{v, w\} \in E | \pi(v) \neq \pi(w)\}|$$

and

$$cut\text{-}size(G, k, M) := \min_{\pi \in \Pi} cut(\pi, G).$$

*The **graph partitioning problem GP=(G,k,M)** is the problem of calculation the cut-size(G, k, M).*

If vertex- and edge-weights are given, i.e. $G = (V, E, g, f)$ with $g : V \to \mathbb{R}_{\geq 0}$ and $f : E \to \mathbb{R}_{\geq 0}$, we use $N := \sum_{v \in V} g(v)$ and $\forall i : N_i := \sum_{v \in V_i} g(v)$ and the definition can be generalized to the balance constraint $\forall i : N_i \leq M$ and

$$cut(\pi, G) := \sum_{\{v,w\} \in E; \pi(v) \neq \pi(w)} f(\{v, w\}).$$

Obviously, the bisection problem is a special case of the k-partitioning problem (using $k = 2$). And the k-partitioning problem is a special case of the graph partitioning problem (using $M = \lceil \frac{N}{k} \rceil$).

1.4 Known Lower Bounds

Since the graph partitioning problem and particularly the graph bisection problem have been of great interest in the past, several approaches to lower bounds are also known. In the following we present the most important approaches. Since the following bounds cannot be used with the most general graph partition problem *GP*, in this subsection we focus on the k-partitioning problem with edge-weights but without vertex weights.

1.4.1 Classical Spectral Method

Probably the most famous lower bound method for the graph partitioning problem is based on the eigenvalues of the Laplace matrix $L(G)$ of a graph G:

Definition 1.4
Let $G = (V,E)$ be an undirected graph with $|V| = n$. The $n \times n$ Laplace matrix $L(G) = \{l_{v,w}\}$ *is defined as*

$$l_{v,w} := \begin{cases} deg(v) & \text{if } v = w \\ -1 & \text{if } v \neq w \wedge \{v,w\} \in E \\ 0 & \text{else} \end{cases} .$$

A $n \times k$-matrix $Y_\pi = (y_{i,j})$ can be used in order to express a partition π: $y_{v,i} := \begin{cases} 1 & \text{if } \pi(v) = i \\ 0 & \text{else} \end{cases}$.
It becomes clear that a matrix Y corresponds to a correct partition, if and only if $Ye_k = e_n \wedge Y^t e_n = \frac{n}{k} e_k \wedge y_{i,j} \in \{0,1\}$. Then

$$\text{cut-size}(G,k) = \min\left\{\frac{1}{2}Tr(Y^t L(G) Y) \mid Ye_k = e_n \wedge Y^t e_n = \frac{n}{k} e_k \wedge y_{i,j} \in \{0,1\}\right\} \quad (1.1)$$

follows. By relaxing some constraints of Y, lower bounds are obtained. The first one was introduced by Donath and Hoffman [DH73]:

$$DH(G,k) := \min\left\{\frac{1}{2}Tr(Y^t L(G) Y) \mid Y^t Y = \frac{n}{k} I_k\right\}$$

Obviously, $DH(G,k)$ is a lower bound on cut-size(G,k). In the spectral graph theory the following formula from Fan [Fan49] holds:

$$\min_{Y^T Y = I_k} \left\{Tr(Y^T A Y)\right\} = \sum_{j=1}^{k} \lambda_j(A)$$

where $\lambda_j(A)$ is the j-th smallest eigenvalue of a matrix A. So the DH-bound can be calculated with the help of eigenvalues and we have:

$$\text{cut-size}(G,k) \leq DH(G,k) = \frac{n}{2k} \sum_{j=1}^{k} \lambda_j(L(G))$$

Since the Laplace matrices of graphs are a well-studied area of spectral graph theory, see e.g. [CDS79, Chu97], this bound can be used with a mass of theoretical graphs.

In general the DH-bound cannot be used with the overall graph partitioning problem. But when we look at the special case of $k = 2$ and arbitrary M, a bound of

$$DH(G,2,M) = \frac{(n-M)M}{n} \lambda_2(L(G))$$

holds (see e.g. [FRW94, Moh97]).

1.4.2 Improved Spectral Methods

A strengthened variation of the DH-bound for the graph bisection problem was proposed by Boppana in [Bop87]. Rendl and Wolkowicz [RW95] independently proposed the same idea for the k-partitioning problem. In [FRW94] a practical implementation which includes an upper bound, i.e. a feasible partitioning, is given. Please notice, that the years of publications are not in order: [FRW94] is a sequel of [RW95]. Boppana, Rendl, and Wolkowicz use a more restricted relaxation of Equation (1.1):

$$BRW(G,k) := \min\left\{\frac{1}{2}Tr(Y^tL(G)Y) \mid Ye_k = e_n \wedge Y^te_n = \frac{n}{k}e_k \wedge Y^tY = \frac{n}{k}I_k\right\}$$

Again, $BRW(G,k) \leq$ cut-size(G,k) is obvious. Furthermore, $DH(G,k) \leq BRW(G,k)$ is also obvious. The efficient computation of $BRW(G,k)$ is much more complicated. In [RW95] it is shown that $BRW(G,k)$ can be expressed as a formula with a function $s(d)$, which has to be minimized. As is the case in the DH-bound, eigenvalues are also used inside the function $s(d)$. This function $s(d)$ is convex, but possibly non-smooth. In [FRW94] an iterative procedure is used in the minimization process. It starts with an initial solution which is equivalent to the DH-bound and then uses the Bundle Trust method for the optimization process. The Lanczos Algorithm is used for the computation of the needed eigenvalues.

1.4.3 Semidefinite programming

Firstly, Alizadeh [Ali95] has shown that the eigenvalue-bounds can be viewed as dual problems of semidefinite relaxations of graph partitioning. Poljak and Rendl [PR95] derived semidefinite relaxations for the graph bisection problem which yield the same bound as the BRW-bound. In the case of general k-partitioning these equivalences were obtained by Karisch and Rendl [KR98]. We give a small survey on the different formulations and relaxations:

Modeling k-partitions

The notation in this part follows [KR98]. The k-partitioning problem can be expressed by

$$\begin{aligned} & \text{cut-size}(G,k) \\ & \min\left\{\frac{1}{2}Tr(Y^tLY) \middle| Ye_k = e_n \wedge Y^te_n = me_k \wedge y_{ij} \in \{0,1\}\right\} \\ = \; & \min\left\{\frac{1}{2}Tr(LX) \middle| X \in conv(\{Y \mid Ye_k = e_n \wedge Y^te_n = me_k \wedge y_{ij} \in \{0,1\}\})\right\} \end{aligned}$$

where $Y \in \{0,1\}^{n\times k}$ indicates a partition with $y_{v,j} = 1$ means that vertex v belongs to partition i. By using this notation, several semidefinite programming based relaxations arise:

$$\begin{aligned} k\text{-GP}_{\text{R1}}(G,k) &:= \min\left\{\frac{1}{2}Tr(LX) \mid X = X^t \wedge diag(x) = e_n \wedge Xe_n = ke_n \wedge X \succeq 0\right\} \\ k\text{-GP}_{\text{R2}}(G,k) &:= \min\left\{\frac{1}{2}Tr(LX) \mid X = X^t \wedge diag(x) = e_n \wedge Xe_n = ke_n \wedge X \geq 0\right\} \end{aligned}$$

It is shown that the k-GP$_{\text{R2}}$ -bound dominates the DH-bound and even the BRW-bound.

Graph bisection

In the case of the graph bisection problem another model is possible:

$$bw(G) = \min\left\{\frac{1}{4}x^tLx \mid x^te_n = 0 \wedge x \in \{-1,1\}^n\right\}$$

where $x_v = -1(1)$ indicates that vertex v belongs to partition one (two). Then, a relaxation of

$$SDP(G) := \min\left\{\frac{1}{4}Tr(LX) \mid diag(X) = e_n \wedge e_n^tXe_n = 0 \wedge X \succeq 0\right\}$$

can be used as bound. This formulation provides the same bound as the BRW-bound. We will not go into more detail of the notation of the semidefinite program here, interested readers are referred to [PR95, KRC00, KR98]. It is already known that combining semidefinite relaxations with polyhedral information provides very tight relaxations. Karisch, Rendl and Clausen in [KRC00] combined the $SDP(G,k)$ with triangle inequalities, this bound is written as

$$\begin{gathered} pSDP(G) := \\ \min\left\{\tfrac{1}{4}Tr(LX) \mid diag(X) = e_n \wedge e_n^tXe_n = 0 \wedge X \succeq 0 \wedge \mathcal{B}(X) + b \geq 0\right\}. \end{gathered}$$

Summary

In order to summarize, we have presented three different lower bounds for the k-partitioning problem. They are all based on relaxations of the exact cut-size-formula (1.1). Furthermore, we have seen the following ranking:

$$DH(G,k) \leq BRW(G,k) \leq k\text{-GP}_{\text{R2}}(G,k) \leq \text{cut-size}(G,k)$$

Accordingly, for the graph partitioning problem when $k = 2$ we have the relations

$$DH(G,2) \leq BRW(G,2) \leq pSDP(G) \leq bw(G).$$

In Chapter 2 we will compare our new bounds to the bounds presented in this section. In the case of the bisection-problem, we will compare our bounds to the *DH*-bound, *BRW*-bound and the *pSDP*-bound. In the case of $k = 4$ we will take the *DH*-bound and the k-$\text{GP}_{\text{R1}}(G,k)$-bound into consideration and in the case of $k = 2$ and $M > \frac{n}{2}$ we will use the *DH*-bound and the *pSDP*-bound. For all these bound-computations, we have used original code of the authors of the bounds, so that the comparisons will be fair.

1.5 Exact Graph Partitioning Algorithms

As we have already mentioned, the graph partitioning problem and especially the graph bisection problem are well known and well studied problems. Therefore, several approaches which exactly solve the problem have been studied. In the following numeration we list the most important ones:

[JMN93]: This is one of the first published approaches which exactly solves graph partitioning problems. They focus on the general graph partitioning problem with relatively large k's. For the computation of a lower bound, they use a linear-program-formulation of exponential size. So, column-generation is used to solve this linear program where one column corresponds to one possible set of vertices. A branch&bound-framework is used around this lower bound.

[BCR97]: Brunetta, Conforti and Rinaldi presented a branch&cut-algorithm for the graph bisection problem without vertex-weights. They use a polyhedral description of any bisection of a graph and solve a linear relaxation of the description. In order to strengthen this lower bound, they add violated inequalities to the linear program until they do not find any more violated inequalities of the set of inequalities which they use. Then, if the lower bound is not strong enough to prune the subproblem, they perform a branching-step and the procedure continues.

[FMdS$^+$98]: This work introduces a branch&cut-algorithm for the k-partitioning problem with vertex-weights. The approach is quite similar to that of [BCR97]: the polyhedral structure together with violated inequalities are used for the computation of a lower bound. S branch&cut-procedure is applied around this lower bound. Unfortunately, they do not compare their results with the results of [BCR97], so it leaves uncertain which approach gives better results.

[KRC00]: This approach, which was presented by Karisch, Rendl and Clausen, is the most recent and successful one. They use the already described pSDP-bound in order to exactly solve the graph bisection problem. A branch&bound-procedure is used as the frame around the pSDP-bound.

Our approach, which we present in this work, is in close competition with these four published procedures. Hence, at the end of Chapter 5 we will present experimental comparisons with these procedures.

1.6 Graphs for Experiments

Throughout this work we will present the results of several experiments on different graphs. Here we describe the different graphs used. The graphs can be divided into three different sets: Firstly, there are some theoretically defined graphs (e.g. DeBruijn-graphs). Secondly, some graphs which were introduced by other researchers are used and finally we use some random based graphs which we have constructed ourselves.

The first set consists of the following graphs:

DB-*d*: DeBruijn-graphs of dimension d. A DeBruijn-graph consists of 2^d vertices with in-degree 2 and out-degree 2, so it has 2^{d+1} directed edges. Since we use undirected graphs only, we ignore the directions of the edges. The exact bisection width of the DeBruijn-graphs is unknown, asymptotically it is $\Theta(\frac{2^d}{d})$. For example, the DeBruijn-graphs are discussed in [Lei92].

SE-*d* : The Shuffle-Exchange graphs of dimension d. They are also discussed in [Lei92]. They have 2^d vertices with degree 3, so they have $3 \cdot 2^{d-1}$ edges. Like the DeBruijn-graphs, the exact bisection width is unknown, asymptotically it is $\Theta(\frac{2^d}{d})$.

Grid-*a*x*b*: The well known grid-graphs, also known as array. They have ab vertices and $2ab-a-b$ vertices. The bisection width is $\min\{a,b\} + \max\{a,b\} \mod 2$.

Torus-*a*x*b*: The well known tori, i.e. girds with wrap-around-edges. They have ab vertices and $2ab$ edges. The bisection width is twice the bisection width of the Grid-*a*x*b*.

The second set of graphs which have already been introduced and used in different papers consists of the following graphs:

BCR-*a*x*b*g: These graphs are introduced in [BCR97]. They are Grid-*a*x*b* with edge-weights from 1 to 10, drawn from a uniform distribution. At present they are available via anonymous ftp: `ftp.math.unipd.it`

BCR-*a*x*b*t: These graphs are introduced in [BCR97]. They are Torus-*a*x*b* with edge-weights from 1 to 10, drawn from a uniform distribution. At present they are available via anonymous ftp: `ftp.math.unipd.it`

Table 1.1: Characteristics of the BCR-mx.i graphs; maxd (mind) is the maximal (minimal) degree

graph	$\|V\|$	$\|E\|$	maxd	mind	$bw(G)$
BCR-m4.i	32	50	6	0	6
BCR-ma.i	54	72	4	0	2
BCR-me.i	60	96	4	1	3
BCR-m6.i	70	120	9	2	7
BCR-mb.i	74	120	4	2	4
BCR-mc.i	74	125	4	2	6
BCR-md.i	80	129	4	2	4
BCR-mf.i	90	146	4	2	4
BCR-m1.i	100	155	4	1	4
BCR-m8.i	148	265	4	2	7

BCR-*a*x*b*m: These graphs are introduced in [BCR97]. They are Grid-*a*x*b* based graphs with edge-weights. The edges of the grid receive weights from 10 to 100 uniformly generated and all the other edges receive a weight from 1 to 10, also uniformly generated. At present they are available via anonymous ftp: `ftp.math.unipd.it`

BCR-m*x*.i: Real-world instances first used in [BCR97]. The graphs arise from an application of the finite elements method where a factorization of the matrix of the linear system has to be carried out. The graphs are unweighted. This factorization can be modeled as a bisection problem. At present they are available via anonymous ftp: `ftp.math.unipd.it.` Table 1.1 gives some characteristics of these graphs.

ex36[abc]: These random graphs were introduced by Karisch et al., results are shown in [KRC00]. The graphs have 36 vertices and each pair of vertices is adjacent with probability 0.5.

cd*n*: These real-world-graphs were introduced by Johnson et al. in [JMN93], they arise in compiler-design-problems. Originally, they have vertex- and edge-weights. However, we will ignore the vertex-weights as it is also done in [KRC00]. Table 1.2 gives some characteristics of these graphs. We have to mention that our experiments gives a bisection width of 2,177 for the cd61-graph, while in [KRC00] a width of 2,176 is reported.

Finally, the following different classes of random based graphs were generated by ourselves:

Table 1.2: Characteristics of the cd$n$-graphs

graph	$\|V\|$	$\|E\|$	$\max f(e)$	$\text{avg} f(e)$	$bw(G)$
cd30	30	47	368	40.9	302
cd45	45	45	487	63.8	760
cd47a	47	101	1,110	70.8	426
cd47b	47	99	387	39.4	580
cd61	61	186	6,151	178.9	2,177

Random-*n*-*p*-*s*: Random unweighted graphs with n vertices, where each possible edge has a probability p to be in the graph. s is the seed of the random-number-generation. This set of graphs with $n = 36$ and $p = 0.5$ corresponds to the "ex36[abc]" graphs.

RandW-*n*-*m*-*s*: Random graphs with n vertices where every edge has a weight from 0 to 9 uniformly generated. s is the seed of the random-number-generation.

RandPlan-*n*-*p*-*s*: Random planar graphs with n vertices. The graphs are generated using the same principle as the one used in LEDA [MN95]: First a maximal planar graph with n vertices is constructed. This is reached by starting with a maximal planar graph with 3 vertices (i.e. a triangle). Then a vertex is added by randomly selecting one from all the faces and connecting the new vertex with all three vertices of this face. After having constructed a random maximal planer graph, each edge is removed with probability $1 - p$ from the graph. Again, s is the seed of the random-number-generation.

RandRegular-*n*-*d*-*s*: Random regular graphs with n vertices of degree d. They are generated using the algorithm from Steger and Wormald [SW99]. Again, s is the seed of the random-number-generation.

Chapter 2

The New Lower Bounds

2.1 Leightons Bound

One of the most well-known technique for the proving of lower bounds on the bisection width of graphs is based on the embedding of a clique into the given graph. This technique is extensively used by Leighton in his famous book "*Introduction to Parallel Algorithms and Architectures*" [Lei92], hence it is often called the "Leighton-Bound".

The basis of this technique is as follows: We embed the clique graph with n vertices (K_n) into the given graph G, where n is the number of vertices of G. The bisection width of K_n is $\frac{n^2}{4}$ (assuming n is even). It is then clear that in every possible bisection of G at least $\frac{n^2}{4}$ of the embedded clique-edges are cut. If the congestion C of the embedding, i.e. the maximal number of clique-edges that are mapped onto one edge of G, is known it follows that at least $\frac{n^2}{4 \cdot C}$ edges of G are cut. So $\frac{n^2}{4 \cdot C}$ is a lower bound on the bisection width of G. Using these considerations, every correct embedding gives a valid lower bound. Obviously, an embedding with a minimal congestion C gives the best lower bound.

In [Lei92] this technique is used in order to show asymptotic or exact lower bounds on the bisection widths of for example r-dimensional arrays, the butterfly, the hypercube, mesh of trees and the shuffle-exchange graphs. In fact, Leighton often used a small improvement of the above technique by not embedding a clique K_n but by embedding a graph $\bar{K}_n$ with n vertices where every pair of vertices is connected by exactly two edges. In so doing, a simple consideration shows that $\frac{n^2}{2 \cdot C}$ gives a valid lower bound on the bisection width.

Another point of view in relation to the embedding of a clique involves viewing it as a multicommodity flow: Realizing an integral multicommodity flow where every vertex sends a commodity of size one to every other vertex is identical to the embedding of the $\bar{K}_n$ graph. Following this view it becomes clear that for any bisection of a graph with n vertices, commodities of at least size $n \cdot \frac{n}{2}$ have to cross the cut (since every vertex sends a commodity of size one to every vertex of the other partition). If the multicommodity flow is realized with a congestion C, we have a lower bound on the bisection width of the given graph G of $\frac{n^2}{2 \cdot C}$, which is identical to the embedding-view. Therefore the computation of this multicommodity flow with minimal congestion gives the same lower bound. Unfortunately, the computation of the minimal congestion of an integral multicommodity flow is an NP-hard problem. So, the calculation of the Leighton bound for an arbitrary graph is non trivial.

2.2 VarMC- and MVarMC-Bound

2.2.1 Idea

In this section we present the main ideas for the new lower bounds. These new lower bounds can be seen as a generalization of the Leighton bound or an upgrade of its multicommodity flow-view:

In the Leighton bound an embedding of a clique is used in order to get a lower bound on the bisection width. Obviously, the same principle can be used with any host graph H, it is not restricted to cliques. The only requirement is that we have to know an (as good as possible) lower bound $lb(H)$ on the bisection width of the host graph. Following the idea of Leightons bound, we get a lower bound of $\frac{lb(H)}{C}$ on the bisection width of the guest graph G where C is the congestion of the embedding. However, keeping in mind our goal of a general lower bound technique for any graph, we do not want to select an appropriate host graph (with a good lower bound on its bisection width) for each guest graph.

However, when we concentrate on the multicommodity flow-view of the Leighton bound, a generalization is more beneficial. Instead of the requirement that all commodities have to have the same unit size, we can also look at instances of multicommodity flows, where the sizes of the commodities are arbitrary for every pair of source and destination. If we know an amount of commodities which has to cross the cut of every possible bisection (in the following we will call this value *Cut-Flow*), a lower bound for the bisection width of the graph follows.

In the rest of this section we formalize this idea, adapt it to vertex-, edge-weights and arbitrary graph partitioning and we prove valid *Cut-Flow*s.

2.2.2 Definition of Multicommodity Flows

As previously mentioned, the following ideas are based on multicommodity flows. So firstly we have to give a definition of multicommodity flows as we will use them.

Definition 2.1
Let $G = (V, E, g, f)$ be a weighted undirected graph, $d : V \times V \to \mathbb{R}_{\geq 0}$ sizes of commodities. The ***multicommodity flow problem MCF=****(G, d) is the problem of computing flows $h : V \times V \times V \to \mathbb{R}_{\geq 0}$ with $\forall c, v, w \in V : \{v, w\} \notin E \Rightarrow h(c, v, w) = 0$ of commodities on the edges of the graph, such that the demands d are fulfilled, i.e.*

$$\forall c, v \in V, v \neq c : \quad \sum_{w \in V} h(c, w, v) - h(c, v, w) = d(c, v), \tag{2.1}$$

with

$$c(\{v, w\}) := \frac{1}{f(\{v, w\})} \sum_{c \in V} h(c, v, w) + h(c, w, v).$$

and the congestion $C := \max_{e \in E} c(e)$ is minimized.

For the purpose of clarification we want to point out that the graph G is undirected while the flows h have directions. Obviously, an optimal flow h which fulfills the condition $\forall c \in V, \{v, w\} \in E : h(c, v, w) = 0 \vee h(c, w, v) = 0$ always exists. Furthermore, the values $d(v, v)$ are never used inside the definition, so they do not matter.

Moreover, it should be noticed that the flows do not need to be integral, as they were in the interpretation of the clique-embedding. Hence, the multicommodity flow problem can be solved using linear programming and so it becomes clear that these problems can be solved in polynomial time. More details of the computation of multicommodity flows follow in Chapter 4.

2.2.3 Lower Bound based on Cut-Flow and Congestion

In the above section we have intuitively used the term Cut-Flow while presenting the ideas of the new bounds. Here the exact definition follows:

Definition 2.2
Let $GP = (G, k, M)$ be a graph partitioning problem and $MCF = (G, d)$ a multicommodity flow problem. Then a valid ***Cut-Flow*** *CF must fulfill the condition*

$$\forall \pi \in \Pi : \quad CF \leq \sum_{v, w \in V : \pi(v) \neq \pi(w)} d(v, w).$$

When this definition of Cut-Flow, which corresponds to the intuition given above is used, the following theorem expresses the connection between the multicommodity-flow problem and the graph-partitioning problem:

Theorem 2.1
Let $GP = (G,k,M)$ be a graph partitioning problem and $MCF = (G,d)$ a multicommodity flow problem with Cut-Flow CF and congestion C. Then

$$\boxed{\textit{cut-size}(G,k,M) \geq \frac{CF}{C}}$$

holds.

Proof:
Let $\pi \in \Pi$ be the partition with the optimal cut-size, i.e. $cut(G,\pi) = \text{cut-size}(G,k,M)$. Then we have

$$\begin{aligned} C \cdot cut(\pi,G) &= C \cdot \sum_{\{v,w\}\in E;\pi(v)\neq\pi(w)} f(\{v,w\}) \\ &\geq \sum_{v,w\in V:\pi(v)\neq\pi(w)} d(v,w) \\ &\geq CF \\ \Leftrightarrow cut(\pi,G) &\geq \frac{CF}{C} \end{aligned}$$

and the proof is completed. ■

2.2.4 Definition of VarMC and MVarMC

The connection between the congestion of a multicommodity flow problem and the cut-size of a graph partitioning problem is presented above. As a result it becomes clear that in principle any multicommodity flow problem can be used in order to obtain a lower bound on the cut-size. The only condition is that a correct Cut-Flow has to be determined. In order to fulfill this we now present three restricted multicommodity flow instances with different levels of restriction and following that we present a formula for correct Cut-Flows for all variants.

The first and most restricted variant corresponds to the Leighton bound as it is described above. The only improvement which is of interest is that it is generalized for the use of vertex-weights and it does not have to be integral.

Definition 2.3
*Let $G = (V,E,g,f)$ be a graph with weights. The **1-1-MC** is a multicommodity flow problem with commodity-sizes $d : V \times V \to \mathbb{R}_{\geq 0}$ of*

$$\forall v,w \in V : \quad d(v,w) = g(v) \cdot g(w).$$

The next variant has more freedom so we have the possibility of selecting an arbitrary source-strength for any sender:

Definition 2.4
*Let $G = (V, E, g, f)$ be a graph with weights and $s : V \to \mathbb{R}_{\geq 0}$ a source-strength for every sender. The **VarMC** is a multicommodity flow problem with commodity-sizes $d : V \times V \to \mathbb{R}_{\geq 0}$ of*

$$\forall v, w \in V : \quad d(v,w) = g(w) \cdot s(v).$$

And finally, we define a variant with the highest degree of freedom:

Definition 2.5
*Let $G = (V, E, g, f)$ be a graph with weights and $s : V \times V \to \mathbb{R}_{\geq 0}$ a source-strength for every sender-destination-pair. The **MVarMC** is a multicommodity flow problem with commodity-sizes $d : V \times V \to \mathbb{R}_{\geq 0}$ of*

$$\forall v, w \in V : \quad d(v,w) = g(w) \cdot s(v,w).$$

In the following sections we will elaborately compare these three different instances of multicommodity flows and their eligibility for the computation of lower bounds on graph partitioning problems.

2.2.5 Cut-Flow of VarMC and MVarMC

The last remaining point in order to use the defined multicommodity flow instances for lower bound-computations of graph partitioning problems is the determination of good and valid Cut-Flows. The following theorem gives a formula for the MVarMC variant.

Theorem 2.2
Let $GP = (G, k, M)$ be a graph partitioning problem and $s : V \times V \to \mathbb{R}_{\geq 0}$ the source-strengths of an MVarMC instance. Then, with $\bar{s_v} := \max_{w \in V} s(v,w)$, a Cut-Flow CF of

$$CF = \sum_{v \in V} \left(\sum_{w \in V} s(v,w) \cdot g(w) - M \cdot \bar{s_v} \right)$$

holds.

Proof:
We are looking for a guaranteed Cut-Flow of the commodities with sizes $s(v,w)$. A Cut-Flow CF is guaranteed if for any possible partition according to the parameters M

and k the actual flow between the partitions is at least as large as CF. So let us assume any feasible partition $V = V_1 \cup \ldots \cup V_k$. Then

$$\begin{aligned} CF &= \sum_{i=1}^{k} \sum_{v \in V_i} \sum_{w \in V \setminus V_i} s(v,w) \cdot g(w) \\ &\geq \sum_{i=1}^{k} \sum_{v \in V_i} \left(\sum_{w \in V} s(v,w) \cdot g(w) - \bar{s_v} \sum_{w \in V_i} g(w) \right) \\ &\geq \sum_{v \in V} \left(\sum_{w \in V} s(v,w) \cdot g(w) - \bar{s_v} \cdot M \right) \end{aligned}$$

holds and proves the theorem. ■

Valid Cut-Flows for the VarMC and 1-1-MC follows from the Cut-Flow for the MVarMC version:

Corollary 2.1
The VarMC instance guarantees a Cut-Flow CF of

$$CF = (N - M) \cdot \sum_{v \in V} s(v).$$

The 1-1-MC instance guarantees a Cut-Flow CF of

$$CF = N \cdot (N - M)$$

Proof:
Let us start with the VarMC variant and let $s : V \to \mathbb{R}_{\geq 0}$ be its source-strengths. In comparison to the MVarMC, we know that $\forall v, w \in V : \ s(v,w) = s(v)$. So, $\bar{s_v} = s(v)$ follows directly and putting this into Theorem 2.2 we get

$$\begin{aligned} CF &= \sum_{v \in V} \left(\sum_{w \in V} s(v,w) \cdot g(w) - M \cdot \bar{s_v}) \right) \\ &= \sum_{v \in V} \left(\sum_{w \in V} s(v) \cdot g(w) - M \cdot s(v) \right) \\ &= \sum_{v \in V} s(v) \left(\sum_{w \in V} g(w) - M \right) \\ &= \sum_{v \in V} s(v) \, (N - M) \end{aligned}$$

and the Cut-Flow of the VarMC is proved.

Table 2.1: Cut-Flows for the different graph partitioning problems and the different multicommodity instances

	1-1-MC	VarMC	MVarMC
GP	$(N-M)N$	$(N-M)\sum_v s(v)$	$\sum_v \left(\sum_w s(v,w)\cdot g(w) - M\cdot \bar{s_v}\right)$
kGP	$(1-\frac{1}{k})N^2$	$(1-\frac{1}{k})N\sum_v s(v)$	$\sum_v \left(\sum_w s(v,w)\cdot g(w) - \frac{N}{k}\cdot \bar{s_v}\right)$
bisGP	$\frac{1}{2}N^2$	$\frac{1}{2}N\sum_v s(v)$	$\sum_v \left(\sum_w s(v,w)\cdot g(w) - \frac{N}{2}\cdot \bar{s_v}\right)$

The 1-1-MC is a specialization of the VarMC with $s(v) = g(v)$. So, putting this into the formula we get

$$\begin{aligned} CF &= (N-M)\sum_{v\in V} s(v) \\ &= (N-M)\cdot N \end{aligned}$$

and the Cut-Flow of the 1-1-MC is proved. ■

The values shown for the Cut-Flows correspond to the general graph partitioning problem. Of course, they can easily be adapted to the k-partitioning problem and the bisection problem. Table 2.1 shows the formulas for all the combinations.

Extension for $M > \frac{n}{k}$

Finally, we present an extension of Theorem 2.2 for the case of the graph partitioning problem with $M > \frac{n}{k}$. The idea of the Extension is based on the observation, that in the equation $\sum_v (\sum_w s(v,w)\cdot g(w) - M\cdot \bar{s_v})$ the worst case that every vertex v is in a partition of maximal size is assumed. Of course, if $M > \frac{n}{k}$ is used then not every single vertex can be in a partition with maximal size M. This leads to the following extension:

Theorem 2.3
Let $GP = (G,k,M)$ be a graph partitioning problem and s the source-strengths of an MVarMC instance. Then, with $\bar{s_v} := \max_{w\in V} s(v,w)$, $\tilde{s} := \min_{v\in V} \frac{\bar{s}(v)}{g(v)}$ and $R := N - M\lfloor\frac{N}{M}\rfloor$, a Cut-Flow CF of

$$CF = \sum_{v\in V}\left(\sum_{w\in V} s(v,w)\cdot g(w) - M\cdot \bar{s_v}\right) + \tilde{s}R(M-R)$$

holds.

Proof:
The theorem can be proved using the same ideas as in the proof of Theorem 2.2 but with a more careful calculation:

$$\begin{aligned}
CF &= \sum_{i=1}^{k}\sum_{v\in V_i}\sum_{w\in V\setminus V_i} s(v,w)\cdot g(w)\\
&\geq \sum_{i=1}^{k}\sum_{v\in V_i}\left(\sum_{w\in V} s(v,w)\cdot g(w) - \bar{s_v}\sum_{w\in V_i} g(w)\right)\\
&= \sum_{v\in V}\left(\sum_{w\in V} s(v,w)\cdot g(w) - \bar{s_v}\cdot M\right) + \sum_{i=1}^{k}(M-N_i)\sum_{v\in V_i}\bar{s}(v)\\
&\geq \sum_{v\in V}\left(\sum_{w\in V} s(v,w)\cdot g(w) - \bar{s_v}\cdot M\right) + \tilde{s}\sum_{i=1}^{k}(M-N_i)
\end{aligned}$$

It remains to be shown that $\sum_{i=1}^{k}(M-N_i)\geq R(M-R)$ holds for every feasible partition. A feasible partition fulfills $\forall i: N_i \leq N \wedge \sum_{i=1}^{k} N_i = N$, so

$$\begin{aligned}
\sum_{i=1}^{k}(M-N_i) &= MN - \sum_{i=1}^{k} N_i^2\\
&\geq MN - \lfloor\frac{N}{M}\rfloor M^2 - R^2\\
&= R(M-R)
\end{aligned}$$

follows. ■

As in the case of Theorem 2.2, Cut-Flows for the VarMC and 1-1-MC follow:

Corollary 2.2
The VarMC instance guarantees a Cut-Flow CF of

$$CF = (N-M)\cdot\sum_{v\in V} s(v) + \tilde{s}R(M-R)$$

with $\tilde{s} := \min_{v\in V}\frac{s(v)}{g(v)}$. *The 1-1-MC instance guarantees a Cut-Flow CF of*

$$CF = N\cdot(N-M) + R(M-R).$$

2.2.6 Summary and Outlook

Summarizing, in this section we have presented how multicommodity flows can in general be used for the computation of lower bounds on the graph partitioning problem.

The main formula for this lower bound is based on the Cut-Flow CF and the congestion C and we have

$$\frac{CF}{C} \leq \text{cut-size}.$$

Three multicommodity-flow instances, 1-1-MC, VarMC and MVarMC, each with a different degree of freedom were introduced. Theorems 2.2 and 2.3 and Corollaries 2.2 and 2.1 provide formulas for valid Cut-Flows for the three different instances. So, in order to compute good lower bounds for a given graph, we have to determine good source-strengths $s : V \longrightarrow \mathbb{R}_{\geq 0}$ or $s : V \times V \longrightarrow \mathbb{R}_{\geq 0}$ for the VarMC and MVarMC instances respectively and we have to compute flows $h : V \times V \times V \longrightarrow \mathbb{R}_{\geq 0}$ such that the congestion is minimized. A lower bound with computed strengths s and flows h will be called an *MVarMC(1-1-MC, VarMC)-solution*. Of course we want to maximize the lower bounds, so we are looking for the best strengths and flows. The resulting bounds which are achievable with the best strengths and flows will be called *MVarMC(1-1-MC, VarMC)-bound*. If we want to say something about all three variations (1-1-MC, VarMC and MVarMC) we will say *MC-solution* or *MC-bound*. In the next section we reveal some experimental results for these MVarMC-, VarMC-, and 1-1-MC-bounds. In Chapter 3 we will present some theoretical observations on these bounds. In Chapter 4 methods for the efficient computation of the bounds for arbitrary graphs are presented and in Chapter 5 an approach to the exact calculation of the cut-size based on the MVarMC-, VarMC- and 1-1-MC-bounds is presented.

2.3 Experimental Evaluation of the Lower Bounds

2.3.1 The Experiments

We have carried out a large amount of experiments in order to discover the strengths and weaknesses of the three proposed lower bounds. On the one hand, we show how these three bounds compare to each other on the different graphs and on the other hand we compare them to the three known lower bounds DH-bound, BRW-bound and pSDP-bound, as they are presented in Section 1.4.

Until now, we have not stated how the MC-bounds are calculated. In order to compute these lower bounds, we have to select the best commodity-sizes for the VarMC and MVarMC instances and we have to calculate the minimal congestion. Fortunately, the selection of the best commodity-sizes can be inserted into the linear program which is used for solving Multicommodity-Flow-problems with given commodity-sizes. More details of these linear programs and their computation will be provided in Chapter 4.

The results which we present here are computed by solving these linear programs with the barrier algorithm of CPLEX [ILO00], version 7.0. The computation of the

Table 2.2: Summary of the lower bounds (upper value) and their computation times (lower value) of bisection problems with different graphs

graph	1-1-MC	VarMC	MVarMC	pSDP	BRW	DH	Opt.
DB-7	27.5 30	**29.0** 53	**29.0** 1:00	22.0 1:24	15.0	9.7	30
Grid-11x10	10.1 9	**11.0** 13	**11.0** 16	8.8 51	4.0	2.2	11
BCR-mf.i	3.0 4	3.2 10	**3.4** 11	2.9 26	1.1	0.3	4
BCR-7x10g	13.6 2	18.3 6	**20.1** 5	19.3 9	10.9	4.5	23
RandRegular-100-3-1	15.2 16	**15.4** 55	**15.4** 35	12.5 31	8.5	6.7	16
RandPlan-100-1-0	24.7 11	35.1 18	**35.1** 25	34.7 34	21.1	12.1	38
Random-100-0.05-3	12.8 12	25.0 35	**34.0** 1:31	32.5 30	22.6	7.2	34
Random-32-0.5-0	76.5 3	87.2 5	87.2 5	**95.3** $<$1	90.6	67.1	96

DH-bound involves the calculation of the smallest eigenvalues of the Laplace matrix. This is done using the standard software-package LAPACK [Dem89] and BLAS [CHL$^+$94]. The computation of the BRW-bound is carried out using a package provided by the authors of [FRW94]. The computations of the DH-bound and BRW-bound last clearly less than one second on all the graphs tested. For this reason we have not given the times of these computations. The pSDP-bound is computed using a package provided by the authors of [KRC00], see also [Kar98]. All calculations are done on the same system with a Pentium-III, 937 MHz, processor with 1GB main memory. All packages are compiled with the gcc-compiler, version 2.95, with the same level of optimization (-O 3).

The best bound in every row is printed bold faced. The column “Opt.” gives the optimal cut-size or, if the value is parenthesized, the best known value. Some missing entries of the 1-1-MC-, VarMC- and DH-bound (e.g. the BCR-m4.i graph) indicate that the graph is not connected.

2.3.2 Bisection Problems

The first set of experiments is applied to the graph bisection problem. The details of all results are presented in the Appendix in Tables A.1 - A.6. In Table 2.2 we give a summary which shows the typical behavior of the different bounds.

When we compare the three different MC-bounds, the following conclusions ca be drawn:

- The 1-1-MC bound gives worse bounds on the majority of the graphs, except those graphs where this bound is already exact.
- The computation time of the 1-1-MC is shorter than the times of the VarMC and MVarMC bounds. Surprisingly, the computation times of these two bounds are comparable.
- On the "more structured" graphs (e.g. DeBruijn graphs) the VarMC and MVarMC bounds are nearly identical. However, the MVarMC bound is better on some "unstructured" and "sparse" graphs (e.g. BCR-mf.i).

So we can conclude that the 1-1-MC bound is a little bit faster but clearly worse than the other bounds. The VarMC bound is nearly as good as the MVarMC bound, but it is not faster. So the use of the MVarMC bound is therefore preferable since it is at least as good as the VarMC bound is, sometimes clearly better, and comparably fast.

When we take the DH-bound, BRW-bound and pSDP-bound into consideration, we can see that the gap between these three bounds is surprisingly large. Finally, when the two possibly best bounds MVarMC and pSDP are compared, it becomes clear that the MVarMC-bound is better on the more "structured" graph, e.g. DeBruijn, grids, random planar, random regular graphs. On the other hand, the pSDP-bound is superior on random dense graphs, e.g. Random-32-0.5 graphs.

2.3.3 k-partitioning Problems

The second set of experiments is done with the k-partitioning problem using $k = 4$. In order to avoid problems with the different bounds, we have used instances with $k|n$, only. Unfortunately, the package we use for the computation of the BRW-bound can only handle the graph bisection problem. So we cannot present results of this bound. Furthermore, the computation of the k-GP_{R2}-bound inside the is CUTSDP-package is faulty, so we use the k-GP_{R1}-bound.

Table 2.3 shows an extract of the resulting bounds. Tables A.7-A.12 show the detailed results. Furthermore, Figure 2.1 show the effect of an increasing number of partitions onto the quality of the bounds. The value of the three MC-bounds are shown and an upper bound on the cut-size is given which is computed using the PARTY-library [PD96]. The figure shows the results on one random planar graph with 60 vertices (i.e. RandPlan-60-1-0).

The following conclusions can then be drawn:

Table 2.3: Summary of the lower bounds and their computation times of 4-partitioning problems with different graphs

graph	1-1-MC	VarMC	MVarMC	GP_{R1}	DH	Opt.
DB-6	23.8	25.4	**27.2**	15.4	13.6	32
	3	9	10	<1		
Grid-12x9	13.5	13.5	**21.1**	5.8	5.1	(23)
	7	8	33	1		
BCR-m8.i	10.5	10.5	**20.7**	4.1	3.5	22
	22	19	2:53	6		
BCR-6x10g	28.6	37.5	**51.2**	20.4	16.7	(59)
	1	3	4	<1		
RandRegular-60-3-0	10.6	11.7	**15.1**	7.1	6.6	(18)
	<1	4	6	<1		
RandPlan-60-1-0	26.8	38.1	**45.5**	31.4	18.8	(47)
	2	11	11	<1		
Random-60-0.05-0	-	-	**16.6**	10.9	-	(18)
			8	<1		
Random-32-0.5-0	114.8	130.8	135.5	**137.0**	108.1	(153)
	3	6	7	<1		

Figure 2.1: Example of the effect of varying k on the MC-bounds

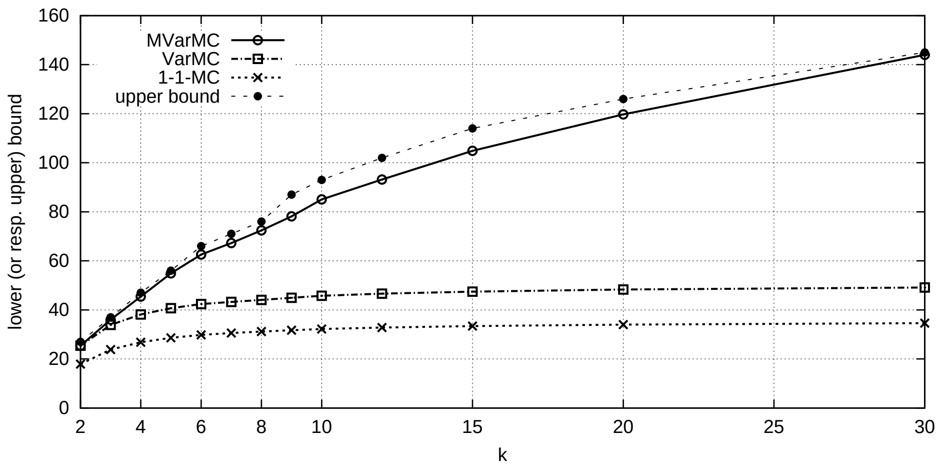


Table 2.4: Summary of the lower bounds and their computation times of graph partitioning problems with $k = 2$, $M = \lfloor \frac{2}{3}n \rfloor$ and different graphs

graph	1-1-MC	VarMC	MVarMC	pSDP	DH	Opt.
DB-7	**24.5**	**24.5**	**24.5**	18.9	6.5	26
	30	32	28	1:19		
Grid-11x10	**9.0**	**9.0**	**9.0**	7.1	1.5	10
	9	8	10	43		
BCR-mf.i	**2.7**	**2.7**	**2.7**	1.6	0.2	3
	4	11	7	21		
BCR-7x10g	12.3	12.5	**12.8**	12.6	3.1	14
	2	8	7	7		
RandRegular-100-3-1	**13.7**	**13.7**	**13.7**	10.7	4.6	14
	16	38	21	28		
RandPlan-100-1-0	22.2	24.2	24.4	**24.5**	8.3	27
	13	21	34	32		
Random-100-0.05-3	11.4	17.0	**26.8**	26.6	4.9	32
	13	52	1:31	29		
Random-32-0.5-0	69.0	72.3	73.5	**81.8**	46.1	82
	3	6	6	<1		

- The MVarMC-bound is generally the best one on all types of graphs. The only exceptions are the random dense graphs, where the $k - \mathrm{GP}_{\mathrm{R1}}$-bound is better or at least as good as the MVarMC-bound.

- However, the computation times for the MVarMC-bound are now generally longer than the times for the other MC-bounds.

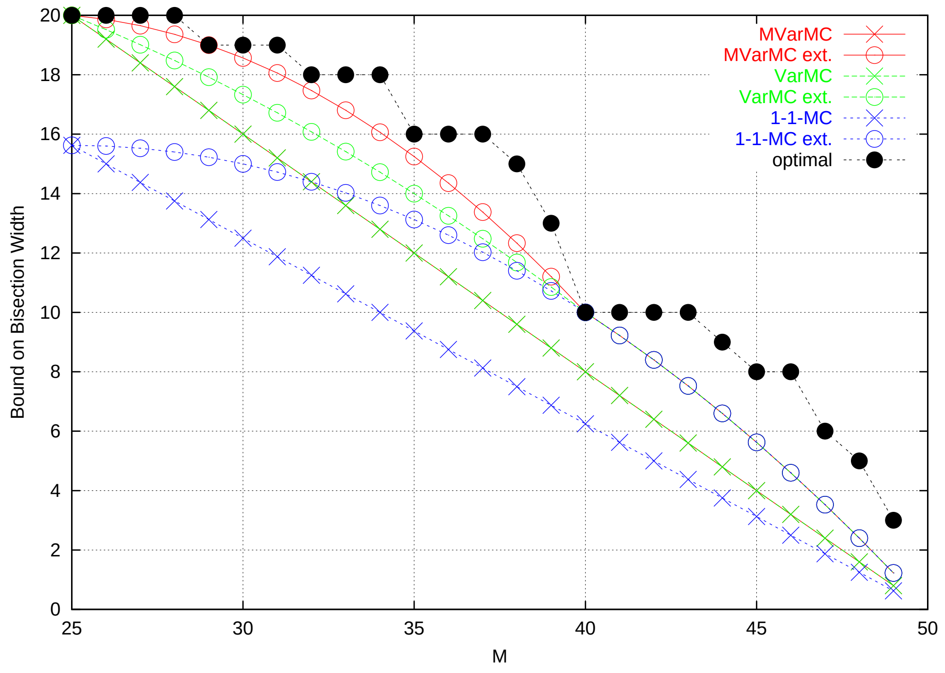
2.3.4 Graph Partitioning Problems with $k = 2$ and $M = \frac{2}{3}n$

Finally, the last set of experiments concentrates on the general graph partitioning problem. In order to concentrate on the effects of unbalanced partitions we have done experiments with $k = 2$ and $M = \frac{2}{3}n$. Again, the BRW-bound could not be computed. The results of the multicommodity-based bounds are attained using the Cut-Flow Theorem 2.3 or Lemma 2.2 respectively. Table 2.4 shows an extract of the resulting bounds. Tables A.13-A.18 give the detailed results.

The following conclusions can then be drawn:

- In these graph partitioning instances, the MC-bounds are much more similar when compared to the bisection problems. The MVarMC-bound is only better for the random graphs .

Figure 2.2: Example of the effect of varying M on the MC-bounds



- The pSDP-bound is superior to the MVarMC bounds on dense random graphs. These two bounds are approximately equal on the sparse random graphs, e.g. Random-100-0.05 or RandPlan-100-1.

Furthermore, Figure 2.2 displays the results of the different MC-bounds with one graph, $k = 2$ and varying M. For this example a random planar graph with 50 vertices (and seed 0) is used. The bounds with "ext." refer to the extension on the Cut-Flow formula as it is given in Theorem 2.3. The data of the MVarMC-bound are not visible since they are always identical to the VarMC in this example. The data of the extended MVarMC and extended VarMC are not visible when $M \geq 40$, since they are identical to the extended 1-1-MC-bound in these cases.

The figure shows on the one hand that the superiority of the MVarMC-bound is the smaller the bigger M is. On the other hand, the practical superiority of the extension on the Cut-Flow formula is shown. Furthermore, the figure illustrates that the MVarMC-bound can utilize the Cut-Flow extension more than the VarMC-bound.

2.3.5 Summary

To sum up the results of all the experiments, we feel the following two main conclusions can be drawn:

Observation 2.1
From the experiments it can be concluded:

- *The MVarMC-bound is preferable when compared to the other MC-bounds. Its bounds are always as good as the other bounds and sometimes much better. The running times are comparable to the times of the VarMC and 1-1-MC if the bounds are identical. The instances where the running times of the MVarMC-bound are longer than the other times, correspond to the instances where the bound is also better.*
- *When the MVarMC-bound is compared to the pSDP-bound, it becomes clear that both have the right to exist. The MVarMC-bound is better on more "structured" or sparse graphs while the pSDP-bound is better on random dense graphs.*

Chapter 3

Theoretical Issues

In this chapter we will present some analyses of the 1-1-MC, VarMC and MVarMC bounds. In Section 3.1 we will show how symmetries of the graphs can be used for the construction maximal bounds. Based on this, in Section 3.2 the MC-bounds for several graphs will be developed. Finally, in Section 3.3 some upper bounds on the MC-bounds which are based on edge-expansion-properties of the graphs will be presented.

Since most defined graphs have no vertex- and edge-weights, we do not use weighted graphs in this chapter. However there is no doubt that all definitions and conclusions can be adapted to weighted graphs.

3.1 Symmetrical Solutions

3.1.1 Definitions

Before we can present the results, first of all we have to give some basic definitions of symmetries in graphs. These symmetries are well known in algebraic graph theory, see e.g. [Whi84].

Definition 3.1
Let $G = (V,E)$ be a graph. An automorphism $\varphi : V \to V$ *is a bijective function of the vertices with*

$$\forall v,w \in V : \{v,w\} \in E \Leftrightarrow \{\varphi(v),\varphi(w)\} \in E$$

$Aut(G)$ denotes the set of all automorphisms of the graph G.

For ease of notation, we will use $\varphi(v_1),\dots,\varphi(v_i) = \varphi(v_1,\dots,v_i)$. It is widely known that $Aut(G)$ is a group under the function composition. The relation $\sim_G$ on V with

$v_1 \sim_G v_2$ if and only if $\exists \varphi(G) : \varphi(v_1) = v_2$, is an equivalence relation . This relation partitions V into equivalent classes, we refer to a class as a vertex-orbit. Similarly, a corresponding equivalence relation partitions the edges E into equivalent classes. Here a class is referred to as an edge-orbit.

A graph is called vertex-symmetric or edge-symmetric, if there are appropriate automorphisms:

Definition 3.2
Let $G = (V,E)$ be a graph. G is vertex-symmetric *if and only if*

$$\forall v, w \in V : \exists \varphi \in Aut(G) : \varphi(v) = w.$$

G is edge-symmetric *if and only if*

$$\forall e, e' \in E : \exists \varphi \in Aut(G) : \varphi(e) = e'.$$

Note that these definitions can also be given in terms of vertex- and edge-orbits: G is vertex-symmetric if and only if G has exactly one vertex-orbit. G is edge-symmetric if and only if G has exactly one edge-orbit.

Now that we have introduced the basic terms of symmetrical graphs, we will define the terms of symmetrical multicommodity-based lower bounds. The idea behind the following definitions is that symmetrical solutions comply with the symmetries in the graph:

Definition 3.3
Let $MVarMC = (G,s,h)$ be an MVarMC-solution on graph $G = (V,E)$ with source-strengths $s : V \times V \to \mathbb{R}_{\geq 0}$ and flows $h : V \times V \times V \to \mathbb{R}_{\geq 0}$. We call this solution a symmetric MVarMC-solution *if and only if* $\forall \{v,w\}, \{v',w'\} \in V \times V$:

$$\begin{aligned} &\exists \varphi \in Aut(G) : \varphi(v,w) = \{v',w'\} \\ &\Rightarrow \quad s(v,w) = s(v',w') \end{aligned} \tag{3.1}$$

and $\forall \{c,v,w\}, \{c',v',w'\} \in V \times V \times V$:

$$\begin{aligned} &\exists \varphi \in Aut(G) : \varphi(c,v,w) = \{c',v',w'\} \\ &\Rightarrow \quad h(c,v,w) = h(c',v',w') \end{aligned} \tag{3.2}$$

Accordingly, let $VarMC = (G,s,h)$ be an VarMC-solution on graph $G = (V,E)$ with source-strengths $s : V \to \mathbb{R}_{\geq 0}$ and flows $h : V \times V \times V \to \mathbb{R}_{\geq 0}$. We call this solution a symmetric VarMC-solution *if and only if*

$$\begin{aligned} &\forall v, v' \in V : \exists \varphi \in Aut(G) : \varphi(v) = v' \\ &\Rightarrow \quad s(v) = s(v') \end{aligned} \tag{3.3}$$

and Equation (3.2) holds.

Accordingly, let 1-1-MC= (G,h) be an 1-1-MC-solution on graph $G=(V,E)$ with flows $h : V \times V \times V \to \mathbb{R}_{\geq 0}$. We call this solution a symmetric 1-1-MC-solution *if and only if Equation (3.2) holds.*

Obviously, the definitions lead to equal congestion on symmetrical edges, which suggests that the definitions are sensible:

Corollary 3.1
Let $h : V \times V \times V \to \mathbb{R}_{\geq 0}$ be the flow of a symmetrical MC-solution. Then

$$\forall e, e' \in E : \quad e \sim_G e' \Rightarrow c(e) = c(e')$$

holds.

Proof:
We look at $e = \{v,w\}$ and $e' = \{v',w'\}$ with $e \sim_G e'$, let $\varphi \in Aut(G)$ be the automorphism with $\varphi(e) = e'$. Then from Equation (3.2) if follows: $\forall u : h(u,v,w) = h(\varphi(u,v,w)) = h(\varphi(u),v',w')$. So we have

$$\begin{aligned} c(e) = c(\{v,w\}) &= \sum_{u \in V} h(u,v,w) + h(u,w,v) \\ &= \sum_{u \in V} h(\varphi(u),v',w') + h(\varphi(u),w',v') \\ &= \sum_{u \in V} h(u,v',w') + h(u,w',v') \\ &= c(\{v',w'\}) = c(e') \end{aligned}$$

which holds since φ is bijective, i.e. $\{\varphi(v) | v \in V\} = V$. ■

3.1.2 Existence of Optimal and Symmetrical MC-solutions

The main point which we would like to state here is that there is always a symmetrical MC-solution which gives the maximal lower bound. So, we can restrict ourselves to symmetrical MC-solutions if the symmetries of the graphs are known.

Theorem 3.1
Let $GP = (G,k,M)$ be a graph partitioning problem. A symmetrical MVarMC-solution $a = (GP,s,h)$ with source-strengths s and flows h with maximal bound exists, i.e.

$$\textit{MVarMC-bound} = \frac{CF_a}{C_a}$$

where CF_a is the Cut-Flow of the MVarMC-solution and C_a is its congestion.

Proof:
Let the source-strengths or the flows of an MVarMC-solution $\tilde{a} = (GP, \tilde{s}, \tilde{h})$ with maximal lower bound be $\tilde{s}$, $\tilde{h}$ respectively. Then we assign

$$s(v,w) = \frac{1}{|Aut(G)|} \sum_{\varphi \in Aut(G)} \tilde{s}(\varphi(v,w))$$

and

$$h(u,v,w) = \frac{1}{|Aut(G)|} \sum_{\varphi \in Aut(G)} \tilde{h}(\varphi(u,v,w)).$$

If we can show that the MVarMC-solution $a = (GP, s, h)$ is feasible, symmetrical and optimal, then the theorem is proved.

- Firstly, we concentrate on the feasibility. In order to be feasible, the constraints (2.1) of Definition 2.1 must be fulfilled. We have $d(v,w) = s(v,w)$ and so $\forall u, v \in V, v \neq u$:

  $$\begin{aligned}
  & \sum_{w \in V} h(u,w,v) - h(u,v,w) \\
  = & \sum_{w \in V} \frac{1}{|Aut(G)|} \sum_{\varphi \in Aut(G)} \tilde{h}(\varphi(u,v,w)) - \frac{1}{|Aut(G)|} \sum_{\varphi \in Aut(G)} \tilde{h}(\varphi(u,w,v)) \\
  = & \frac{1}{|Aut(G)|} \sum_{\varphi \in Aut(G)} \sum_{w \in V} \tilde{h}(\varphi(u,w,v)) - \tilde{h}(\varphi(u,v,w)) \\
  = & \frac{1}{|Aut(G)|} \sum_{\varphi \in Aut(G)} \tilde{s}(\varphi(u,v)) \\
  = & s(u,v)
  \end{aligned}$$

  holds which proves that the pair (s,h) is a feasible multicommodity-flow.

- In order to be symmetrical the constraints (3.1) and (3.2) of Definition 3.3 must be fulfilled. Firstly, we look at any pairs $\{v,w\}$, $\{v',w'\}$ with $\bar{\varphi}(v,w) = \{v',w'\}$. Then we have

  $$\begin{aligned}
  s(\bar{\varphi}(v,w)) &= \frac{1}{|Aut(G)|} \sum_{\varphi \in Aut(G)} \tilde{s}(\varphi(\bar{\varphi}(v,w))) \\
  &= \frac{1}{|Aut(G)|} \sum_{\varphi \in Aut(G)} \tilde{s}(\varphi(v,w)) \\
  &= s(v,w)
  \end{aligned}$$

  which holds since $Aut(G)$ is a group under composition, so the constraints (3.1) are fulfilled. Now we look at any triples $\{u,v,w\}$, $\{u',v',w'\}$ with $\bar{\varphi}(u,v,w) =$

$\{u', v', w'\}$. Then we have

$$\begin{aligned} h(\bar{\varphi}(u,v,w)) &= \frac{1}{|Aut(G)|} \sum_{\varphi \in Aut(G)} \tilde{h}(\varphi(\bar{\varphi}(u,v,w))) \\ &= \frac{1}{|Aut(G)|} \sum_{\varphi \in Aut(G)} \tilde{h}(\varphi(u,v,w)) \\ &= h(u,v,w) \end{aligned}$$

which holds since $Aut(G)$ is a group under composition, so the constraints (3.2) are fulfilled.

- Finally, we have to show that the constructed MVarMC-solution gives the maximal lower bound. So we must look at the Cut-Flow CF_a and the congestion C_a of the constructed MVarMC-solution. In the following steps we will show that $CF_a \geq CF_{\tilde{a}}$ and $C_a \leq C_{\tilde{a}}$, so that the bound of the constructed symmetric solution is at least as good as the bound of the given solution $\tilde{a}$:

  – We begin with the congestion. Let e_c be the edge of solution a with the maximal congestion, so $C_a = c(e_c)$. Let E_c be the edge-orbit with $e_c \in E_c$. Then we will show, that $c(e_c) = \frac{1}{|E_c|} \sum_{e \in E_c} \tilde{c}(e)$:

$$\begin{aligned} c(e_c) &= \sum_{v \in V} h(v, e_c) \\ &= \frac{1}{|Aut(G)|} \sum_{\varphi \in Aut(G)} \sum_{v \in V} \tilde{h}(\varphi(v, e_c)) \\ &= \frac{1}{|Aut(G)|} \sum_{\varphi \in Aut(G)} \sum_{v \in V} \tilde{h}(v, \varphi(e_c)) \\ &=^{(a)} \frac{1}{|A_1| + \ldots + |A_k|} \sum_{i=1}^{k} \sum_{\varphi \in A_i} \sum_{v \in V} \tilde{h}(v, \varphi(e_c)) \\ &=^{(b)} \frac{1}{k|A_1|} \sum_{i=1}^{k} |A_1| \sum_{v \in V} \tilde{h}(v, e_i) \\ &= \frac{1}{|E_c|} \sum_{e \in E_c} \tilde{c}(v, e) \end{aligned}$$

  In equation (a) we have used a disjoint partition of $Aut(G) = \cup_{i=1}^{k} A_i$ with $A_i = \{\varphi \in Aut(G) | \varphi(e_c) = e_i\}$ and $E_c = \{e_1, \ldots, e_k\}$. In equation (b) we have used the fact that $\forall i: \ |A_i| = |A_1|$; this can be shown for example with the help of a bijective function $\sigma_i : A_i \rightarrow A_1$ with $\sigma_i(\varphi) = \varphi_{i1} \circ \varphi$ where φ_{i1} is any automorphism with $\varphi_{i1}(e_i) = e_1$. It is then straightforward to show that σ_i is bijective.

Therefore, $c(e_c) = \frac{1}{|E_c|}\sum_{e\in E_c}\tilde{c}(e)$ holds and it follows: $\exists e \in E_c : \tilde{c}(e) \geq c(e_c)$, so $C_a \leq C_{\tilde{a}}$ becomes clear, i.e. the congestion of the constructed symmetrical solution is not bigger than the congestion of solution a.

– Secondly, we concentrate on the Cut-Flow CF_a with

$$CF_a = \sum_v \left(\sum_w s(v,w) - M \max_w s(v,w)\right).$$

In order to show that $CF_a \geq CF_{\tilde{a}}$ we show $\sum_v\sum_w s(v,w) \geq \sum_v\sum_w \tilde{s}(v,w)$ and $\sum_v M \max_w s(v,w) \leq \sum_v M \max_w \tilde{s}(v,w)$:

$$\begin{aligned}\sum_{v\in V}\sum_{w\in V} s(v,w) &= \sum_{v\in V}\sum_{w\in V}\frac{1}{|Aut(G)|}\sum_{\varphi\in Aut(G)}\tilde{s}(\varphi(v,w))\\ &= \sum_{v\in V}\sum_{w\in V}\tilde{s}(v,w)\end{aligned}$$

and

$$\begin{aligned}\sum_{v\in V}\max_{w\in V} s(v,w) &= \sum_{v\in V}\max_{w\in V}\frac{1}{|Aut(G)|}\sum_{\varphi\in Aut(G)}\tilde{s}(\varphi(v,w))\\ &\leq \sum_{v\in V}\max_{w\in V}\frac{1}{|Aut(G)|}\sum_{\varphi\in Aut(G)}\max_{w'\in V}\tilde{s}(\varphi(v),w')\\ &= \frac{1}{|Aut(G)|}\sum_{\varphi\in Aut(G)}\sum_{v\in V}\max_{w\in V}\tilde{s}(\varphi(v),w)\\ &= \sum_{v\in V}\max_{w\in V}\tilde{s}(v,w)\end{aligned}$$

So, $CF_a \geq CF_{\tilde{a}}$ is then proved.

So, altogether we have shown that the constructed solution is feasible, symmetric and has no bound which is worse than any other MVarMC-solution. Therefore, the theorem is proved. ■

According to Theorem 3.1 we can also show that there is always a symmetrical VarMC-solution with a maximal bound:

Corollary 3.2
Let $GP = (G,k,M)$ be a graph partitioning problem. There is a symmetrical VarMC-solution $a = (GP,s,h)$ with source-strengths s and flows h with a maximal bound, i.e.

$$\textit{VarMC-bound} = \frac{CF_a}{C_a}$$

where CF_a is the Cut-Flow of the VarMC-solution and C_a is its congestion.

Proof:
Let the source-strengths or the flows of the VarMC-solution $\tilde{a} = (GP, \tilde{s}, \tilde{h})$ with maximal lower bound be $\tilde{s}$, $\tilde{h}$, respectively. Then we assign

$$s(v,w) = \frac{1}{|Aut(G)|} \sum_{\varphi \in Aut(G)} \tilde{s}(\varphi(v)) \tag{3.4}$$

and

$$h(u,v,w) = \frac{1}{|Aut(G)|} \sum_{\varphi \in Aut(G)} \tilde{h}(\varphi(u,v,w)).$$

If we can show that the VarMC-solution $a = (GP, s, h)$ is feasible, symmetrical and optimal, the Corollary is proved. From the proof of Theorem 3.1 we know that solution a is a feasible, symmetrical and optimal MVarMC-solution. In fact it remains to be shown that the solution is a VarMC-instance, and not an MVarMC-instance. From the above assignment for $s(v,w)$ it follows directly that $\forall v, w, w' \in V : s(v,w) = s(v,w')$, which is the condition for a VarMC-instance. The Corollary is then proved. ∎

And finally, the same applies to 1-1-MC-instances:

Corollary 3.3
Let $GP = (G, k, M)$ be a graph partitioning problem. A symmetrical 1-1-MC-solution $a = (GP, h)$ with flows h and maximal bound exists, i.e.

$$\textit{1-1-MC-bound} = \frac{CF_a}{C_a}$$

where CF_a is the Cut-Flow of the VarMC-solution and C_a is its congestion.

Proof:
Let $\tilde{h}$ be the flows of an VarMC-solution $\tilde{a} = (GP, \tilde{h})$ with maximal lower bound. Then we assign

$$h(u,v,w) = \frac{1}{|Aut(G)|} \sum_{\varphi \in Aut(G)} \tilde{h}(\varphi(u,v,w)).$$

If we can show that the 1-1-solution $a = (GP, h)$ is feasible, symmetrical and optimal, the Corollary is proved. From the proof of Corollary 3.2 we know that solution a is feasible, symmetrical and optimal VarMC-solution according to the assigned source-strength in equation (3.4). In fact it remains to be shown that these source-strengths represent a 1-1-MC instance. However, this is trivial since all source-strengths are equal to one in the given solution $\tilde{a}$, so the equation (3.4) leads to source-strengths of size one. ∎

3.1.3 Some Implications

Now that we have shown that there are always symmetrical MC-solutions with a maximal lower bound, we can describe some simple implications. The first implication discusses the relation between the 1-1-MC-bound and VarMC-bound:

Corollary 3.4
If a graph G is vertex-symmetric, the 1-1-MC-bound and the VarMC-bound are identical for any graph partitioning problem on this graph.

Proof:
The Corollary follows directly from the definition of symmetrical solutions and also from the fact, that there are always symmetrical solutions with maximal lower bound. A symmetrical solution of the VarMC-bound requires that $\forall v,w \in V : s(v) = s(w)$ if the graph is vertex-symmetric. Therefore, a symmetrical VarMC-solution on a vertex-symmetric graph is a 1-1-MC-solution (except for a scaling-factor). ■

So, if a graph is vertex-symmetric, the larger degree of freedom of the VarMC-instances does not have any advantage over the 1-1-instances.

The next Corollary restricts the possible flows which can be used in optimal MC-solutions:

Corollary 3.5
If a graph G is edge-symmetric, the flows of an optimal symmetrical MC-solution of any graph partitioning problem on this graph correspond to shortest paths.

Proof:
From Corollary 3.1 we can deduce that $\forall e,e' \in E : c(e) = c(e')$. So, if the flow of a commodity from vertex v to vertex w uses a path of length $d_{v,w}$ we have

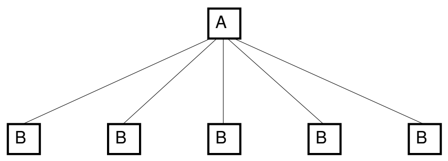
$$\begin{aligned} \sum_{v,w\in V} d_{v,w}s(v,w) &= \sum_{e\in E} c(e) \\ \Rightarrow \qquad C &= \frac{1}{|E|}\sum_{v,w\in V} d_{v,w}s(v,w) \end{aligned}$$

and it becomes obvious that the congestion is minimal if minimal $d_{v,w}$'s are used. ■

As a result, if a graph is edge-symmetric, we can focus our attention on symmetrical solutions which use shortest paths.

The two Corollaries above make use of the condition that the graph is vertex- or edge-symmetric. In both cases we can show that the used conditions are sharp in some sense of symmetry: It will be shown that only a slightly less symmetry does not be sufficient:

Figure 3.1: Example of a graph with two vertex-orbits, one edge-orbit and different 1-1-MC- and VarMC-bounds



Theorem 3.2
There is a graph with exactly two vertex-orbits and one edge-orbit where the VarMC-bound for the bisection problem is better than the 1-1-MC-bound.

Proof:
The theorem is proved by giving an example, this example is illustrated in Figure 3.1. Obviously, the graph has two vertex-orbits (vertex A and the vertices B) and is edge-symmetric. The 1-1-MC-bound on this graph for the bisection problem is $\frac{9}{5}$ (since $CF = 18$ and $C = 10$). The VarMC-bound is achieved with $s(A) = 1 \wedge s(B) = 0$, which gives a lower bound of 3 (since $CF = 3$ and $C = 1$). ■

In the next theorem we show that nearly edge-symmetry is not sufficient for the restriction on shortest paths:

Theorem 3.3
There is a graph with exactly one vertex-orbit and two edge-orbits where a symmetrical MC-solution using shortest paths does not give the best lower bound for the bisection problem.

Proof:
The theorem is proved by giving an example, this example is illustrated in Figure 3.2. Obviously, the graph has one vertex-orbits and two edge-orbits (the inner- and the circle-edges). We look at 1-1-MC-solutions (Since the graph is vertex-symmetric, the VarMC-bound is identical). The use of only shortest paths results in a congestion of 4 on the circle-edges and 6 on the inner-edges. With $CF = 32$ we have a bound of $\frac{32}{6} \approx 5.33$. However, any flow which uses an inner-edge could also be routed over two circle-edges. If we do this with $\frac{2}{3}$ of the 6 flows on every inner-edge it results in a congestion of $5\frac{1}{3}$ on the inner- and circle-edges. So, with this improvement we get a bound of $\frac{32}{5\frac{1}{3}} = 6$, which corresponds to the bisection width. ■

In Corollary 3.4 the relation of the 1-1-MC-bound to the VarMC-bound if the underlying graph is vertex-symmetric is discussed. It would be convenient if we could say

Figure 3.2: Example of a graph with one vertex-orbit and two edge-orbits where a solution using only shortest paths only does not give the best bound

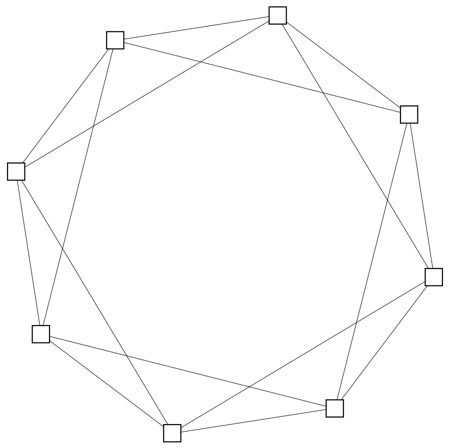
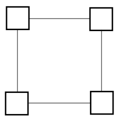


Figure 3.3: Example of a vertex- and edge-symmetric graph where the MVarMC-bound is better than the VarMC-bound



something similar about the MVarMC-bound. However the same statement does not apply:

Theorem 3.4
The MVarMC-bound can be better than the 1-1-MC-bound and VarMC-bound, even if the graph is vertex-symmetric and edge-symmetric.

Proof:
We give a simple example which proves this the theorem. We look at the graph in Figure 3.3 and the 4-partitioning problem. Since the graph is vertex-symmetric, we have 1-1-MC-bound = VarMC-bound. The 1-1-MC-bound is 3 (since $C = 4$ and $CF = 12$). The optimal MVarMC-bound is achieved if every vertex sends a commodity of size one to its two adjacent vertices. In doing this we get MVarMC-bound = 4 (since $C = 2$ and $CF = 8$). ■

Note that we have used the 4-partitioning problem in order to prove the theorem above, while in previous examples we always have used the simpler graph bisection problem.

In fact, it is an open problem if there is a vertex-symmetric and edge-symmetric graph where the MVarMC-bound is better than the 1-1-MC-bound in the case of the graph bisection problem. However, we can show that the MVarMC-bound is identical to the 1-1-MC-bound for graph bisection problems if the graph is vertex- and edge-symmetric if the following Conjecture is true:

Conjecture 3.1
Let $G = (V,E)$ be a vertex-symmetric graph. Let n_i be the number of vertices with distance i from one fixed vertex $v \in V$. Let D be the diameter of the graph. Then

$$\forall 0 < x < D: \quad \sum_{i=1}^{x} i \cdot n_i \geq \frac{x+1}{2} \sum_{i=1}^{x} n_i$$

holds.

We were not able to find any counter-example of this conjecture, but we also were not able to prove it.

Firstly, one general theorem is presented which we will use often:

Theorem 3.5
Let $f(x) : D \to \mathbb{R}$ be any function of the form $f(x) = \frac{ax+C_1}{bx+C_2}$ with $a,b,C_1,C_2 \in \mathbb{R}$ and $D \subseteq \mathbb{R}$. Then $f(x)$ has its global maximum and minimum at $\min\{x|x \in D\}$ *or* $\max\{x|x \in D\}$.

Proof:
The derivation of $f(x)$ is

$$\begin{aligned} f'(x) &= \frac{a(bx+C_2) - b(ax+C_1)}{(bx+C_2)^2} \\ &= \frac{aC_2 - bC_1}{(bx+C_2)^2} \end{aligned}$$

Obviously, the derivation has no zero-point (or $\forall x : f'(x) = 0$). So, $f(x)$ is monotonic increasing or decreasing. As a result of this it becomes clear that the global maximum and minimum have to be at the endpoints of the range D of x. ■

Now:

Theorem 3.6
The MVarMC-bound for the graph bisection problem is identical to the 1-1-MC-bound on vertex- and edge-symmetric graphs if Conjecture 3.1 is true.

Proof:
Firstly, the graph is edge-symmetric, so we use only shortest paths and get

$$\begin{aligned} C &= \frac{1}{m}\sum_v\sum_w s(v,w)\cdot d(v,w) \\ &= \frac{n}{m}\sum_w s(v,w)\cdot d(v,w) \ \ \forall v \in V \end{aligned}$$

where $d(v,w)$ is the distance of the vertices v and w. Secondly, we can assume that all $\max_w s(v,w)$ are equal since the graph is vertex-symmetric, and since the any MC-solution is scalable we assume $\max_w s(v,w)$=1. Then we have

$$\begin{aligned} CF &= \sum_v\sum_w s(v,w) - \frac{n}{2} \\ &= n\sum_w s(v,w) - \frac{n^2}{2}. \end{aligned}$$

If we put this together we have an MVarMC-bound Bd of

$$Bd = m\frac{\sum_w s(v,w) - \frac{n}{2}}{\sum_w s(v,w)\cdot d(v,w)}.$$

This formulation directly shows that $d(v,w) > d(v,w') \Rightarrow s(v,w) \le s(v,w')$. Furthermore, it becomes clear that the formulations corresponds to that one of Theorem 3.5 and we can follow that $s(v,w) \in \{0,1\}$. If we put this together we see that we can express the bound Bd as a function of a variable x where $s(v,w) = 1 \Leftrightarrow d(v,w) \le x$. If we do this we get

$$Bd(x) = m\frac{\sum_{i=0}^x n_i - \frac{n}{2}}{\sum_{i=0}^x i\cdot n_i}$$

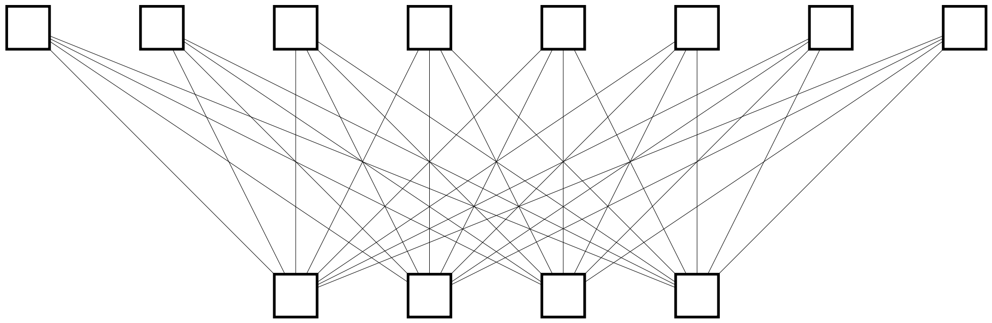
and we can follow that

$$\begin{aligned} & Bd(x+1) && \ge && Bd(x) \\ \Leftrightarrow \quad & \frac{n_{x+1}}{(x+1)n_{x+1}} && \ge && \frac{\sum_{i=0}^x n_i - \frac{n}{2}}{\sum_{i=0}^x i\cdot n_i} \\ \Leftrightarrow \quad & \sum_{i=0}^x i\cdot n_i && \ge && (x+1)\left(\sum_{i=0}^x n_i - \frac{n}{2}\right). \end{aligned}$$

Now, if Conjecture 3.1 is true we know that $\forall 0 < x < D: \quad \sum_{i=1}^x i\cdot n_i \ge \frac{x+1}{2}\sum_{i=1}^x n_i$ and we have

$$\begin{aligned} \sum_{i=0}^x i\cdot n_i &\ge \frac{x+1}{2}\sum_{i=1}^x n_i \\ &\ge \frac{x+1}{2}\left(\sum_{i=1}^x n_i + \sum_{i=0}^{x+1} n_i - n\right) \\ &\ge (x+1)\left(\sum_{i=0}^x n_i - \frac{n}{2}\right) \end{aligned}$$

Figure 3.4: Illustration of the $K_{4,8}$ graph



and the theorem is proved. ■

3.2 Some Specific Graphs

3.2.1 Bisection of the Complete Bipartite Graph $K_{a,b}$

In this subsection we analyze the bounds for the Graph Bisection problem of the complete bipartite graph $K_{a,b}$. Figure 3.4 shows the $K_{4,8}$ graph. The $K_{a,b}$ is generally defined as:

Definition 3.4
The complete bipartite graph $K_{a,b} = (V,E)$ is an undirected graph with

$$V = V_a \cup V_b \text{ with } V_a \cap V_b = \emptyset \wedge |V_a| = a \wedge |V_b| = b$$

and

$$E = \{\{v,w\} \,|\, v \in V_a \wedge w \in V_b\}\,.$$

So, the $K_{a,b}$ has $n = a+b$ vertices and ab edges. For simplification purposes, in the following we concentrate on the case of an even number of vertices. It is clear that the $K_{a,b}$ graph is edge-symmetric and has at most two vertex-orbits (exactly one vertex-orbit if and only if $a = b$). The optimal bisection width of the $K_{a,b}$ is:

Theorem 3.7
The bisection width of the $K_{a,b}$ graph is

$$bw(K_{a,b}) = \lceil \frac{ab}{2} \rceil$$

and is reached with a partition of the vertices, such that there are one half of the vertices of V_a and V_b in both partitions.

Proof:
We look at any partition $V = V_1 \cup V_2$ with $|V_1 \cap V_a| = \frac{a}{2} + x$ with $x \in \mathbb{Z}$ (if a is even, otherwise $x = x' + \frac{1}{2} \wedge x' \in \mathbb{Z}$). Then it follows that $|V_1 \cap V_b| = \frac{b}{2} - x$ and $|V_2 \cap V_a| = \frac{a}{2} - x$ and $|V_2 \cap V_b| = \frac{b}{2} + x$. With this partition, a cut-size of

$$\begin{aligned} & (\frac{a}{2}+x)(\frac{b}{2}+x) + (\frac{b}{2}-x)(\frac{a}{2}-x) \\ = \; & \frac{1}{2}ab + 2x^2 \end{aligned}$$

follows. Obviously, this is minimal with $x = 0$ or $x = \frac{1}{2}$ respectively and the theorem is proved. ■

1-1-MC-Bound

Now we examine the 1-1-MC-bound for the $K_{a,b}$ graph. From Corollary 3.5 it follows that we only have to look at shortest paths. So, the congestion can be easily computed:

Lemma 3.1
The congestion C of a symmetrical 1-1-MC-solution on a $K_{a,b}$ graph which uses only shortest paths is

$$C = \frac{2}{ab}(a^2 + b^2 + ab - a - b).$$

Proof:
From Corollary 3.5 it follows that $C = \frac{1}{|E|} \sum_{v,w \in V} d_{v,w}$. Using

$$d_{v,w} = \begin{cases} 0 & \text{if } v = w \\ 1 & v \in V_a \wedge w \in V_b \\ 2 & \text{else} \end{cases}$$

and $|E| = ab$ we have

$$\begin{aligned}
C &= \frac{1}{|E|}\sum_{v,w\in V} d_{v,w} \\
&= \frac{1}{ab}\left(\sum_{v\in V_a}\left(\sum_{w\in V_a, w\neq v} 2 + \sum_{w\in V_b} 1\right) + \sum_{v\in V_b}\left(\sum_{w\in V_a} 1 + \sum_{w\in V_b, w\neq v} 2\right)\right) \\
&= \frac{1}{ab}\left(a(2(a-1)+b) + b(a+2(b-1))\right) \\
&= \frac{1}{ab}\left(2a^2 - 2a + ab + ab + 2b^2 - 2b\right)
\end{aligned}$$

and the lemma is proved. ■

With this congestion the 1-1-MC-bound follows:

Theorem 3.8
The 1-1-MC-bound for the $K_{a,b}$ graph and the graph bisection problem is

$$\frac{1}{4}ab\frac{(a+b)^2}{a^2+b^2+ab-a-b}.$$

Proof:
From Corollary 2.1 we have $CF = \frac{1}{2}(a+b)^2$. Together with the congestion of the lemma above, the theorem follows. ■

VarMC-bound

Now, we consider the VarMC-bound. In contrast to the 1-1-MC-bound we also have to determine the source-strengths. From the vertex-symmetry of the graph it follows that we can look at restricted strengths with $\forall v, w \in V_a : s(v) = s(w)$ and $\forall v, w \in V_b : s(v) = s(w)$. So we use $s_a \in \mathbb{R}_{\geq 0}$ and $s_b \in \mathbb{R}_{\geq 0}$ with $\forall v \in V_a : s(v) = s_a$ and $\forall v \in V_b : s(v) = s_b$. Furthermore, since any MC-solution is scalable, we use $s_b = 1$ and the best s_a remains to be determined (if $s_b = 0$ is optimal, $s_a \to \infty$ will follow). Firstly, we present the resulting congestion:

Lemma 3.2
Any symmetrical VarMC-solution for the $K_{a,b}$ graph with $\forall v \in V_a : s(v) = s_a$ and $\forall v \in V_b : s(v) = 1$ which uses only shortest paths, has a congestion C of

$$C = \frac{1}{ab}\left(s_a a(2a-2+b) + b(2b-2+a)\right).$$

Proof:
As was the case with the 1-1-MC-solution, the lemma follows from Corollary 3.5 and the known distances of the $K_{a,b}$ graph. ∎

With this congestion the VarMC-bound follows:

Theorem 3.9
The VarMC-bound for the $K_{a,b}$ graph with $a \geq b$ and the graph bisection problem is

$$\frac{1}{2}ab\frac{(a+b)}{a+2b-2}$$

and it is reached by a VarMC-solution where the vertices of the larger part of the bipartite graph send nothing.

Proof:
Using the source-strengths s_a and $s_b = 1$, we have a Cut-Flow of $CF = \frac{a+b}{2}(as_a + b)$. When we use the congestion of the lemma above, a bound $Bd(s_a)$ follows with

$$Bd(s_a) \quad = \quad \frac{1}{2}ab\frac{(a+b)(as_a+b)}{s_a a(2a-2+b)+b(2b-2+a)}$$

with $s_a \in \mathbb{R}_{\geq 0}$. In order to achieve the maximal bound, we look at the first derivative:

$$Bd'(s_a) = \frac{1}{2}ab\frac{(a+b)(b-a)}{(s_a a(2a-2+b)+b(2b-2+a))^2}$$

Since the enumerator of $Bd'(s_a)$ is independent of s_a and $Bd'(s_a) \leq 0 \Leftrightarrow b \leq a$ it follows that a minimal s_a gives the maximal bound, if $a \geq b$. As a result of this we choose $s_a = 0$ and a bound of $\frac{1}{2}ab\frac{a+b}{a+2b-2}$ follows. ∎

MVarMC-bound

Now, we consider the MVarMC-bound. Again, due to the symmetries of the vertices, we can restrict the possible set of source-strengths to the four variables $s_{aa}, s_{ab}, s_{ba}, s_{bb} \in \mathbb{R}_{\geq 0}$ with $\forall v \in V_x, w \in V_y : s(v,w) = s_{xy}$ with $x, y \in \{a,b\}$. Without loss of generality, we assume $a \geq b$. This leads to a further restriction on the source-strengths:

Lemma 3.3
Let $s_{aa}, s_{ab}, s_{ba}, s_{bb} \in \mathbb{R}_{\geq 0}$ be source-strengths of an MVarMC-instance for the $K_{a,b}$ with $a \geq b$. Then for every optimal MVarMC-instance

$$s_{ab} \leq s_{aa} \quad \wedge \quad s_{bb} \leq s_{ba}$$

holds.

Proof:
We look at any MVarMC-instance which does not fulfill $s_{ab} \leq s_{aa}$. Using our Cut-Flow Theorem 2.2 we have

$$CF = a(as_{aa} + bs_{ab} - \frac{n}{2}s_{ab}) + b(as_{ba} + bs_{bb} - \frac{n}{2}\max\{s_{ba}, s_{bb}\}).$$

Since $a \geq b$ we have $\frac{n}{2} \geq b$. Therefore, decreasing s_{ab} increases the Cut-Flow and decreases the congestion, which means that the bound is increased. Therefore, if the condition $s_{ab} \leq s_{aa}$ is not fulfilled, an MVarMC-solution with a better bound exists. The same considerations can be applied to the $s_{bb} \leq s_{ba}$ condition. ■

Now, using this restriction, a closed form of the Cut-Flow and the congestion can be given. Combining these together, we get the following bound:

Lemma 3.4
Let $s_{aa}, s_{ab}, s_{ba}, s_{bb} \in \mathbb{R}_{\geq 0}$ be source-strengths of an MVarMC-instance for the $K_{a,b}$ with $a \geq b$ and $s_{ab} \leq s_{aa} \wedge s_{bb} \leq s_{ba}$. This instance gives a lower bound on the bisection width of

$$ab \cdot \frac{a(\frac{a-b}{2}s_{aa} + bs_{ab}) + b(\frac{a-b}{2}s_{ba} + bs_{bb})}{a\left(2(a-1)s_{aa} + s_{ab}\right) + b\left(as_{ba} + 2(b-1)s_{bb}\right)}$$

Proof:
If we use a Cut-Flow CF and a congestion C the bound is $\frac{CF}{C}$. Using the given source-strengths the Cut-Flow follows directly from Theorem 2.2 with

$$CF = a(\frac{a-b}{2}s_{aa} + bs_{ab}) + b(\frac{a-b}{2}s_{ba} + bs_{bb}).$$

In order to get the congestion, we use the fact that the graph is edge-symmetric. Therefore, shortest-paths gives the best congestion and the congestion can be calculated using the distances of the graph and we get

$$C = \frac{1}{ab}a\left(2(a-1)s_{aa} + s_{ab}\right) + b\left(as_{ba} + 2(b-1)s_{bb}\right).$$

■

So, we now have an exact formula for the bound of an MVarMC-solution using source-strengths $s_{aa}, s_{ab}, s_{ba}, s_{bb} \in \mathbb{R}_{\geq 0}$. The source-strengths, such that the bound is maximized, remain to be selected.

With the help of Theorem 3.5 we can determine the best source-strengths for the MVarMC-bound. The following theorem summarizes the result:

Theorem 3.10
The MVarMC-bound for the bisection width of the $K_{a,b}$ graph is identical to the VarMC-bound.

Proof:
From Lemma 3.4 we know that the bound for any MVarMC-solution with the corresponding source-strengths is

$$\begin{aligned} Bd(s_{aa}, s_{ab}, s_{bb}, s_{ba}) &= ab \cdot \frac{a(\frac{a-b}{2}s_{aa} + bs_{ab}) + b(\frac{a-b}{2}s_{ba} + bs_{bb})}{a\,(2(a-1)s_{aa} + s_{ab}) + b\,(as_{ba} + 2(b-1)s_{bb})} \\ & ab \cdot \frac{a\frac{a-b}{2}s_{aa} + abs_{ab} + b\frac{a-b}{2}s_{ba} + b^2 s_{bb}}{2a(a-1)s_{aa} + as_{ab} + bas_{ba} + 2b(b-1)s_{bb}}. \end{aligned}$$

with $s_{ab} \leq s_{aa} \wedge s_{bb} \leq s_{ba}$. The function $Bd(s_{aa}, s_{ab}, s_{bb}, s_{ba})$ takes the form as it is required in Theorem 3.5 for each of the four variables s_{aa}, s_{ab}, s_{bb}, and s_{ba}. So, $Bd(s_{aa}, s_{ab}, s_{bb}, s_{ba})$ is maximal at the endpoints of the ranges of the four variables. We know, that the case of $s_{aa} \longrightarrow \infty$ ($s_{ba} \longrightarrow \infty$) corresponds to the case with $s_{ba} = 0$ ($s_{aa} = 0$). So we only have to compare the following four cases:

1. $s_{aa} = s_{ab} = 0$ and $s_{ba} = s_{bb} = 1$ gives $Bd_1 = \frac{1}{2}ab\frac{a+b}{a+2b-2}$
2. $s_{aa} = s_{ab} = 0$ and $s_{ba} = 1 \wedge s_{bb} = 0$ gives $Bd_2 = \frac{1}{2}ab\frac{a-b}{a}$
3. $s_{ba} = s_{bb} = 0$ and $s_{aa} = s_{ab} = 1$ gives $Bd_3 = \frac{1}{2}ab\frac{a+b}{2a-2+a}$
4. $s_{ba} = s_{bb} = 0$ and $s_{aa} = 1 \wedge s_{ab} = 0$ gives $Bd_4 = \frac{1}{2}ab\frac{a-b}{2a-2}$

Then, the MVarMC-bound is $\max\{Bd_1, Bd_2, Bd_3, Bd_4\}$. Firstly, since we assume without loss of generality that $a \geq b$, it is obvious that $Bd_1 \geq Bd_3$. Secondly, since $a \geq 1$, it is also obvious that $Bd_2 \geq Bd_4$. Bd_1 and Bd_2 remain to be compared:

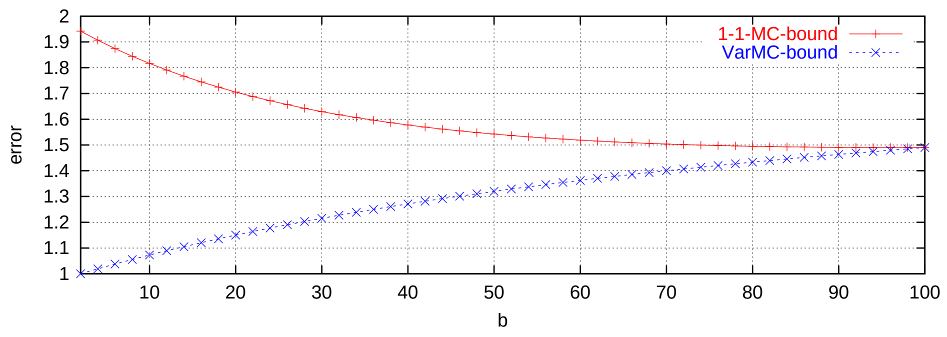
$$\begin{aligned} Bd_1 &\geq Bd_2 \\ \Leftrightarrow \frac{1}{2}ab\frac{a+b}{a+2b-2} &\geq \frac{1}{2}ab\frac{a-b}{a} \\ \Leftrightarrow \frac{a+b}{a+2b-2} &\geq \frac{a-b}{a} \\ \Leftrightarrow (a+b)a &\geq (a-b)(a+2b-2) \\ \Leftrightarrow 0 &\geq -2a - 2b^2 + 2b \end{aligned}$$

So, Bd_1 is always at least as big as Bd_2. Therefore, MVarMC-bound = $Bd_1 = \frac{1}{2}ab\frac{a+b}{a+2b-2}$ which is equal to the VarMC-bound, as we already know from Theorem 3.9. ■

Table 3.1: Bounds and their errors on the bisection width of the $K_{a,b}$

	result	asympt. error $b = \alpha a$	asympt. error $b = o(a)$
1-1-MC	$\frac{1}{4}ab\frac{(a+b)^2}{a^2+b^2+ab-a-b}$	$2 \cdot \frac{\alpha^2+\alpha+1}{\alpha^2+2\alpha+1}$	2
VarMC	$\frac{1}{2}ab\frac{a+b}{a+2b-2}$	$\frac{1+2\alpha}{1+\alpha}$	1
MVarMC	$\frac{1}{2}ab\frac{a+b}{a+2b-2}$	$\frac{1+2\alpha}{1+\alpha}$	1

Figure 3.5: The error of the MC-bounds for the bisection of the $K_{a,b}$ for $a = 100$, depending on b



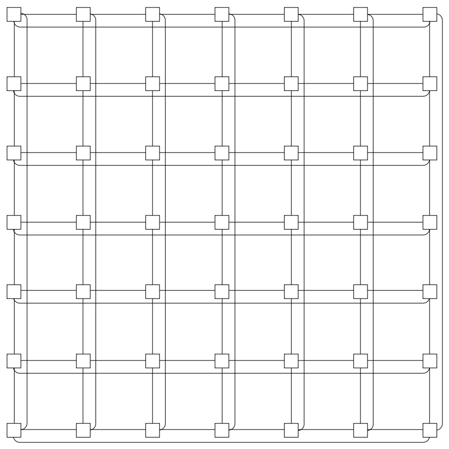
Summary

In conclusion, we have shown the 1-1-MC-bound, VarMC-bound and MVarMC-bound for the bisection width of the $K_{a,b}$ graph. In order to complete the analyses, we would also like to give the asymptotic errors of the bounds using two cases, firstly when $b = \alpha a$ with $\alpha \in (0, 1]$ and secondly when $b = o(a)$. The error of a bound is the relation $\frac{\text{optimum}}{\text{bound}}$, so an error of 1 means that the bound is exact while the bigger the error is, the worse the bound is. Table 3.1 summarizes the results while Figure 3.5 illustrates the dependence of the errors on b.

In order to sum up the results of the bisection width of the $K_{a,b}$ graph, the following observations can be made:

- The MVarMC-bound is identical to the VarMC-bound.
- The VarMC-bound is asymptotically optimal when $b = o(a)$, while it has an error of 1.5 when $b = a$.

Figure 3.6: Illustration of the 7×7-torus



- The 1-1-MC-bound has an error of 2 if $b = o(a)$ and 1.5 if $b = a$.

Finally, it is interesting to note the fact, that the VarMC-bound and the 1-1-MC-bound are identical in the case of $b = a$ could be concluded directly from the fact, that the $K_{a,b}$ graph is vertex-symmetric in this case. Therefore, the $K_{a,b}$ graph is a good example for the fact that the VarMC-bound could be the better the less vertex-symmetric the graph is when compared to the 1-1-MC-bound.

3.2.2 The k-partitioning of the $a \times a$-Torus

In this section we present the MC-bounds for the k-partitioning problem with the $a \times a$-torus. Figure 3.6 shows the 7×7-torus. Firstly we give a definition of the well-known torus:

Definition 3.5
The $a \times a$-torus$= (V,E)$ is an undirected graph with the set of vertices

$$V = \{(x,y) | x,y \in \{0,\ldots,a-1\}\}$$

and the set of edges

$$\begin{aligned} E \; &= \{\{(x,y),(x',y')\} \mid \; \big(|x-x'| \; mod \; (a-2) = 1 \wedge y = y'\big) \\ &\qquad \vee \big(x = x' \wedge |y-y'| \; mod \; (a-2) = 1\big)\}. \end{aligned}$$

The $a \times a$-torus has a^2 vertices and $2a^2$ edges. The asymptotic cut-size for the k-partitioning of the $a \times a$-torus is $2a\sqrt{k}$, which is shown in [BR97]. It is vertex-symmetric and edge-symmetric. So the VarMC-bound is equal to the 1-1-MC-bound.

We can therefore restrict our examinations to shortest paths and we can calculate the congestions by simply using the distances.

In the following we often need to know how many vertices with a specific distance to a given vertex exist. So first of all, we will present a result in relation to this:

Lemma 3.5
Let $D_a(x) : \{1,\dots,a\} \to \mathbb{N}$ be the number of vertices with distance x to any fixed vertex inside the $a \times a$-torus. Then

$$D_a(x) = \begin{cases} 4x & \text{if } x < \frac{a}{2} \\ 2a-2 & \text{if } x = \frac{a}{2} \\ 4a-4x & \text{if } \frac{a}{2} < x < a \\ 1 & \text{if } x = a \end{cases}$$

holds if a is even. And $\forall x < \frac{a}{2} : D_a(x) = 4x$ also holds if a is odd.

Proof:
We start with the case when $x < \frac{a}{2}$. We look at vertex $v = \left\{\frac{k}{2}, \frac{k}{2}\right\}$. The vertices $\left\{\frac{k}{2}+s, \frac{k}{2}+t\right\}$ with $-x \le s,t \le x$ and $|s|+|t| = x$ are the set of vertices which have distance x to the vertex v. There are $4x$ possible pairs s,t, so $D_a(x) = 4x$ is shown if $x < \frac{a}{2}$. Obviously, this is also true if a is odd.

If $x = \frac{a}{2}$ the same considerations as above can be applied. We only have to consider the fact that the vertices with $s,t = \frac{a}{2}$ do not exist. There are exactly two such vertices, so we have $D_a(\frac{a}{2}) = 4 \cdot \frac{a}{2} - 2 = 2a-2$.

If $\frac{a}{2} < x < a$, the same considerations as in the case of $x < \frac{a}{2}$ can be applied. Now we have restrictions $-\frac{a}{2} \le s,t < \frac{a}{2}$ and $|s|+|t| = x$. So there are $4a-4x$ pairs of s,t which fulfill these conditions.

Finally, if $x = a$ only the vertex $\{0,0\}$ has distance a from vertex $v = \left\{\frac{k}{2}, \frac{k}{2}\right\}$. ■

1-1-MC-bound

We start with an examination of the 1-1-MC-bound. Since shortest paths are used the congestion can be calculated using the number of vertices in the different distances:

Lemma 3.6
A symmetrical 1-1-MC-solution on the $a \times a$-torus using only shortest paths has congestion C with

$$C = \frac{1}{4}a^3$$

if a is even.

Proof:
The congestion can be calculated using

$$C = \frac{1}{2a^2} \cdot a^2 \sum_{x=1}^{a} x \cdot D_a(x).$$

A careful examination of this sum gives $C = \frac{1}{4}a^3$. ∎

With this congestion, the bound follows immediately:

Theorem 3.11
For the k-partitioning of an $a \times a$-torus when a is even and $k|a^2$

$$\textit{1-1-MC-bound} = (1 - \frac{1}{k}) \cdot 4a$$

holds.

Proof:
The Cut-Flow of an 1-1-MC-solution is $(1 - \frac{1}{k})a^4$. Using the congestion of the lemma above the theorem follows immediately. ∎

MVarMC-bound

In the following we present the analysis of the MVarMC-bound for the k-partitioning of an $a \times a$-torus. We have to determine source-strengths and the resulting congestion so that the bound is maximized. From the edge-symmetry it follows that the congestion can be computed using only the distances, which we have already done for the 1-1-MC-bound. From the vertex-symmetry it follows that we can restrict our exploration to determining the source-strengths of only one sender-vertex.

The following Lemma is the first step in determining the source-strengths:

Lemma 3.7
Any symmetric MVarMC-solution for the $a \times a$-torus with maximal bound fulfills

$$\forall v, w, w' \in V : \ d(v,w) > d(v,w') \Rightarrow s(v,w) \leq s(v,w')$$

where $d(v,w)$ is the distance of the vertices v and w.

Proof:
We assume we have an MVarMC-solution which does not fulfill the condition above, i.e. $\exists v,w,w' \in V:\ d(v,w) > d(v,w') \wedge s(v,w) > s(v,w')$. Let v,w,w' be such a triple of vertices. From the edge-symmetry a congestion C with

$$C = \frac{1}{2a} \sum_{v\in V} \sum_{w\in V} d(v,w) \cdot s(v,w)$$

follows. So, if we exchange $s(v,w)$ and $s(v,w')$, the congestion will decrease. Furthermore, the Cut-Flow will not change. Therefore, this exchange would give an MVarMC-solution with a better bound. ∎

The next lemma shows that the source-strengths depend only on the distance of the two vertices. And it shows that we can restrict the source-strengths onto the set $\{0,1\}$.

Lemma 3.8
A symmetrical MVarMC-solution with maximal bound which fulfills

$$\forall v,w \in V: \quad s(v,w) = f(d(v,w))$$

with $f: \mathbb{N} \to \{0,1\}$ *exists.*

Proof:
In order to prove the lemma we look at the Cut-Flow and congestion of a symmetrical MVarMC-solution. Since any MVarMC-solution is scalable we can restrict the source-strengths such that $\forall v,w \in V: s(v,w) \leq 1$ and $\forall v: \exists w: s(v,w) = 1$. As we have seen in the proof of the previous lemma, we have $C = \frac{1}{2a}\sum_v \sum_w d(v,w)\cdot s(v,w)$. The Cut-Flow CF is

$$CF = \sum_{v\in V} \left(\sum_{w\in V} s(v,w) - \frac{a^2}{k} \right)$$

using the above restrictions on the source-strengths. So, altogether the bound is

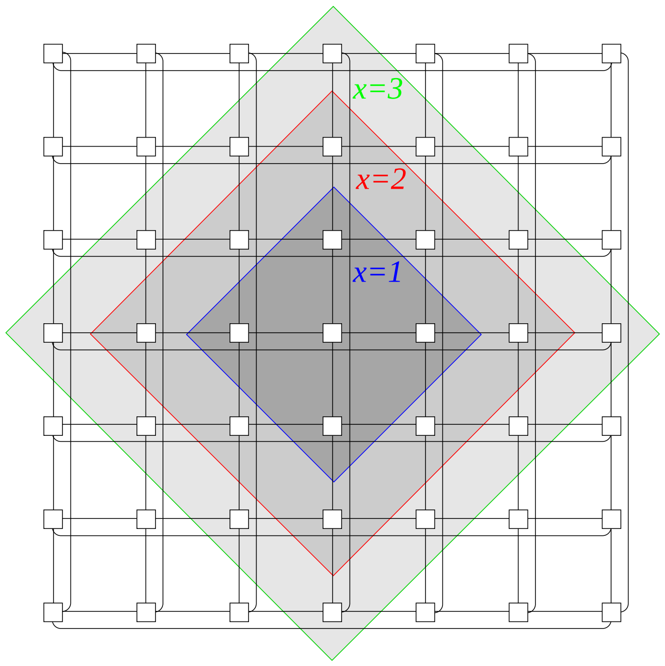
$$Bd = \frac{CF}{C} = 2a \frac{\sum_{v\in V} \left(\sum_{w\in V} s(v,w)\right) - \frac{a^4}{k}}{\sum_{v\in V} \sum_{w\in V} d(v,w)\cdot s(v,w)}.$$

Obviously, this formula fulfills the condition of Theorem 3.5 for each of the source-strengths $s(v,w)$. So, $\forall v,w \in V: s(v,w) \in \{0,1\}$ follows directly for a solution with maximal bound.

Furthermore, the derivation for every single $s(v,w)$ is

$$Bd'(s(v,w)) = \frac{C_{rest} - d(v,w)\cdot CF_{rest}}{C^2}$$

Figure 3.7: Illustration of the set of destinations from one sending vertex depending on the variable x



where $C_{rest}(CF_{rest})$ is the congestion (Cut-Flow) without the summand which contains $s(v,w)$. If $Bd'(s(v,w)) > 0$ we have to use $s(v,w) = 1$ and if $Bd'(s(v,w)) < 0$ we have to use $s(v,w) = 0$ in order to maximize the bound. As you can see in the formula, the sign of $Bd'(s(v,w))$ only depends on $d(v,w)$, C_{rest} and CF_{rest}. So, a pair of vertices with the same distance has the same sign. The lemma follows. ■

The following corollary sums up the results above:

Corollary 3.6
There is a symmetrical MVarMC-instance with maximal bound for the k-partitioning of the $a \times a$-torus with source-strengths

$$\forall v,w \in V: \quad s(v,w) = \begin{cases} 1 & \text{if } d(v,w) \leq x \\ 0 & \text{else} \end{cases}$$

with $x \in \{1,\ldots,a\}$.

Figure 3.7 illustrates what happens inside the torus concentrating on one sending vertex. The x which gives the maximal lower bound remains to be quantified. In order to do this, we present a closed form of the bound:

Lemma 3.9
The bound $Bd(x)$ of an MVarMC-solution with source-strengths which correspond to Corollary 3.6 is

$$Bd(x) = 3\frac{a^2(2x^2k + 2xk - a^2 + k)}{kx(x+1)(2x+1)}$$

if $x < \frac{a}{2}$.

Proof:
The bound is given with Cut-Flow/congestion. The Cut-Flow $CF(x)$ is

$$\begin{aligned} CF(x) &= a^2\left(\sum_{i=1}^{x} D_a(i) + 1 - \frac{a^2}{k}\right) \\ &= a^2(2x(x+1) - \frac{a^2}{k} + 1) \end{aligned}$$

if $x < \frac{a}{2}$, using the number of vertices with distance $D_a(x)$ according to Lemma 3.5. Correspondingly, congestion $C(x)$ is

$$\begin{aligned} C(x) &= \frac{a^2}{2a^2}\sum_{i=1}^{x} i \cdot D_a(i) \\ &= \frac{1}{3}x(x+1)(2x+1) \end{aligned}$$

■

Finally, we can give asymptotic results for the MVarMC-bound:

Theorem 3.12
The asymptotic MVarMC-bound for the k-partitioning of an $a \times a$-torus with $k|a$ is

$$\textit{MVarMC-bound} = \sqrt{\frac{8}{3}ka} + O(1)$$

if $k > 6$. *It is reached by a solution corresponding to Corollary 3.6 with*

$$x = \left\lceil \sqrt{\frac{6}{4k}}a \right\rceil.$$

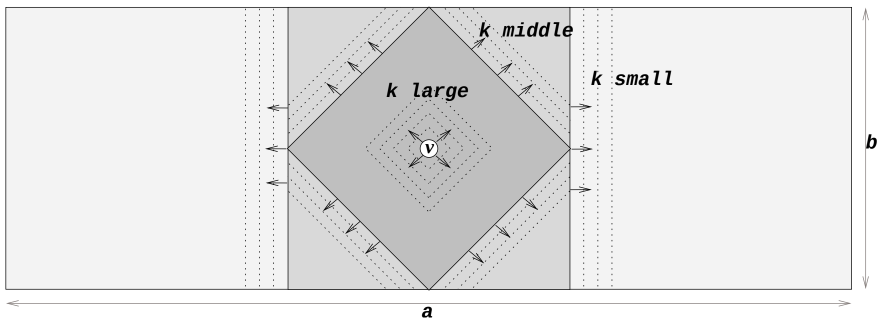
Proof:
The theorem can be proved by a careful analysis of the bound $Bd(x)$ of Lemma 3.9. ■

So, we have seen that the $a \times a$-torus is an example of a vertex-symmetric and edge-symmetric graph where the MVarMC-bound is up to a factor of $\sqrt{\frac{1}{6}k}$ better than the 1-1-MC-bound and VarMC-bound.

Table 3.2: Summary of the asymptotic results for k-partitioning an $a \times a$-torus with $a \to \infty$ and $k > 6$

method	result	error
optimal	$\sqrt{4k}a$	
1-1-MC-bound	$4a$	$\sqrt{\frac{1}{4}k} \quad = 0.5 \cdot \sqrt{k}$
VarMC-bound	$4a$	$\sqrt{\frac{1}{4}k} \quad = 0.5 \cdot \sqrt{k}$
MVarMC-bound	$\sqrt{\frac{8}{3}ka}$	$\sqrt{\frac{3}{2}} \quad \approx 1.22$

Figure 3.8: Illustration of the behavior inside a $a \times b$-torus



Summary and Outlook

Table 3.2 summarizes the results in this section. It demonstrates that the MVarMC-bound is definitely superior to the 1-1-MC-bound and VarMC-bound. The larger k is, the bigger the superiority of the MVarMC-bound.

Furthermore, we have also looked at $a \times b$-tori and a^d-tori. We do not give the details on those analysis but we want to briefly introduce the main results: First of all, both are vertex-symmetric, so the VarMC-bound is equal to the 1-1-MC-bound.

When we look at $a \times b$-tori, the behavior of the MVarMC-bound is similar to the $a \times a$-torus if k is large. I.e. with large k every vertex sends commodities of unit size to every other vertex up to an distance x. This x increases while k decreases. At the point where $x = \frac{1}{2}b$, where we assume $b \leq a$, this behavior changes. In the next phase every vertex $v = (a_v, b_v)$ send commodities of unit size to every vertex $w = (a_w, b_w)$ if $|a_v - a_w| \leq \frac{b}{2}$ and distance $d(v,w) \leq x$ where x increases while k decreases. Now, this behavior changes at the point where $x = \frac{1}{2}(a+b)$. In the following final phase every vertex $v = (a_v, b_v)$ sends commodities of unit size to every vertex $w = (a_w, b_w)$

if $|a_v - a_w| \leq x$ where x increases while k decreases. This behavior is illustrated in Figure 3.8.

When we look at the a^d-torus, the behavior of the MVarMC-bound corresponds to the behavior inside a $a \times a$-torus: Every vertex sends commodities of unit size to every other vertex up to a distance x. This distance x increases while k decreases. This behavior gives an MVarMC-bound of $da^{d-1}\sqrt[d]{\frac{4}{(d+1)!}k}$. The asymptotic cut-size of an a^d-torus is $da^{d-1}\sqrt[d]{k}$; therefore, the MVarMC-bound has an asymptotic error of $\sqrt[d]{\frac{1}{4}(d+1)!}$. Compared to this, the 1-1-MC-bound and VarMC-bound have an asymptotic error of $\frac{1}{4}d\sqrt[d]{k}$.

3.2.3 The k-partitioning of the $a \times a$-Grid

Now that we have examined the quadratic torus, the grid suggests itself for examination since it is very similar to the torus. Unfortunately, the analysis of the MC-bounds are more complex since the grid is neither vertex-symmetric nor edge-symmetric.

Definition 3.6
The $a \times a$-grid$= (V,E)$ is an undirected graph with the set of vertices

$$V = \{(x,y) | x,y \in \{0,\ldots,a-1\}\}$$

and the set of edges

$$E = \left\{ \{(x,y),(x',y')\} \mid \big(|x-x'| = 1 \wedge y = y'\big) \vee \big(x = x' \wedge |y-y'| = 1\big) \right\}.$$

The $a \times a$-grid has a^2 vertices and $2a(a-1)$ edges.

1-1-MC-bound and VarMC-bound

Since the grid is not vertex-symmetric, we cannot directly conclude that the VarMC-bound is equal to the 1-1-MC-bound. But of course, VarMC-bound $\geq$ 1-1-MC-bound holds, as it always does. We will show in this section, that the VarMC-Bound is equal to the 1-1-MC-bound in the case of the quadratic grid. In order to do this, we first give an upper bound on the VarMC-bound and then a lower bound on the 1-1-MC-bound. Since the upper and lower bounds are identical, the exact bounds are shown. In order to simplify the formulas, we assume in this subsection that a is even and $k|a^2$.

Lemma 3.10
For the k-partitioning of an $a \times a$-grid it holds that

$$\textit{VarMC-bound} \leq (1 - \frac{1}{k})2a.$$

Proof:
Let $s(v)$ be the source-strengths of any VarMC-solution. Now we look at a set E' of edges with $E' = \left\{\{(x,y),(x,y')\} \,|y = \frac{a}{2} - 1 \wedge y' = \frac{a}{2}\right\}$. Removing these edges partitions the grid into two sub-grids of size $a \times \frac{a}{2}$. So, for any sender v there are $\frac{1}{2}a^2$ vertices in the other partition which have to receive $s(v)$ of the commodity each. So we can sum up a minimal flow over the edges of E':

$$\sum_{e \in E'} c(e) \geq \frac{1}{2}a^2 \sum_{v \in V} s(v)$$

This gives us the following lower bound on the congestion:

$$\begin{aligned} C &\geq \frac{1}{|E'|} \sum_{e \in E'} c(e) \\ &\geq \frac{1}{2}a \sum_{v \in V} s(v) \end{aligned}$$

For the Cut-Flow CF of the VarMC-solution it holds that $CF = (1 - \frac{1}{k})a^2 \sum_v s(v)$. So for the bound Bd of any VarMC-solution it holds that

$$\begin{aligned} Bd &= \frac{CF}{C} \\ &\leq \frac{(1-\frac{1}{k})a^2 \sum_v s(v)}{\frac{1}{2}a \sum_v s(v)} \\ &= (1 - \frac{1}{k})2a \end{aligned}$$

which proves the lemma. ■

Next, we give a lower bound on the 1-1-MC-bound.

Lemma 3.11
For the k-partitioning of an $a \times a$-grid it holds that

$$\textit{1-1-MC-bound} \geq (1 - \frac{1}{k})2a.$$

Proof:
In order to prove the lemma we give an 1-1-MC-solution with bound $(1 - \frac{1}{k})2a$. An 1-1-MC-solution is given by determining the paths of the commodities. We use only shortest paths where every commodity first uses the edges of set E_x and then the edges of set E_y with $E_x = \{\{(x,y),(x',y)\} \in E\}$ and $E_y = \{\{(x,y),(x,y')\} \in E\}$. In order to analyze the congestion, we use an additional partition of the edges with

$$\begin{aligned} \forall i \in \{1,\ldots,a-1\}: \qquad & E_{xi} := \left\{\{(x,y),(x',y)\} \in E_x | x = i-1 \wedge x' = i\right\} \\ \wedge \quad & E_{yi} := \left\{\{(x,y),(x,y')\} \in E_y | y = i-1 \wedge y' = i\right\}. \end{aligned}$$

With this partitioning and the used paths given we have

$$\forall z \in \{x,y\}\, \forall i \in \{1,\ldots,a-1\} : \quad e \in E_{zi} \Rightarrow c(e) = 2i(a-i)a.$$

Therefore, the maximal congestion exists at the edges when $i = \frac{a}{2}$ and when we have $C = \frac{1}{2}a^3$. With the Cut-Flow CF of $CF = (1-\frac{1}{k})a^4$ of any 1-1-MC-instance, we have a bound of

$$\frac{(1-\frac{1}{k})a^4}{\frac{1}{2}a^3} = (1-\frac{1}{k})2a$$

which proves the lemma. ■

So, the exact 1-1-MC-bound and VarMC-bound follow with:

Theorem 3.13
For the k-partitioning of an $a \times a$-grid it holds that

$$\text{1-1-MC-bound} = \text{VarMC-bound} = (1-\frac{1}{k})2a.$$

Proof:
The theorem follows directly from the two previous lemmas. ■

MVarMC-Bound

Now we analyze the MVarMC-bound for the k-partitioning of an $a \times a$-grid. Our main goal is to provide asymptotic results, since the exact formulas are quite complex and give no further understanding of what is happening. We start with the presentation of an upper bound on the MVarMC-bound:

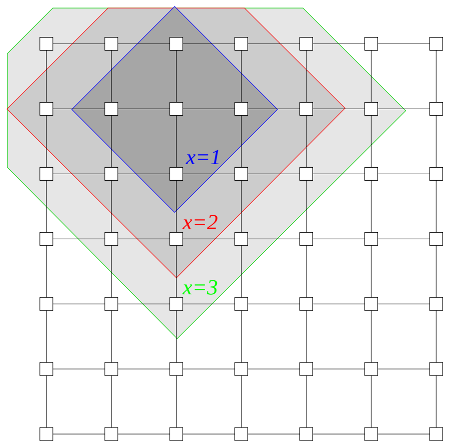
Lemma 3.12
Let Bd_G be the MVarMC-bound for the k-partitioning of an $a \times a$-grid and Bd_T the MVarMC-bound of an $a \times a$-torus. Then

$$Bd_G \leq Bd_T$$

holds.

Proof:
Since the $a \times a$-grid is a subgraph of the $a \times a$-torus with the same vertices, any MC-solution for the $a \times a$-grid can be applied to the $a \times a$-torus and the same bound will

Figure 3.9: Illustration of the set of destinations from one sending vertex which depends on the variable x inside a grid



be delivered. So, the MVarMC-solution with maximal bound can also be applied, and the lemma follows. ■

A direct conclusion which can be taken from the above lemma is that the asymptotic MVarMC-bound for the k-partitioning of an $a \times a$-grid is at most $\sqrt{\frac{8}{3}k}a$. Next we give a lower bound on the asymptotic MVarMC-bound:

Lemma 3.13
For the k-partitioning of an $a \times a$-grid it holds that

$$\text{MVarMC-bound} \geq \left(\sqrt{\frac{8}{3}k} + \sqrt{\frac{96}{k}} - 8 \right) a + O(1)$$

Proof:
We present an MVarMC-solution which also provides a corresponding bound. In principle we apply the optimal MVarMC-solution of the $a \times a$-torus, i.e. every vertex sends a commodity of size one to every other vertex with a distance which is less than or equal to x. Figure 3.9 illustrates this. We also apply the paths of the commodities of the $a \times a$-torus. Obviously, the resulting congestion on the edges cannot be bigger than the congestion in the case of the torus, since any commodity which is routed on the grid is also routed on the identical edges of the torus. In fact, the congestion on the grid could actually be smaller, since less commodities have to be routed at the border of the grid . So we have

$$C_G(x) \leq C_T(x) = \frac{1}{3}x(x+1)(2x+1)$$

where $C_G(x)$ is the congestion on the grid and $C_T(x)$ is the corresponding congestion on the torus; $C_T(x)$ is known from Lemma 3.9.

On the other hand, the Cut-Flow CF_G of the grid is less than the corresponding Cut-Flow CF_T of the torus. But we give a lower bound on the CF_G by taking into account only those vertices with a distance of at least x from the border of the grid. So we have

$$CF_G(x) \geq (a-2x)^2(2x(x+1) - \frac{a^2}{k} + 1)$$

which follows directly from the CF_T in Lemma 3.9.

If we put the congestion and the Cut-Flow together and use $x = \alpha a$, we have a bound Bd_G on the grid of

$$Bd_G \geq 3\frac{(a-2\alpha a)^2(2\alpha a(\alpha a+1) - \frac{a^2}{k} + 1)}{\alpha a(\alpha a+1)(2\alpha a+1)}$$

which gives an asymptotic behavior of

$$Bd_G \geq \left(\sqrt{\frac{8}{3}k} + \sqrt{\frac{96}{k}} - 8\right)a + O(1)$$

with $\alpha = \sqrt{\frac{6}{4k}}$, the same selection as for the torus. ■

So, we have presented upper and lower bounds on the MVarMC-bound for the k-partitioning of the $a \times a$-grid. The following theorem sums up the results:

Theorem 3.14
For the MVarMC-bound Bd_G for the k-partitioning of the $a \times a$-grid with $k|a$

$$\left(\sqrt{\frac{8}{3}k} + \sqrt{\frac{96}{k}} - 8\right)a + O(1) \leq Bd_G \leq \sqrt{\frac{8}{3}k}a + O(1)$$

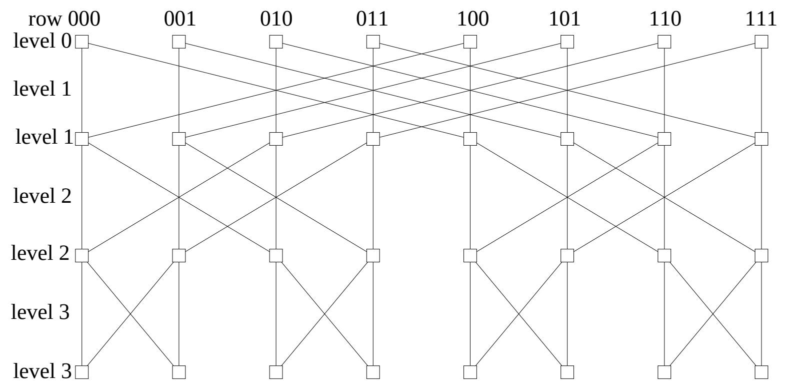
holds if $k > 6$.

So, if $k = \omega(1)$ the asymptotic bound is shown exactly, and it is identical to the asymptotic bound for the k-partitioning of an $a \times a$-torus.

3.2.4 Bisection of the Butterfly

In this section we present analyses of the butterfly network. The butterfly network is a very well known graph. There are two versions, one with wraparound-edges and one without wraparound-edges. With wraparound-edges the network is vertex- and edge-symmetric, while the butterfly-network without wraparound-edges is neither vertex- nor edge-symmetric. We will focus on the version without wraparound-edges in the following.

Figure 3.10: Butterfly of dimension 3



Definition 3.7
The d-dimensional butterfly $BF(d) = (V,E)$ is an undirected graph with vertices

$$V = \left\{(w,i) | i \in \{0,\dots,d\} \wedge w \in \{0,\dots,2^d - 1\}\right\}$$

and edges

$$E = \left\{\left\{(w,i),(w',i')\right\} | i = i' - 1 \wedge (w = w' \vee |w - w'| = 2^{d-i-1})\right\}.$$

The butterfly has $(d+1)2^d$ vertices and $d2^{d+1}$ edges. Figure 3.10 shows the butterfly of dimension three. The vertices of the butterfly can be divided into "rows" w and levels i. It is often very helpful to look at the rows w as their binary number, as we have done in the figure. Furthermore, the edges can be divided into levels which is also shown in the figure.

It is known that the vertices in one level are symmetric. Vertices of level i and $d-i$ are also symmetric. Furthermore, edges inside one level are symmetric and edges of level i and $d-i+1$ are also symmetric.

The butterfly has a simple bisection with bisection-width 2^d: vertices of rows $0,\dots,2^{d-1}-1$ form the one partition. A better bisection had not been discovered for a very long time until Bornstein et al. showed in [BLM$^+$98] that the bisection width of the $BF(d)$ is $2(\sqrt{2}-1)2^d + o(2^d) \approx 0.82 \cdot 2^d$. In the following we present the MC-bounds for the bisection-width of the butterfly.

In order to determine the MC-bounds, firstly we show that we can restrict the flows to shortest paths, even though the butterfly is not edge-symmetric:

Lemma 3.14
There are optimal symmetrical MC-solutions on the butterfly graph which only use shortest paths.

Proof:
Firstly, from the symmetry of the edges inside one level we can conclude that we only have to count the number of uses of edges of the different levels in order to calculate the congestion. Secondly, from the construction of the butterfly it becomes clear that every path from a vertex (w,i) to a destination (w',i') has to use edges of levels j, if w and w' differ in the j'th bit. The minimal usage of the different levels of edges of any path follows directly from this fact. So, any path generates at least as much flow on each level as shortest paths do. As a result of this we can restrict our considerations to shortest paths. ■

Next, we give a lemma which states how much flow is generated on the different levels of edges:

Lemma 3.15
Let $Load_d(k,l)$ be the sum of the flows on edges of level k, if all vertices of level l send a commodity of size one to every other vertex using only shortest paths. Then

$$Load_d(k,l) = 2^{2d} \cdot \begin{cases} X_{k,d} & \text{if } k > l \\ Y_{k,d} & \text{else} \end{cases}$$

with $X_{k,d} := d+k+1-k\cdot 2^{k-d}$ and $Y_{k,d} = 2d-k+2-(d-k+1)2^{-k+1}$ holds.

Proof:
Firstly, from the symmetry of the butterfly it follows that $Load_d(k,l) = Load_d(d-k+1, d-l)$ holds. So, we only have to look at the case $k > l$, the else-case will then follow when we use this symmetry. Again, due to the symmetry of the vertices inside one level, we look at one specific sender-vertex. Firstly, there are the destinations on levels i with $i \geq k$: the commodities have to use edges of level k exactly once for these destinations. They are $(d-k+1)2^d$ destinations. Secondly, we look at destinations on level i with $i < k$: Flows to vertices on these levels with rows which do not differ in any bit j with $j \geq k$ do not have to use level-k edges at all. These are 2^{k-1} vertices on each level. The flows for all other destinations have to use level-k edges in two instances: going up and going down. These are $2^d - 2^{k-1}$ vertices. Altogether we have

$$\begin{aligned} Load_d(k,l) &= 2^d\left((d-k+1)2^d + 2k(2^d - 2^{k-1})\right) \\ &= 2^{2d}\left(d+k+1-k\cdot 2^{k-d}\right) = 2^{2d}X_{k,d} \end{aligned}$$

which proves the lemma. ■

1-1-MC-bound

Using the instrument of the above lemma the 1-1-MC-bound follows:

Theorem 3.15
The 1-1-MC-bound for the bisection width of the butterfly is

$$\textit{1-1-MC-bound} = \frac{1}{2}2^d + O(\frac{1}{d}2^d).$$

Proof:
The Cut-Flow of any 1-1-MC-solution for the bisection of the butterfly is $\frac{1}{2}(d+1)^2 2^{2d}$. The congestion $c(k)$ of the edges on level k is

$$\begin{aligned} c(k) &= \frac{1}{2^{d+1}} \sum_{l=0}^{d} Load_d(k,l) \\ &= 2^{d-1}(kX_{k,d} + (d-k+1)Y_{k,d}) \end{aligned}$$

So, the 1-1-MC-bound Bd is

$$\begin{aligned} Bd &= \min_{k\in\{1,\ldots,d\}} \frac{(d+1)^2 2^{2d}}{2c(k)} \\ &= \min_{k\in\{1,\ldots,d\}} \frac{(d+1)^2 2^d}{kX_{k,d} + (d-k+1)Y_{k,d}} \end{aligned}$$

Exploring the asymptotic behavior of the formula provides us with the result of the theorem. ■

VarMC-Bound and MVarMC-Bound

The determination of the VarMC-bound is more complex than the determination of the 1-1-MC-bound, since we also have to find the optimal source-strengths. Firstly, we present a lemma which will provide us with an argument for choosing the right source-strengths:

Lemma 3.16
Let be $X_{k,d}$ and $Y_{k,d}$ according to Lemma 3.15 with $1 \le k \le d$. Then

$$k \ge \frac{d+1}{2} \Leftrightarrow X_{k,d} \ge Y_{k,d}$$

holds.

Proof:
We substitute $k = \frac{d+1}{2} + x$ with $\frac{-d-1}{2} \leq x \leq \frac{d-1}{2}$ inside $X_{k,d}$ and $Y_{k,d}$ and we get:

$$\begin{aligned} X_{k,d}(x) &= \frac{1}{2}\left(3d+3+2x-(d+1+2x)2^{x+\frac{1-d}{2}}\right) \\ Y_{k,d}(x) &= \frac{1}{2}\left(3d+3-2x-(d+1-2x)2^{-x+\frac{1-d}{2}}\right) \end{aligned}$$

which shows that $X_{k,d}(x) = Y_{k,d}(-x)$ holds. Therefore, the lemma is proved if $x \geq 0 \Rightarrow X_{k,d}(x) \geq Y_{k,d}(x)$ holds. With $0 \leq x \leq \frac{d-1}{2}$ we have

$$\begin{aligned} & X_{k,d}(x) &\geq& \frac{1}{2}(d+3+2x) \\ \wedge\ & Y_{k,d}(x) &\leq& \frac{1}{2}\left(3d+3-2x-2^{d+2}\right) \end{aligned}$$

so

$$\begin{aligned} X_{k,d}(x) &\geq Y_{k,d}(x) \\ \Leftarrow \quad \frac{1}{2}(d+3+2x) &\geq \frac{1}{2}\left(3d+3-2x-2^{d+2}\right) \\ \Leftrightarrow \quad 2x+2^{d+1} &\geq d \end{aligned}$$

which is true. ■

Now, the following theorem gives the bound:

Theorem 3.16
The VarMC-bound for the bisection width of the butterfly is

$$\textit{VarMC-bound} = \frac{2}{3}2^d + O\left(\frac{1}{d}2^d\right)$$

and can be accomplished by an instance where only vertices of level 0 *and d send commodities to all other vertices.*

Proof:
Due to the symmetry of the vertices, we only have to determine source-strengths for every level of vertices. Let $s(l)$ be the source-strength of the vertices of level l. Then we have congestion $c(k)$ on the vertices of level k :

$$c(k) = \frac{1}{2^{d+1}} \sum_{l=0}^{d} s(l) \cdot Load_d(k,l)$$

Furthermore, an analysis of $Load_d(k,l)$ and Lemma 3.16 shows that $\forall l \leq \frac{d}{2} : \forall k : Load_d(k,l) \geq Load_d(k,0)$ and $\forall l > \frac{d}{2} : \forall k : Load_d(k,l) \geq Load_d(k,d)$ hold. So, if

$\exists l\ 0 < l < d : s(l) \neq 0$ then we could change the corresponding VarMC-solution by decreasing $s(l)$ and increasing $s(0)$ or $s(d)$ respectively accordingly. This modified VarMC-solution has the same Cut-Flow and less *Load* on every edge-level k. Finally, due to the symmetry of level 0 and level d, we have $s(0) = s(d) = 1$ and $\forall l\ 0 < l < d : s(l) = 0$. This gives a VarMC-bound of

$$\begin{aligned} \text{VarMC-bound} &= (d+1)2^{d+1} \min_k \frac{1}{X_{k,d} + Y_{k,d}} \\ &= \frac{2}{3} 2^d + O(\frac{1}{d} 2^d) \end{aligned}$$

where the minimum is reached when $k = \frac{d}{2}$. ■

Therefore, the asymptotic VarMC-bound is a factor of $\frac{1}{3}$ better than the asymptotic 1-1-MC-bound. Finally, we present the results of the more general MVarMC-bound:

Theorem 3.17
The MVarMC-bound for the bisection width of the butterfly is identical to the VarMC-bound.

Proof:
We prove the theorem by looking at any MVarMC-solution and we show that it can be adapted to an VarMC-solution while not worsening the bound. Let s be the source-strengths of the MVarMC-instance. Any MVarMC-instance is a VarMC-instance if and only if $\forall v, w : s(v,w) = \bar{s_v}$ (with $\bar{s_v}$ according to Theorem 2.2). We will show that if $\exists v, w : s(v,w) < \bar{s_v}$ we can increase $s(v,w)$ to $\bar{s_v}$ while not worsening the bound. Since we can restrict our calculations to symmetrical solutions, we do this for all vertices in one level at the same time. Let $a = \bar{s_v} - s(v,w)$, Bd (Bd') be the bound of the original (adapted) solution, CF (CF') be the Cut-Flow of the original (adapted) solution and $c(k)$ ($c'(k)$) be the congestion of level-k edges of the original (adapted) solution. Then we have

$$\begin{aligned} Bd' &= \min_k \frac{CF'}{c'(k)} \\ &= \min_k \frac{CF + 2^d \cdot a}{c(k) + \frac{1}{2^{d+1}} 2^d \cdot a \cdot x_{k,l}} \end{aligned}$$

where $x_{k.l}$ is the usage of level-k edges in the given case. From the proof of Lemma 3.15 we know that $x_{k,l} \in \{0,1,2\}$. We already know from the minimization inside the formula above that $Bd' \geq Bd \ \Leftarrow \forall k : \frac{CF'}{c'(k)} \geq \frac{CF}{c(k)}$. So, we have to show that

$$\forall k : \quad \frac{CF'}{c'(k)} = \frac{CF + 2^d a}{c(k) + \frac{1}{2^{d+1}} 2^d a x_{k,l}} \geq \frac{CF}{c(k)}$$

holds. Generally, $\frac{a+b}{c+d} \geq \frac{a}{c} \Leftrightarrow \frac{b}{d} \geq \frac{a}{c}$ holds. So we have

$$\begin{array}{rrcl} & Bd' & \geq & Bd \\ \Leftrightarrow & \forall k: \frac{2^d a}{\frac{1}{2^{d+1}} 2^d a x_{k,l}} & \geq & \frac{CF}{c(k)} \\ \Leftrightarrow & \forall k: \frac{2^{d+1}}{x_{k,l}} & \geq & \frac{CF}{c(k)} \end{array}$$

and since $x_{k,l} \in \{0,1,2\}$ we have $\forall k: \frac{2^{d+1}}{x_{k,l}} \geq 2^d$. Since $\frac{CF}{c(k)}$ is a valid bound and there is a bisection with cut-size 2^d we have $\frac{CF}{c(k)} \leq 2^d$. So, $\forall k: \frac{2^{d+1}}{x_{k,l}} \geq \frac{CF}{c(k)}$ is true and $Bd' \geq Bd$ follows. ■

3.2.5 Bisection of the Beneš Network

In this section we will present analyses of the Beneš network. The Beneš network is very similar to the butterfly network, in fact the Beneš network consists of back-to-back butterflies, as shown in Figure 3.11. The Beneš network is famous for the fact that it is a rearrangeable network, i.e. for any mapping of the level-0 to level-$2d$ vertices (input and output vertices), edge-disjoint paths which connect the pairs can be constructed.

Definition 3.8
The d-dimensional Beneš $BE(d) = (V,E)$ is an undirected graph with vertices

$$V = \left\{ (w,i) | i \in \{0,\ldots,2d\} \wedge w \in \{0,\ldots,2^d-1\} \right\}$$

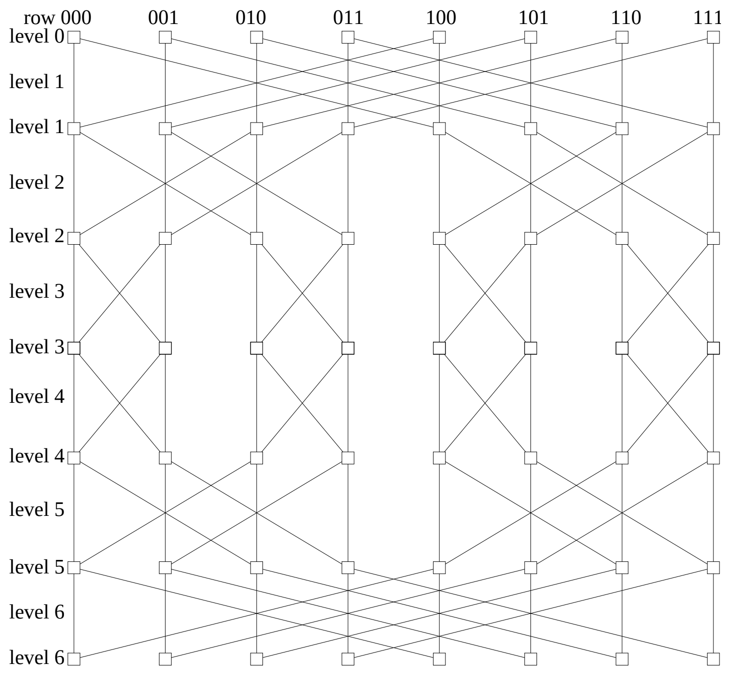
and edges

$$E = \left\{ \{(w,i),(w',i')\} \, | i = i'-1 \wedge \left(w = w' \vee |w-w'| = \left\{ \begin{array}{ll} 2^{d-1-i} & \mathit{if}\, i < d \\ 2^{i-d} & \mathit{else} \end{array} \right. \right) \right\}.$$

The Beneš network has $(2d+1)2^d$ vertices and $d2^{d+2}$ edges. The vertices of the Beneš network can be divided into "rows" w and levels i. It is often very helpful to look at the rows w as their binary number, which has been done in the figure. Furthermore, the edges can be divided into levels which is also shown in the figure. The first and last $d+1$ levels form a d-dimensional butterfly.

The Beneš network has similar symmetries to the butterfly: Vertices and edges inside one level are symmetric. Vertices of level i and $2d-i$ are symmetric and edges of

Figure 3.11: Beneš network of dimension 3



level i and $2d-i+1$ are symmetric. As with the butterfly the Beneš network has a simple bisection with cut-size 2^{d+1}. In the following we present the MC-bounds for the bisection-width of the butterfly. The analysis are quite similar to the analysis of the butterfly.

Lemma 3.17
Let $Load_d(k,l)$ be the sum of the flows on edges of level k and $2d-k+1$ if all vertices of level l and $2d-l$ send a commodity of size one to every other vertex. Then with $1 \le k \le d$ and $0 \le l \le d$

$$Load_d(k,l) = \begin{cases} 2^{2d+1} & (2X_{k,d} - 1 + k2^{k-d}) & \text{if } k > l \\ 2^{2d+1} & (2Y_{k,d} - 2 - 2^{1-k}) & \text{if } k \le l \wedge l < d \\ 2^{2d} & (2Y_{k,d} - 2 - 2^{1-k}) & \text{if } l = d \end{cases}$$

with $X_{k,d}$ and $Y_{k,d}$ according to Lemma 3.15 holds.

Proof:
The proof corresponds to the proof of Lemma 3.15. Since the vertices inside these two levels are symmetrical, we only look at vertex $(0,l)$.

We start by looking at the case of $k > l$: Firstly, all destinations in levels i with $i \in \{k,\dots,2d-k\}$ have to use level-l edges exactly once. These are altogether $(2d-2k+$

$1)2^d$ destinations. Secondly, we look at destinations in levels i with $i \in \{0,\dots,k-1\}$, these are k levels altogether. Vertices inside these levels on rows which do not differ in bits $k,\dots,d$ do not use level-k edges at all, they are $k2^{k-1}$ vertices. The other paths to vertices inside these levels have to use level-k edges exactly twice, so they generate a usage of $2k(2^d - 2^{k-1})$. Thirdly, paths to destinations in levels i with $2d-k+1 \le i \le 2d$ have to use level-k edges exactly twice, so they generate a usage of $2k2^d$. So, altogether we have flows of

$$\begin{aligned} & 2^{d+1}\left((2d-2k+1)2^d + 2k(2^d-2^{k-1}) + 2k2^d\right) \\ = \;& 2^{2d+1}\left(2d+2k+1-k2^{k-d}\right) \\ = \;& 2^{2d+1}(2X_{k,d}-1+k2^{k-d}). \end{aligned}$$

Next, we look at the case of $k \le l$: Firstly, paths to destinations on levels i with $i \in \{0,\dots,k-1,2d-k+1,\dots,2d\}$ use level-k edges exactly once. So they generate a flow of $2k2^d$. Secondly, we look at destinations on levels i with $i \in \{k,\dots,2d-k\}$, they are $2d-2k+1$ levels. Paths to destinations on rows which do not differ in bits $0,\dots,k-1$ do not have to use level-k edges, they are 2^{d-k}vertices per level. Paths to the other vertices on these levels use level-k edges twice, so they generate a flow of $2(2d-2k+1)(2^d-2^{d-k})$. Altogether we have

$$\begin{aligned} & 2k2^d + 2(2d-2k+1)(2^d-2^{d-k}) \\ = \;& 2^d\left(2d-2k+2-(2d-2k+1)2^{1-k}\right) \\ = \;& 2^d\left(2Y_{k,d}-2+2^{1-k}\right) \end{aligned}$$

flows per sender. If $l < d$, there are 2^{d+1} senders and if $l = d$ there are 2^d senders. ∎

The high similarity of the loads of the Beneš network to the load of the butterfly suggests that the bounds and their proofs are also very similar. In fact, they are very similar, so we can leave out some details in the following. We begin with the 1-1-MC-bound:

Theorem 3.18
The 1-1-MC-bound for the bisection width of the Beneš network is

$$\textit{1-1-MC-bound} = \frac{1}{2}2^{d+1} + O(\frac{1}{d}2^{d+1}).$$

Proof:
The Cut-Flow of an 1-1-MC-solution is $(2d+1)^2 2^{2d-1}$. Let $c(k)$ be the congestion of

level-k edges, then we have

$$\begin{aligned} c(k) &= \frac{1}{2^{d+1}} \sum_{l=0}^{d} Load_d(k,l) \\ &= 2^d \left(2k(2X_{k,d} - 1 + k2^{k-d}) + (2d - 2k + 1)(2Y_{k,d} - 2 + 2^{1-k}) \right). \end{aligned}$$

So, for the 1-1-MC-bound Bd

$$\begin{aligned} Bd &= \min_k \frac{(2d+1)^2 2^{2d-1}}{c(k)} \\ &= \min_k \frac{(2d+1)^2 2^{d+1}}{k(X_{k,d} - \frac{1}{2} + k2^{k-d-1}) + (d - k + \frac{1}{2})(Y_{k,d} - 1 + 2^{-k})} \\ &= \frac{1}{2} 2^{d+1} + O(\frac{1}{d} 2^{d+1}) \end{aligned}$$

holds. ■

And for the VarMC-bound we have:

Theorem 3.19
The VarMC-bound for the bisection width of the Beneš network is

$$\textit{VarMC-bound} = \frac{2}{3} 2^{d+1} + O(\frac{1}{d} 2^{d+1}).$$

Proof:
The proof is similar to the proof of the VarMC-bound for the butterfly. Firstly, due to the symmetry of the vertices we can restrict the source-strengths for every vertex to source-strengths $s(l)$ for every level $0 \leq l \leq d$. Secondly, since levels in the middle are worse than levels at the end we have $\forall 0 < l < d : s(l) = 0$. Due to the scalability of solutions, we can set $s(0) = 1$ and the optimal $s(d)$ has to be determined. A careful examination of the resulting formulas show that $s(d) = 2$ gives the best bound. In this case, the edges of level $\frac{d}{2}$ have the maximal congestion and the asymptotic behavior of

$$\text{VarMC-bound} = \frac{2}{3} 2^{d+1} + O(\frac{1}{d} 2^{d+1})$$

follows. ■

And finally, for the MVarMC-bound we have:

Theorem 3.20
The MVarMC-bound for the bisection width of the Beneš network is identical to the VarMC-bound.

Proof:
The proof of this theorem can be done similarly to the proof of Theorem 3.17. ■

3.2.6 The k-partitioning of the Hypercube $Q(d)$

The hypercube is one of the most versatile and efficient networks which has yet been discovered for the purpose of parallel computation. It can efficiently simulate any other network of the same size. The hypercube has a low diameter ($\log n$) and a high bisection width ($\frac{n}{2}$).

Definition 3.9
The hypercube $Q(d) = (V,E)$ of dimension d is an undirected graph with

$$V = \{0,1\}^d$$

and

$$E = \{\{u,v\} : \ u,v \in V \textit{ differs in exactly one bit}\}.$$

The $Q(d)$ has $n = 2^d$ vertices, degree d and therefore $d2^{d-1}$ edges. It is vertex- and edge-symmetric. Figure 3.12 illustrates the hypercubes of dimensions $0,\ldots,4$ and 7.

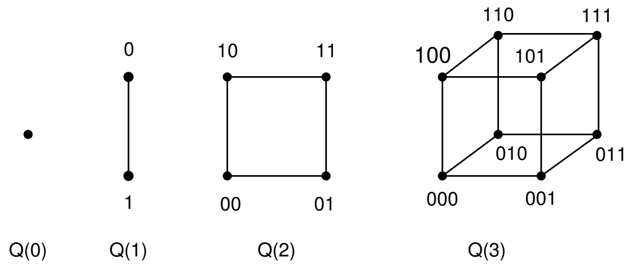
The cut-size of the hypercube is $\frac{n}{2} \cdot \log k$ if $k|n$. Since the optimal partitions fulfill the property that every vertex has the same number of adjacent edges which cross the cut, the eigenvalue-based bounds are exact. We will later see that the Leighton-bound (i.e. 1-1-MC-bound) has an error of about $\log k$. Therefore we see it as a challenge to achieve a better result using the MVarMC-bound. In the following firstly we will present an exact formula for the MVarMC-bound. Since a lot of considerations which we have used in the development of this formula are identical to the ones which we have already presented for other graphs, we will not give many details of this development. Secondly, we will give an asymptotic analysis of this formula.

Lemma 3.18
The MVarMC-bound B_d for the k-partitioning of the $Q(d)$ is

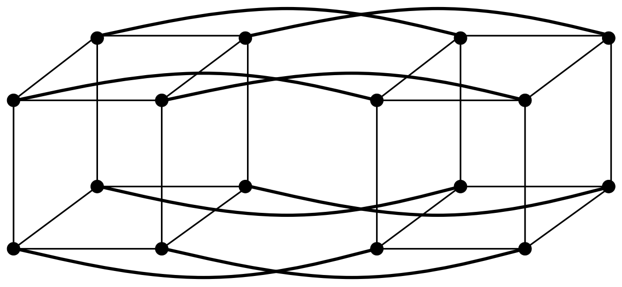
$$B_d = 2^{d-1} \max_{x \in \{1,\ldots,d\}} \frac{\sum_{i=0}^{x} \binom{d}{i} - \frac{2^d}{k}}{\sum_{i=0}^{x-1} \binom{d-1}{i}}$$

Proof:
Firstly, since the hypercube is edge-symmetric, the congestion can be computed by adding the distances of the vertices. Therefore, with the help of Theorem 3.5 we know

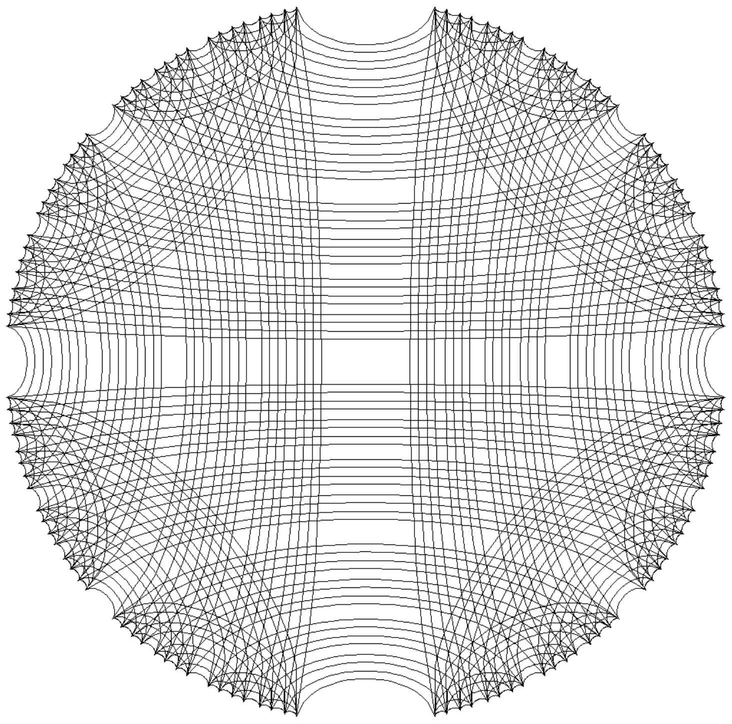
Figure 3.12: Hypercubes $Q(d)$ of dimensions $0,\ldots,4$ and 7



(a) $d = 0,\ldots,3$



(b) $d = 4$



(c) $d = 7$

that we can assume $s(v,w) = \{0,1\}$ and that $s(v,w)$ is monotonically decreasing with the distance of v and w. So, every vertex v sends commodities of size one to every vertex w with $dist(v,w) \leq x$ and the determination of the optimal x remains. There are $\binom{d}{x}$ vertices with distance x from any specific vertex. This reveals a Cut-Flow CF of

$$CF = 2^d \left(\sum_{i=0}^{x} \binom{d}{i} - \frac{2^d}{k} \right)$$

and a congestion C of

$$C = \frac{2^d}{d2^{d-1}} \sum_{i=0}^{x} i \cdot \binom{d}{i} = 2 \sum_{i=0}^{x-1} \binom{d-1}{i}$$

and with $B_d = \frac{CF}{C}$ the lemma is proved. ■

Unfortunately, the expression $\frac{\sum_{i=0}^{x}\binom{d}{i} - \frac{2^d}{k}}{\sum_{i=0}^{x-1}\binom{d-1}{i}}$ cannot be given in a closed form. Therefore, the analysis is difficult. However, the asymptotic behavior can be almost exactly analyzed. Using the following theorem, we present an upper bound on the MVarMC-bound:

Theorem 3.21
Let B_d be the MVarMC-bound of the $Q(d)$, then

$$\frac{B_d}{2^{d-1}} \leq 1 + \frac{2}{1-\alpha} + O(\frac{1}{d})$$

holds for the k-partitioning with $k = 2^{\alpha d}$ with $\alpha \in (0,1)$, $k|2^d$ and $d \to \infty$.

Proof:
In this proof we concentrate on the term $B_d(x)$ with

$$B_d(x) := \frac{\sum_{i=0}^{x}\binom{d}{i} - \frac{2^d}{k}}{\sum_{i=0}^{x-1}\binom{d-1}{i}}.$$

We can transform and estimate $B_d(x)$:

$$\begin{aligned} B_d(x) &= 2 + \frac{\binom{d-1}{x} - 2^{(1-\alpha)d}}{\sum_{i=0}^{x-1}\binom{d-1}{i}} \\ &\leq 2 + \frac{\binom{d-1}{x} - 2^{(1-\alpha)d}}{\binom{d-1}{x-1}} \\ &= 1 + \frac{d}{x} - \frac{2^{(1-\alpha)d}}{\binom{d-1}{x-1}} \end{aligned}$$

Obviously, $\lim 2^{(1-\alpha)d} < \lim \binom{d-1}{x-1}$ must hold for the x which maximizes B_d. If we use $x = \beta(d-1)+1$ we achieve

$$\begin{aligned} \lim \binom{d-1}{x-1} &= \lim \binom{d-1}{\beta(d-1)} \\ &= \beta^{-1}\left(2\pi(d-1)(\beta^{-1}-1)\right)^{-\frac{1}{2}} \cdot C(\beta^{-1})^{d-1} \end{aligned}$$

with $C(b) := b^{\frac{1}{b}} \left(\frac{b}{b-1}\right)^{\frac{b-1}{b}}$. If we use this, we have

$$\begin{aligned} & \lim 2^{(1-\alpha)d} < \lim \binom{d-1}{x-1} \\ \Leftrightarrow \quad & 2^{1-\alpha} \le C(\beta^{-1}) \\ \Rightarrow \quad & \beta \ge \frac{1-\alpha}{2}. \end{aligned}$$

So, we have $x \ge \frac{1-\alpha}{2}(d-1)+1$ and

$$\begin{aligned} B_d &= 2^{d-1} \max_x B_d(x) \\ &\le 2^{d-1}\left(1 + \frac{d}{x} - \frac{2^{(1-\alpha)d}}{\binom{d-1}{x-1}}\right) \\ &\le 2^{d-1}\left(1 + \frac{2}{1-\alpha} + O(\frac{1}{d})\right) \end{aligned}$$

and the theorem is proved. ■

Similar estimates provide us with a lower bound on the MVarMC-bound:

Theorem 3.22
Let B_d be the MVarMC-bound of the $Q(d)$, then

$$\frac{B_d}{2^{d-1}} \ge \frac{2}{1-\alpha} + O(\frac{1}{d})$$

holds for the k-partitioning with $k = 2^{\alpha d}$ with $\alpha \in (0,1)$, $k | 2^d$ and $d \to \infty$.

Proof:
We again use

$$B_d(x) := \frac{\sum_{i=0}^{x} \binom{d}{i} - \frac{2^d}{k}}{\sum_{i=0}^{x-1} \binom{d-1}{i}}.$$

We can transform and estimate $B_d(x)$ using $x = \beta(d-1)+1$:

$$\begin{aligned} B_d(x) &= 2 + \frac{\binom{d-1}{x} - 2^{(1-\alpha)d}}{\sum_{i=0}^{x-1}\binom{d-1}{i}} \\ &\geq 2 + \frac{\binom{d-1}{x} - 2^{(1-\alpha)d}}{\frac{1-\beta}{1-2\beta}\binom{d-1}{\beta(d-1)}} \\ &= 2 + \frac{1-2\beta}{\frac{1}{d-1}+\beta} - \frac{2^{(1-\alpha)d}}{\frac{1-\beta}{1-2\beta}\binom{d-1}{\beta(d-1)}} \end{aligned}$$

Now, we choose a β such that $\lim 2^{(1-\alpha)} < \lim \binom{d-1}{\beta(d-1)}^{-d}$ holds. We have seen in the previous proof, that this is fulfilled by using $\beta = \frac{1-\alpha}{2}$. With this β, we achieve

$$\begin{aligned} B_d &\geq 2^{d-1} B_d(\beta(d-1)+1) \\ &= 2^{d-1}\left(\frac{2}{1-\alpha} + O(\frac{1}{d})\right) \end{aligned}$$

and the theorem is proved. ■

Altogether, we now have asymptotic lower and upper bounds on the MVarMC-bound which are almost identical:

Corollary 3.7
Let B_d be the MVarMC-bound of the $Q(d)$, then

$$\frac{2}{1-\alpha} + O(\frac{1}{d}) \leq \frac{B_d}{2^{d-1}} \leq 1 + \frac{2}{1-\alpha} + O(\frac{1}{d})$$

holds for the k-partitioning with $k = 2^{\alpha d}$ with $\alpha \in (0,1)$, $k|2^d$ and $d \to \infty$.

Figure 3.13 illustrates the asymptotic error of the MVarMC-bound depending on α while Figure 3.14 illustrates the exact error with $d = 50$. Both figures show that the maximal error is obtained with $\alpha \approx 0.5$, there we have a maximal asymptotic error of $0.125d$ while the example with $d = 50$ gives a maximal error of about $0.085d$. Furthermore, both figures also show that when $\alpha \to 0$ or $\alpha \to 1$, the error decreases.

This can be substantiated by an analysis of the formula of Lemma 3.18. If we use $x = d$, the MVarMC-bound is equal to the 1-1-MC-bound. Then we have an error of $\frac{\log k}{2(1-\frac{1}{k})}$, so if $k = 2$ the bound is equal to the cut-size and if k is constant, the error is constant. On the other hand if we use $x = 1$, the bound is equal to the cut-size if $k = 2^d$ or $k = 2^{d-1}$ and it is asymptotically exact if $\frac{k}{2^d}$ is constant.

Figure 3.13: Asymptotic error of the MVarMC-bound for the k-partitioning of the hypercube

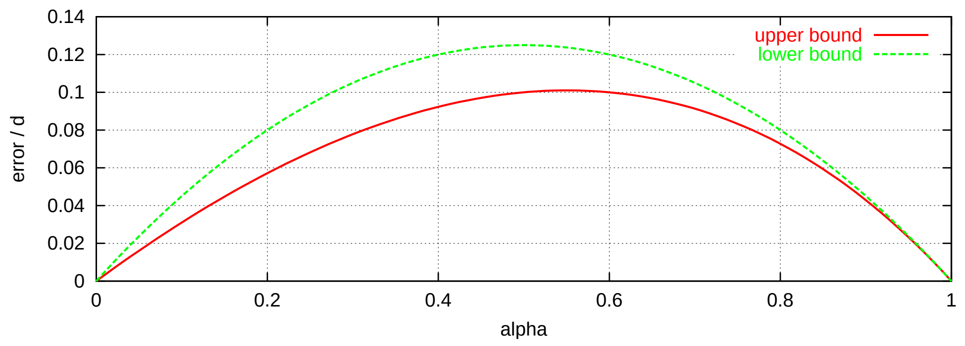
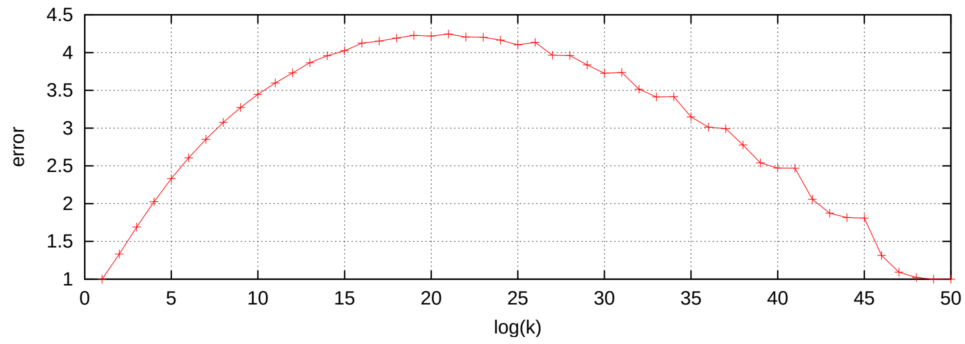


Figure 3.14: Exact error of the MVarMC-bound for the k-partitioning of the hypercube with $d = 50$



3.2.7 Summary

Table 3.3 summarizes the results which we have proved in the preceding sections. It can be seen that we have obtained an enhancement of all bounds on the bisection width by using the VarMC-bound instead of using the 1-1-MC-bound. If k-partitioning problems have been looked up, we have obtained an drastic enhancement by using the MVarMC-bound instead of the 1-1-MC-bound or VarMC-bound.

We have also included the errors of the classical eigenvalue-bound, the DH-bound. The second smallest eigenvalue λ_2 of the $K_{a,b}$ graph is $\lambda_2 = b$ if $b \leq a$. Using this we obtain the results. The second smallest eigenvalue of the butterfly network is $\lambda_2 = 4 - 4\cos\frac{\pi}{2d+1} \longrightarrow \frac{\pi^2}{2d(d+1)}$, this provides us with an asymptotic bound of $\frac{\pi^2}{8d}2^d$. The calculation of the DH-bound for the k-partitioning of the hypercube $Q(d)$ is complicated, we have to use the fact that the $Q(d)$ has $\binom{d}{i}$ eigenvalues with a value of $2i$.

Table 3.3: Summarization of the analyzed asymptotic errors

problem	1-1-MC	VarMC	MVarMC	DH
$K_{a,b}$, $b = o(a)$, $k = 2$	2	1	1	2
$K_{a,b}$, $b = \alpha a$, $k = 2$	$2 - \frac{2\alpha}{(\alpha+1)^2}$	$1 + \frac{\alpha}{\alpha+1}$	$1 + \frac{\alpha}{\alpha+1}$	$1 + \frac{1-\alpha}{1+\alpha}$
butterfly, $k = 2$	≈ 1.64	≈ 1.23	≈ 1.23	$\approx 0.66 \cdot d$
Beneš, $k = 2$	≤ 2	≤ 1.5	≤ 1.5	n.a.
$a \times a$-torus	$0.5\sqrt{k}$	$0.5\sqrt{k}$	≈ 1.22	(≈ 14.36 if $k = 6$)
$a \times a$-grid	$0.5\sqrt{k}$	$0.5\sqrt{k}$	≈ 1.22	n.a.
hypercube $Q(d)$, $k = 2^{\alpha d}$	$\frac{1}{2}\alpha d$	$\frac{1}{2}\alpha d$	$\frac{1}{2}\alpha(1-\alpha)d$	constant

Therefore, a sum like $\sum_{i=1}^{x} i\binom{d}{i}$ has to be analyzed. If we do this we obtain an constant asymptotic error if $k = 2^{\alpha d}$ is assumed. Finally, the DH-bound for the $a \times a$-torus gives very bad result. The quoted term is based on an improvement of the DH-bound which can be applied for graphs with a special structure, see [ELM01]. You cannot give a closed form of this improved bound for arbitrary k's, therefore we have quoted the bound for $k = 6$.

The comparison of the presented MC-bounds with the eigenvalue-based bounds show that the new bounds are very competitive. In the presented example the DH-bound is only superior when the hypercube is regarded. On the other hand it delivers clearly worse bounds regarding the butterfly network or the $a \times a$-torus.

3.3 Upper Bounds on the Lower Bounds

In this section we will present upper bounds on the MC-bounds. These upper bounds are based on the edge-expansion $e(G,k)$ of a graph G. The edge-expansion is a known attribute of graphs in the graph-theory, see e.g. [BLM$^+$98, Nog93]. For ease of notation in this section, we do not use vertex- and edge-weights.

Definition 3.10
The edge-expansion $e(G,l)$ *of an unweighted graph* $G = (V,E)$ *is defined as*

$$\forall 1 \leq l \leq \frac{n}{2}: \quad e(G,l) := \min_{U \subset V, |U| = l} \frac{c(U, V \setminus U)}{l}$$

with $c(U, V \setminus U) := |\{\{v,w\} \in E : v \in U \wedge w \notin U\}|$.

By using this edge-expansion, the following theorem provides us with an upper bound on the 1-1-MC-bound:

Theorem 3.23
Let $GP = (G, k, M)$ *be a graph partitioning problem. Then*

$$\forall 1 \leq l \leq \frac{n}{2}: \quad \textit{1-1-MC-bound} \leq \frac{n(n-M)}{2(n-l)} e(G,l)$$

holds.

Proof:
The Cut-Flow of any 1-1-MC-instance is $n(n-M)$. We get a lower bound on the congestion if we look at the edges which correspond to $e(G,l)$: The number of commodities which have to cross this cut are $2l(n-l)$, there are $l \cdot e(G,l)$ edges crossing this cut, so these edges have a congestion of at least $\frac{2l(n-l)}{l \cdot e(G,l)} = \frac{2(n-l)}{e(G,l)}$. If we put the Cut-Flow and congestion together, we get a bound B of

$$B = \frac{CutFlow}{C} \leq \frac{n}{2(n-l)} \cdot (n-M) \cdot e(G,l)$$

and the theorem is proved. ∎

In contrast to the 1-1-MC-bound, the VarMC-bound can react more flexibly on a possible bottle-neck. The following theorem gives an upper bound on the VarMC-bound:

Theorem 3.24
Let $GP = (G, k, M)$ *be a graph partitioning problem. Then*

$$\forall 1 \leq l \leq \frac{n}{2}: \quad \textit{VarMC-bound} \leq (n-M) \cdot e(G,l)$$

holds.

Proof:
We concentrate on the congestion of the edges which correspond to the edge-expansion $e(G,l)$. In order to minimize this congestion, it is optimal if the vertices of $V \setminus U$ only send commodities. If we do so, we can achieve a Cut-Flow of $(n-M)(n-l)$ and a congestion of at least $\frac{(n-l)l}{l \cdot e(G,l)} = \frac{n-l}{e(G,l)}$. If we put the Cut-Flow and congestion together, we get a bound B of

$$B = \frac{CutFlow}{C} \leq \frac{(n-l)(n-M)}{n-l} e(G,l) = (n-M) \cdot e(G,l)$$

Table 3.4: Upper bounds on the MC-bounds with reference to the bisection width

MC-bound	upper bound
1-1-MC	$\frac{n^2}{4(n-l)} \cdot e(G,l)$
VarMC	$\frac{n}{2} \cdot e(G,l)$
MVarMC	$\frac{1}{2}(n-l)(\frac{n}{2}-l)+l\cdot e(G,l)$

and the theorem is proved. ■

If we refer to the MVarMC-bound, we realize that the considerations which we have used for the 1-1-MC-bound and VarMC-bound do not give us any upper bound on the MVarMC-bound. This is due to the fact that MVarMC-instances are not forced to send any commodities across the cut of the edge-expansion. However, if we restrict our considerations to the graph bisection problem, we can provide an upper bound on the MVarMC-bound:

Theorem 3.25
Let bisGP=(G) be a graph bisection problem. Then

$$\forall 1 \le l \le \frac{n}{2}: \quad \textit{MVarMC-bound} \le \frac{1}{2}(n-l)(\frac{n}{2}-l)+l\cdot e(G,l)$$

holds.

Proof:
Since MC-instances are scalable, we restrict our considerations on MVarMC-instances with a congestion of one. This means that at most $l\cdot e(G,l)$ commodities can cross the cut of the edge-expansion. If we assume the best case that $V\setminus U$ is a clique-graph, the vertices inside this clique can send commodities of size $\frac{1}{2}$ to each other. So a Cut-Flow CF of

$$\begin{aligned} CF &= l\cdot e(G,l)+\frac{n}{2}-l+\frac{1}{2}(n-l+1)(\frac{n}{2}-l-1) \\ &\le \frac{1}{2}(n-l)(\frac{n}{2}-l)+l\cdot e(G,l) \end{aligned}$$

is possible. Since the congestion is one, the theorem follows. ■

Figure 3.15: Illustration of the dependence of the upper bounds on l with $n = 100$, $e(G,l) = 2$ and with reference to the bisection problem

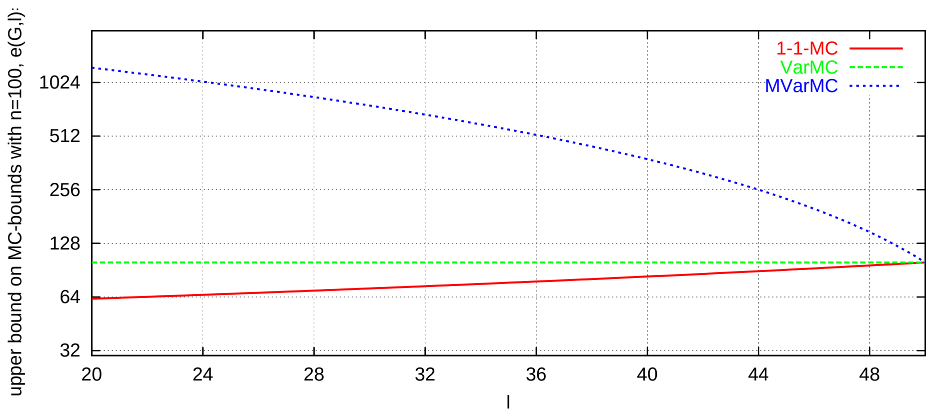


Table 3.4 summarizes the upper bounds and Figure 3.15 illustrates the upper bounds with $n = 100$, $e(G,l) = 2$ and $M = 50$. The figure clarifies once more the superiority of the VarMC-bound when compared to the 1-1-MC-bound and the superiority of the MVarMC-bound when compared to the VarMC-bound and 1-1-MC-bound:

If $l = 50 = \frac{n}{2}$, the edge-expansion corresponds to a bisection width, so all upper bounds which are based on the edge-expansion have to be identical and correspond to the exact value. Then, when l is decreasing the differences between the abilities of the MC-bounds to react on bottle-necks inside graphs are illustrated in the figure. However, we have to annotate that while the upper bounds on the 1-1-MC-bound and VarMC-bound are realistic, the upper bounds on the MVarMC-bound are very optimistic and only hold, if the graph is very dense except for the one cut which corresponds to the edge-expansion.

Chapter 4

Computation of the Lower Bounds

Until now, we have not commented on how we can efficiently compute the MC-bounds. We have already presented computational results mainly in Section 2.3 without giving details of how the bounds are computed. So in this chapter we will present different methods which can be used for the computation of the MC-bounds.

The standard-method is using linear programming. So in the Section 4.1 we will present and compare the linear programs which we have used. In Section 4.2 we will present cost-decomposition based methods which can be used for computing the MC-bounds and finally in Section 4.3 we will present an approximation algorithm for it. At the end of this chapter we will present the results of some experiments which compare the three methods.

Of course, the faster the methods are the better they are. However besides of the running times, the linear programming approach has two disadvantages, which hopefully can be avoided by using the other approaches. These two disadvantages are on the one hand the fact that the linear programming solver is very memory consuming. For example, the computation of an MC-bound for the DeBruijn graph of dimension 9 with CPLEX requires more than 2 GByte of main memory. Therefore, if we want to tackle graphs of this size we need to use methods which require less memory. On the other hand we will use a branch&bound approach for the computation of the exact cut-size. In a branch&bound approach there is always a valid upper bound and the computation of a lower bound in a specific subproblem can be stopped if the lower bound is bigger than the upper bound or if it becomes clear that the lower bound will not reach the upper bound. Therefore, we want to have methods that deliver lower and upper bounds on the current MC-bound while they compute this bound. These lower and upper bounds on the MC-bound directly correspond to primal and dual values of the linear programs. Unfortunately, in a run of the barrier algorithm or other interior point algorithms the actual primal and dual solutions are infeasible for a long time. So, we cannot trust them and cannot prune the computation of the MC-bounds.

Consequently, beside of the running time we hope that the cost-decomposition method and the approximation algorithm are less memory-consuming and deliver feasible primal and dual solutions more quickly.

4.1 Linear Programming

The use of linear programming for solving multicommodity-flow problems is a standard method. One side-effect of the linear programming formulations is that they directly prove that the computation of the MC-bounds can be done in polynomial time. Furthermore, they allow the use of highly tuned software-packages for solving the problems, e.g. CPLEX.

The main idea which should be taken into account as regards the linear programs is the fact that any MC-solution is scalable without changing the bound. This means that we can change all flows, source-strengths, congestions and the Cut-Flow by any factor. So, we can fix the congestion to be at most one and the cost-function of the linear program corresponds to the maximization of the Cut-Flow.

If we use the notation of the general graph partitioning problem (Definition 1.3) with vertex- and edge-weights, we get the following linear program for the 1-1-MC-bound:

$$\begin{array}{ll} \text{maximize} & N(N-M)\cdot s \\ \text{subject to} & \\ & \forall u,v \in V, u \neq v: \quad s\cdot g(u)g(v) - \sum_{\{v,w\}\in E} h(u,w,v) - h(u,v,w) = 0 \\ & \forall \{v,w\} \in E: \quad \sum_{u\in V} h(u,v,w) + h(u,w,v) \leq f(\{v,w\}) \end{array}$$

With variables $s \in \mathbb{R}$ and $h \in V^3 \to \mathbb{R}_{\geq 0}$. The first constraints ensure that every destination-vertex receives the right amount of commodities, and the second constraints provide a congestion of at most one. The linear program has $2nm+1$ variables (with $n = |V|$ and $m = |E|$) and $n(n-1)+m$ constraints.

For the VarMC-bound we have a linear program of

$$\begin{array}{ll} \text{maximize} & (N-M)\cdot \sum_{u\in V} s(u) \\ \text{subject to} & \\ & \forall u,v \in V, u \neq v: \quad s(u)\cdot g(v) - \sum_{\{v,w\}\in E} h(u,w,v) - h(u,v,w) = 0 \\ & \forall \{v,w\} \in E: \quad \sum_{u\in V} h(u,v,w) + h(u,w,v) \leq f(\{v,w\}). \end{array}$$

It has $n(2m+1)$ variables and $n(n-1)+m$ constraints. And finally, the following linear program can be used for the MVarMC-bound:

Table 4.1: Sizes of the linear programs for the MC-bounds

	variables	constraints	non-zeros
1-1-MC	$2nm+1$	$n(n-1)+m$	$n(n-1)+2m(3n-2)$
VarMC	$2nm+n$	$n(n-1)+m$	$n(n-1)+2m(3n-2)$
MVarMC	$2nm+n^2$	$2n(n-1)+m$	$2n(n-1)+2m(3n-2)$
all	$\Theta(nm)$	$\Theta(n^2)$	$\Theta(nm)$

$$\begin{array}{lrl}
\text{maximize} & \multicolumn{2}{l}{\sum_v \sum_w s(v,w)g(w) - (M-g(v))\bar{s_v}} \\
\text{subject to} & & \\
& \forall u,v \in V, u \neq v: & s(u,v)\cdot g(v) - \sum\limits_{\{v,w\}\in E} h(u,w,v) - h(u,v,w) = 0 \\
& \forall \{v,w\} \in E: & \sum\limits_{u\in V} h(u,v,w) + h(u,w,v) \leq f(\{v,w\}) \\
& \forall u,v \in V: & \bar{s_u} - s(u,v) \geq 0
\end{array}$$

It has $n(n+2m)$ variables and $2n(n-1)+m$ constraints.

Table 4.1 summarizes the sizes of the linear programs. In addition to the number of variables and constraints, the number of non-zero entries in the constraint-matrix is given. The table illustrates that the sizes are asymptotically identical for all three MC-bounds. Therefore, we do not expect different orders of sizes for the costs of solving the linear programs. However since specific implementations of linear program solvers behave differently on different linear programs, we cannot predict the differences between the costs of solving the linear programs. On the other hand the fact that the number of variables and non-zeros depend on m, involves that the computation of the MC-bounds will last clearly longer on dense graphs.

As we have stated in Section 2.3, we used the linear programs presented above for the experimental results presented there. The barrier algorithm of CPLEX was used in the calculation of these results. The use of primal or dual based simplex algorithms lead to running times which are orders of magnitude larger than the use of the barrier algorithm.

4.2 Cost Decomposition

Another way of computing the MC-bounds is the use of cost-based decomposition techniques, which were originally developed for the min-cost multicommodity flow problem. The idea is to relax the capacity constraints (another term used in literature is *bundle constraints*) in order to decompose the problem into a set of shortest path problems. In the following we will only discuss the VarMC-bound.

In this section we will develop an approach on this problem which uses an integration of column generation and Lagrangian relaxation. We assume that the reader is familiar with these two concepts. As an introduction, we refer you to [AMO93]. The following methods are based on the linear program of the VarMC-bound as it has been presented in the Section 4.1.

4.2.1 Column Generation

In principle, a path-based formulation could be be used to build on a column generation approach. This would mean that we would not have variables for every edge and commodity, but that we would have variables which represent whole paths. However, taking into account the large number of source-sink pairs that we are faced with, when computing the VarMC-bound (it is about n^2) we would have to cope with an LP with a large number of constraints. For example, for the DeBruijn of dimension 9 the simplex tableau would have more than 2^{18} rows. These are exactly the same rows as they appear in the linear programs of the preceding section and we have already seen there that we have $\Theta(n^2)$ constraints.

In contrast to the number of columns that can be controlled firstly by generating columns with only negative reduced costs and secondly by successive matrix compressions, such a huge number of rows cannot be handled efficiently. Thus, in practice we cannot afford to generate columns that represent paths in the graph, even though this would be advantageous with respect to the total number of columns that have to be generated in order to achieve a near optimal solution. For the same reason, a master problem consisting of *key paths* and *cycles* which was proposed in [BHJS95] cannot be handled efficiently using our application. Thus, we will use a master problem that is based on trees.

Let $T_v := \{t_{v,1}, \ldots, t_{v,|T_v|}\}$ denote the set of all trees rooted at v and routing the commodities from this sender to its sinks. Define $T := \bigcup_v T_v$ and let $\forall v \in V, t \in T_v : c_{v,t}(e) = \sum_{w \in t(e)} g(w)$ denote the congestion on an edge $e \in E$ with $t(e)$ being the set of all vertices which are "under" the edge e inside the tree (if $e \notin E$ then $t(e) = \emptyset$). Then, the problem can be written as

$$\begin{array}{lrll}
\text{minimize} & C & & \\
\text{subject to} & f(e) \cdot C - \sum\limits_{v \in V} \sum\limits_{t_v \in T_v} s(t_v) \cdot c_{v,t}(e) & \geq 0 & \forall e \in E \\
 & s(t) & \geq 0 & \forall v \in V, t \in T_v \\
 & (N - M) \sum\limits_{v \in V} \sum\limits_{t \in T_v} s(t) & \geq CF &
\end{array}$$

Note that we have used a fixed Cut-Flow CF in this formulation and that we minimize the congestion instead of fixing the congestion and maximizing the Cut-Flow as we have done in the preceding section.

Starting with a subset $T' \subset T$, we solve the reduced *master problem*. Using dual variables $r_e \leq 0$ for the capacity constraints, we generate new columns with negative reduced costs by solving the subproblem

$$\begin{array}{lrll} \text{minimize} & \multicolumn{3}{l}{\sum_{u\in V}\sum_{\{v,w\}\in E}(h(u,v,w)+h(u,w,v))\cdot(-r_{\{v,w\}})} \\ \text{subject to} & \sum_{\{v,w\}\in E} h(u,w,v)-h(u,v,w) & = g(v) & \forall u,v \in V \\ & h(u,v,w) & \geq 0 & \forall u \in V, \forall \{v,w\} \in E. \end{array}$$

Note that the subproblem decomposes into n single-source shortest path problems with non-negative edge costs. We can prune the search inside the branch&bound, if the congestion of the master problem is small enough.

The process of generating columns and solving the master problem is iterated until we can no longer compute trees with associated negative reduced costs or until a master iteration limit is reached. The number of columns in the master matrix is controlled by a frequent compressing step that reduces the number of columns with respect to the current associated reduced costs. However, our experiments have shown that the compression must not be carried out too aggressively, because we need a significantly large number of columns in the master matrix in order to keep the total number of master iterations within reasonable limits. This is a clear drawback of the tree-based formulation of the master problem which results in a relatively high solution time for the master problem.

4.2.2 Lagrangian Relaxation based Column Generation

Thus, we try to keep the number of master iterations to a minimum by adding a whole set of columns between two master problem solutions. The only question that remains to be answered is how meaningful new columns can be generated without the help of new dual variables. We use Lagrangian relaxation for this purpose, firstly on a min-congestion formulation of the problem, and secondly, on a max-Cut-Flow representation. The idea to use Lagrangian relaxation to generate new columns is motivated by the fact that the optimal Lagrangian multipliers in the min-congestion formulation are also optimal dual values for the column generation procedure and vice versa.

The whole procedure works as follows: we start a subgradient optimization of the Lagrangian dual and we achieve an upper bound for the Cut-Flow or a lower bound on the congestion, respectively. At the same time, we feed the tree-flows which are computed in the successive Lagrangian subproblems into the matrix of the master problem. If the upper bound on the maximum Cut-Flow is smaller than a specific threshold or if the lower bound on the minimum congestion is greater than a specific threshold, respectively, we have proved that the VarMC-bound will not allow the current search node to be pruned and we can therefore branch right away. Otherwise, we solve the master problem and achieve a feasible flow which yields an upper bound on the congestion. If we achieve a flow with an associated congestion that is small enough, we can prune the current search node. Otherwise, we must restart the Lagrangian subroutine again with the current dual values. This process is iterated until we find optimal flows or until an iteration limit is reached.

To complete the description, we will briefly present in more detail the two Lagrangian relaxations which we used to generate columns in a column generation framework. By relaxing the capacity constraints in the linear program of the VarMC-bound using Lagrangian multipliers $r_e \leq 0$, we get a max-Cut-Flow formulation of

$$\begin{array}{llll} \text{maximize} & \multicolumn{3}{l}{\sum_{u\in V} s(u) + \sum_{\{v,w\}\in E}(h(u,v,w)+h(u,w,v))r_{\{v,w\}}} \\ \text{subject to} & \sum_{\{v,w\}\in E} h(u,w,v)-h(u,v,w) & = s(u)g(v) & \forall u,v \in V \\ & h(u,v,w) & \geq 0 & \forall u\in V, \forall\{v,w\}\in E \end{array}$$

Or, we can aim for a min-congestion formulation of

$$\begin{array}{llll} \text{minimize} & \multicolumn{3}{l}{\sum_{u\in V}\sum_{\{v,w\}\in E}(h(u,v,w)+h(u,w,v))\cdot(-r_{\{v,w\}})} \\ & \sum_{\{v,w\}\in E} h(u,w,v)-h(u,v,w) & = s(u)g(v) & \forall u,v \in V \\ \text{subject to} & h(u,v,w) & \geq 0 & \forall u\in V, \forall\{v,w\}\in E \\ & \sum_v s(v) & \geq CF & \end{array}$$

In both cases, we need to solve n single-source shortest path problems again. Thus, both formulations allow us to use the shortest path trees which were computed in the Lagrangian subproblems to generate columns for the master problem. Note, that both Lagrangian subproblems may be unbounded. This problem is overcome by setting upper bounds to $s(v)$ (for example the out-capacity of v) or to C (for example $1.1 \cdot$ branch), respectively.

The Lagrangian multipliers can be updated by different subgradient algorithms using different formulas for the computation of the new search direction d^t. Let s^t denote a subgradient in iteration t. We can set $d^t = s^t$ (pure subgradient); or $d^t = \alpha d^{t-1} + s^t$, whereby $0 < \alpha < 1$ is fix (so called *Crowder rule* [Cro76]); or we may set $d^t = \alpha^t d^{t-1} + s^t$, with $\alpha^t = ||s^t||/||d^{t-1}||$ if $(s^t)^T d^{t-1} \leq 0$, and $\alpha = 0$ otherwise (*modified Camerini-Fratta-Maffioli rule* [CFM75]); another possibility is setting $d^t = \alpha d^{t-1} + (1-\alpha)s^t$, whereby $0 < \alpha < 1$ is fix (so called *Volume Algorithm* [BA00]). All variants will be evaluated in the Section 4.4.

4.3 Approximation Algorithm

4.3.1 Literature and Idea

There are a couple of results that deal with the approximation of multicommodity flow problems. In the following we give a small survey on the most important results for our application. There are a lot of different versions of multicommodity flow problems. For our application of computing the MC-bounds, the *maximum multicommodity flow* problem and the *maximum concurrent flow* problem are the most interesting ones since they have the biggest similarity to the computation of the MC-bounds. In both problems a set of commodities K (which are pairs of vertices with a demand d_k) and a bound on the congestion for every edge is given. In the maximum multicommodity flow problem the sum of all flows of the commodities has to be maximized, i.e. $\max \sum_{k \in K} s(k)$. In the maximum concurrent flow problem a specific quota α of all commodities has to be satisfied, this quota has to be maximized, i.e. $\max \alpha$ with $\forall k : s(k) = \alpha d_k$.

Obviously, the 1-1-MC-bound can directly be expressed as a maximum concurrent flow problem. The VarMC-bound is a mixture of the maximum multicommodity flow problem and the maximum concurrent flow problem: Some commodities from the same sender are connected so that they must be satisfied with the same portion (i.e. maximum concurrent flow), while the sum of all these portions has to be maximized (i.e. maximum multicommodity flow).

One of the most recent work on these problems is done by Fleischer [Fle00]. In contrast to our problems these results concern directed graphs. For the maximum multicommodity flow problem they give an ε-approximation algorithm in time $O^*(\varepsilon^{-2}m^2)$. Notice, that the running time is independent from the number of commodities. For the maximum concurrent flow problem an algorithm with running time $O^*(\varepsilon^{-2}m(m+k))$ is given. The notation O^* is the O-notation where logarithmic factors are omitted.

Algorithm 1: Informal approximation Algorithm for the maximum multicommodity flow problem due to Fleischer

```
 1: init $\forall e \in E : l(e) \leftarrow \delta$;
 2: **repeat**
 3:    **for all** commodities $k$ **do**
 4:       $P \leftarrow$ shortest path for commodity $k$ according $l(e)$;
 5:       **if** $cost(P)$ small enough **then**
 6:          include $P$ into solution;
 7:          update $l(e)$;
 8:       **end if**
 9:    **end for**
10: **until** finished;
11: scale solution such that congestion is fulfilled;
```

Both algorithms are based on similar ideas, we present the ideas for the maximum multicommodity flow problem: Algorithm 1 illustrates the main procedure of the algorithm. The solution is built iteratively, i.e. we start with no flow at all and add commodities with their paths to the current solution. This is done again and again until a specific end-criterion is reached. Furthermore, edge-lengths are used where heavily used edges have a big length. In every iteration the commodity which has lowest cost is taken with an according path and this commodity with the path is added to the solution and the edge-lengths of the used edges are increased accordingly. The cost of a specific commodity with its path is the sum of the lengths of the used edges.

Furthermore, by relaxing that not the commodity with the path with minimal cost has to be used but a commodity with a path with nearly minimal cost can be used, the running time is improved.

The adaption of this algorithm to the VarMC-bound is done by not routing only one commodity with its path in every iteration but by routing all the commodities from one sender to all destinations according a shortest-path tree in one iteration. The adaption of this to the MVarMC-bound is done by allowing to use parts of the shortest-paths tree (which corresponds to parts of the set of destinations). The selection of the best part is done according a specific cost-term.

4.3.2 The Approximation Algorithm

In this section we present the approximation algorithm for the VarMC-bound and MVarMC-bound. As we will see in Chapter 5 the bounds which have to be computed inside the branch&bound algorithm are not the clean VarMC- or MVarMC-bounds as they are define in Chapter 2 but a kind of mixture of these variants has to be computed. Due to this we present an approximation algorithm for a more general problem. It will be obvious that the VarMC-bound and MVarMC-bound are covered in this more general model.

The following table shows the input of a bound computation:

$G=(V,E)$	the given directed graph
$g:V\rightarrow\mathbb{N}$	vertex-weights
$f:E\rightarrow\mathbb{R}_{\geq 0}$	edge-weights
$K=K_1\cup K_2$	set of commodities
$o:K\rightarrow V$	origin of commodities
$D:K\rightarrow 2^V$	destinations of commodities
$F_1:K_1\rightarrow\mathbb{N}$	Cut-Flow-factor for VarMC
$F_2:K_2\rightarrow\mathbb{N}$	negative Cut-Flow-factor for MVarMC

Additionally, we use $K_{a,v,w} := \{k\in K_a | o_k = v \wedge w\in D_k\}$ with $a\in\{1,2\}$. We point out that we use a directed graph in this model. However, since in the graph-partitioning problem the graphs are undirected, we assume that any edge is either present in both direction in the graph or it is not present at all. The following formulation of the bound-computation as a linear program uses variables:

$s_1:K_1\rightarrow\mathbb{R}_{\geq 0}$	sender-strength for VarMC-commodities
$s_2:K_2\times V\rightarrow\mathbb{R}_{\geq 0}$	sender-strength for MVarMC-commodities
$\hat{s}_2:K_2\rightarrow\mathbb{R}_{\geq 0}$	maximal strength of MVarMC-commodities
$h_{1,2}:V\times E\rightarrow\mathbb{R}_{\geq 0}$	flow of source v on an edge e

Primal Formulations

Using this the linear program is:

$$\text{maximize} \sum_{k\in K_1} F_{1,k}\cdot s_1(k) \;+\; \sum_{k\in K_2}\sum_{v\in D_k} g_v\cdot s_2(k,v) - F_{2,k}\cdot \hat{s}_2(k)$$

with

$$\begin{array}{rrcl}
\forall e\in E: & \sum_{v\in V} h_1(v,e)+h_2(v,e) & \leq & f_e\\
\forall v,w\in V, v\neq w: & \sum_{e=(u,w)} h_1(v,e)-h_2(v,e) & & \\
 & +\sum_{e=(w,u)} h_2(v,e)-h_1(v,e) & \geq & g_w \sum_{k\in K_{1,v,w}} s_1(k)\\
 & & & +g_w \sum_{k\in K_{2,v,w}} s_2(k,w)\\
\forall k\in K_2, w\in D_k: & s_2(k,w) & \leq & \hat{s}_2(k)
\end{array}$$

Throughout this section we use a notation with underscore indices if variables are involved which are part of the input (e.g. D_k). In contrast to this we use a functional notation if variables of the linear program or of the algorithm are involved (e.g. $s_1(k)$). It is clear that the above formulation of the problem includes the VarMC-bound and the MVarMC-bound. For example, if the VarMC-bound for a bisection problem has to be computed we have $K_2 = \emptyset$, $\forall k: D_k = V \wedge F_{1,k} = \frac{n}{2}$.

The above primal linear program is edge-based. It can also be expressed in a tree-based form. Then we use

$T_{k,\overline{V}}$: set of all trees with root o_k and destinations $\overline{V}$ with $\overline{V}\subseteq D_k$.

Additionally, we use $g: 2^V \to \mathbb{R}_{\geq 0}$ with $g_V := \sum_{v\in V} g_v$ and $g: T_{k,\overline{V}}\times E \to \mathbb{R}_{\geq 0}$ with $g_{t,e} := \sum_{v\in V_{t,e}} g_v$ with $V_{t,e}$ is the set of vertices which uses edge e for their path inside tree t. Only two variables are necessary:

$s_1: T_{k,\overline{V}} \to \mathbb{R}_{\geq 0}$ sender-strength, with $k\in K_1, \overline{V}\subseteq V$
$s_2: T_{k,\overline{V}} \to \mathbb{R}_{\geq 0}$ sender-strength, with $k\in K_2, \overline{V}\subseteq V$

The linear program has the form:

$$\text{maximize} \sum_{k\in K_1} F_{1,k}\cdot \sum_{t\in T_{k,D_k}} s_1(t) \;+\; \sum_{k\in K_2}\sum_{\overline{V}\subseteq D_k} (g_{\overline{V}} - F_{2,k})\cdot \sum_{t\in T_{k,\overline{V}}} s_2(t)$$

$$\text{with } \forall e\in E: \sum_{t\in T_{k,\overline{V}}} g_{t,e}(s_1(t)+s_2(t)) \leq f_e$$

Algorithm 2: Approximation algorithm for the MC-bounds

```
1:  $\forall e \in E : l(e) \leftarrow \delta$;
2:  Init $\hat{\alpha}$;          //according Lemma 4.1
3:  repeat
4:      for all $k \in K$ do
5:          $t \leftarrow$ RelShortPath(k,l);
6:          while $cost_l(t,k) < \min\{1,(1+\varepsilon)\hat{\alpha}\}$ do
7:              route(k,t,l,h);   //realize flow according t
8:              $t \leftarrow$ RelShortPath(k,l);
9:          end while
10:     end for
11:     $\hat{\alpha} \leftarrow \hat{\alpha}(1+\varepsilon)$;
12: until $\hat{\alpha} \geq 1$;
13: scale solution such that congestion is fulfilled;
```

Dual formulation

Using the tree-based primal formulation, the following dual form arises:

$$\text{minimize} \sum_{e \in E} f_e l(e)$$

$$\begin{array}{lrcl} \text{with} & \forall k \in K_1, t \in T_{k,D_k}: & \sum_{e \in t} g_{t,e} l(e) & \geq & F_{1,k} \\ & \forall k \in K_2, \overline{V} \subseteq D_k \text{ with } g_{\overline{V}} > F_{2,k}, t \in T_{k,\overline{V}}: & \sum_{e \in t} g_{t,e} l(e) & \geq & g_{\overline{V}} - F_{2,k} \end{array}$$

Notice, that the restrictions are equivalent to $\forall k \in K_1 : \sum_{v \in D_k} g_v dist_{k,v}(l) \geq F_{1,k}$ and $\forall k \in K_2, \overline{V} \subseteq D_k$ with $g_{\overline{V}} > F_{2,k} : \sum_{v \in \overline{V}} g_v dist_{k,v}(l) \geq g_{\overline{V}} - F_{2,k}$ with $dist_{k,v}(l)$ is the distance of vertex o_k to vertex v corresponding to the length-function $l(e)$.

The Algorithm

Algorithm 2 is the approximation algorithm for the VarMC- and MVarMC-bound.

The initial value of $\hat{\alpha}$ has to be smaller than the minimal cost of any routing tree. The function `RelShortPath(k,l)` computes the shortest-path-tree from source o_k to all destinations D_k; additionally, if $k \in K_2$ a subset $\overline{V} \subseteq D_k$ with $g_{\overline{V}} > F_{2,k}$ is computed and used as set of destinations such that $cost_l(t,k)$ is minimized. Following the dual linear program, we use

$$cost_l(t,k) := \frac{1}{factor_{t,k}} \sum_{e \in t} g_{t,e} l(e)$$

Algorithm 3: `route(k,t,l,h):` realization of a routing inside the approximation algorithm

```
1:  $\phi \leftarrow \min_{e \in t} \frac{2\bar{h}(o_k,e)+f_e}{g_{t,e}}$; where $\bar{h}$ is the flow in opposite direction than in $t$
2:  for all $e \in E$ do
3:      $\bar{\Delta}_e \leftarrow \min\{\bar{h}(o_k,e), \phi g_{t,e}\}$;
4:      $\Delta_e \leftarrow \phi g_{t,e} - \bar{\Delta}_e$;
5:      if $\Delta_e - \bar{\Delta}_e > 0$ then
6:          $l(e) \leftarrow l(e)(1+\varepsilon\frac{\Delta_e - \bar{\Delta}_e}{f_e})$;
7:      end if
8:      $h(o_k,e) += \Delta_e$;
9:      $\bar{h}(o_k,e) -= \bar{\Delta}_e$;
10: end for
```

with

$$\forall k, t \in T_{k,\overline{V}} : \quad factor(t,k) := \begin{cases} F_{1,k} & \text{if } k \in K_1 \\ g_{\overline{V}} - F_{2,k} & \text{if } k \in K_2. \end{cases}$$

Therefore, the computed tree t with minimal costs corresponds to that tree whose violation of the constraint in the dual formulation is the "biggest".

The realization `route(k,t,l,h)` of a given tree t, commodity k, edge-lengths l and flows h is done by Algorithm 3.

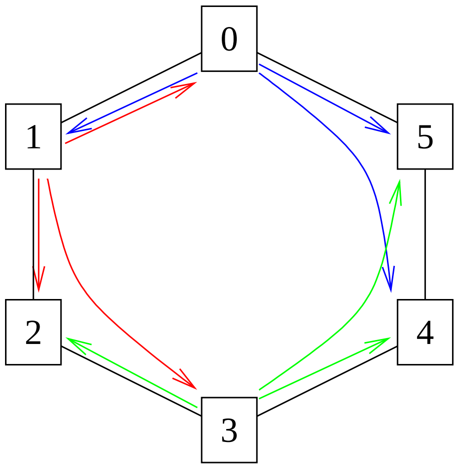
Table 4.2 illustrates a run of the algorithm, in this example the MVarMC-bound is computed for a 3 -partitioning problem and the graph is a 6-node-circle. Every row in the table corresponds to a commodity which is realized with the `route()`-procedure. The column "v" gives the sender-vertex of a realized flow, "dest." is the set of destinations, "ϕ" is the amount of flow which is realized according Algorithm 3, "obj." gives the primal value which is obtained after the actual commodity is routed and "$l'(v,w)$" gives the length-function of the corresponding edge with $l'(v,w) \cdot \delta = l(v,w)$. You can see that the solution which will be computed by the algorithm corresponds to the one which is illustrated in Figure 4.3. However, the approximation algorithm makes a "mistakes" by its first realized routing. All subsequent routings correspond to the illustrated one and the objective approaches three, which is the exact MVarMC-bound.

This example also demonstrates, that the resulting MC-solution depends on the ordering of the vertices in the graph. If the graph in the example would be ordered, such that the vertices 0, 2, 4 would be the first, the algorithm would produce an instance where these three vertices send commodities of unit size to their neighbors. Doing so, the primal value would already be optimal after the first round.

Table 4.2: Approximation of the MVarMC-bound for the 6-node-ring with $k = 3$

v	dest.	ϕ	obj	$l'(0,1)$	$l'(1,2)$	$l'(2,3)$	$l'(3,4)$	$l'(4,5)$	$l'(5,0)$
0	1,5	1.0	1.00	1.10	1.00	1.00	1.00	1.00	1.10
1	0,2,3	0.5	1.33	1.16	1.10	1.05	1.00	1.00	1.10
3	2,4,5	0.5	2.00	1.16	1.10	1.10	1.10	1.05	1.10
0	1,4,5	0.5	2.00	1.21	1.10	1.10	1.10	1.10	1.21
1	0,2,3	0.5	2.00	1.27	1.21	1.16	1.10	1.10	1.21
3	2,4,5	0.5	2.40	1.27	1.21	1.22	1.21	1.16	1.21
0	1,4,5	0.5	2.33	1.34	1.21	1.22	1.21	1.22	1.33
1	0,2,3	0.5	2.29	1.40	1.33	1.28	1.21	1.22	1.33
3	2,4,5	0.5	2.57	1.40	1.33	1.34	1.33	1.28	1.33
0	1,4,5	0.5	2.50	1.47	1.33	1.34	1.33	1.34	1.46
1	0,2,3	0.5	2.44	1.55	1.46	1.41	1.33	1.34	1.46
3	2,4,5	0.5	2.67	1.55	1.46	1.48	1.46	1.41	1.46
0	1,4,5	0.5	2.60	1.63	1.46	1.48	1.46	1.48	1.61
1	0,2,3	0.5	2.55	1.71	1.61	1.55	1.46	1.48	1.61
3	2,4,5	0.5	2.73	1.71	1.61	1.63	1.61	1.55	1.61
0	1,4,5	0.5	2.67	1.79	1.61	1.63	1.61	1.63	1.77
1	0,2,3	0.5	2.62	1.88	1.77	1.71	1.61	1.63	1.77
3	2,4,5	0.5	2.77	1.88	1.77	1.80	1.77	1.71	1.77

Table 4.3: Illustration of the resulting flows of Table 4.2



Cut-Flow $CF = 3 \cdot 2 = 6$
congestion $C = 2$
$\rightarrow$ bound$= 3$

Correctness of the Algorithm

Now, we look at the correctness of running time of the algorithm. The first lemma shows a valid initial value for $\hat{\alpha}$:

Lemma 4.1
An initial value of $\hat{\alpha}$ *with* $\hat{\alpha} = \delta \min \left\{ \min_{k \in K_1} \frac{g_{D_k}}{F_{1,k}}, \min_{k \in K_2} \left(1 - \frac{F_{2,k}}{g_{D_k}}\right)^{-1} \right\}$ *is a lower bound on the cost of every routing with edge-lengths* δ*.*

Proof:
Firstly we look at $k \in K_1$, let β be the minimal cost then

$$\begin{aligned} \beta &= \min_{k \in K_1} \frac{1}{F_{1,k}} \sum_{v \in D_k} g_v dist_{k,v}(l(0)) \\ &\geq \delta \min_{k \in K_1} \frac{g_{D_k}}{F_{1,k}} \end{aligned}$$

Accordingly, for $k \in K_2$ let β be the minimal cost then

$$\begin{aligned} \beta &= \min_{k \in K_2} \min_{\bar{V} \subseteq D_k} \frac{1}{g_{\bar{V}} - F_{2,k}} \sum_{v \in \bar{V}} g_v dist_{k,v}(l(0)) \\ &\geq \delta \min_{k \in K_2} \min_{\bar{V} \subseteq D_k} \frac{g_{\bar{V}}}{g_{\bar{V}} - F_{2,k}} \\ &= \delta \min_{k \in K_2} \frac{1}{1 - \frac{F_{2,k}}{g_{D_k}}} \end{aligned}$$

Putting this together proves the lemma. ■

The next lemma shows that every realization using `route()` generates not too much and not too less load on some edges:

Lemma 4.2
In every iteration where the current flow is changed according a tree t
a) $\forall e \in t : \Delta_e - \overline{\Delta}_e \leq f_e$
b) $\exists e \in t : \Delta_e - \overline{\Delta}_e = f_e$

holds.

Proof:
Let k be the commodity which is realized by tree t. We look at an edge e, let $\overline{h}(e)$ be the flow of commodity k of the already computed solution on edge e which goes in the opposite direction as it goes in tree t.

The algorithm calculates $\Delta_e - \overline{\Delta}_e = \phi g_{t,e} - 2\overline{\Delta}_e = \phi g_{t,e} - 2\min\{\overline{h}(e), \phi g_{t,e}\}$. So

case $\bar{h}(e) < \phi g_{t,e}$: Then $\Delta_e - \bar{\Delta}_e = \phi g_{t,e} - 2\bar{h}(e) \leq 2\bar{h}(e) + f_e - 2\bar{h}(e) = f_e$.

case $\bar{h}(e) > \phi g_{t,e}$: Then $\Delta_e - \bar{\Delta}_e = \phi g_{t,e} - 2\phi g_{t,e} = -\phi g_{t,e} < f_e$. (clear, since $\phi > 0$ and $g_{t,e} > 0$).

Now look at edge e with minimal $\frac{2\bar{h}(e)+f_e}{g_{t,e}}$. Then $\phi g_{t,e} = 2\bar{h}(e) + f_e$, the first of the two above cases enters and we have equality instead of lower-equal. ■

The next lemma shows that we get a feasible flow using a scaling depending on the values of $l(e)$:

Lemma 4.3
Let $X_1(l) := \max_e l(e)$. Then we get a feasible flow if we use a scaling factor of $\frac{1}{\sigma}$ with $\sigma = \log_{1+\varepsilon} \frac{X_1(l)}{\delta}$.

Proof:
Let T be the set of all trees which are routed inside the algorithm. Let $\Delta_e(t)$ be the flow on edge e regarding tree t according to Algorithm 3. We have to show that $\forall e \in E$: $\frac{1}{\sigma}\sum_{t\in T} \Delta_e(t) - \bar{\Delta}_e(t) \leq f_e$. It we look at $l(e)$ and use $\forall 0 \leq x \leq 1 : (1+\varepsilon x) \geq (1+\varepsilon)^x$ we have

$$\begin{aligned}
\forall e \in E:\ l(e) &\geq \delta\Pi_t(1+\varepsilon\frac{\Delta_e(t)-\bar{\Delta}_e(t)}{f_e}) \\
&\geq \delta\Pi_t(1+\varepsilon)^{\frac{\Delta_e(t)-\bar{\Delta}_e(t)}{f_e}} \\
&= \delta(1+\varepsilon)^{\frac{1}{f_e}\sum_t \Delta_e(t)-\bar{\Delta}_e(t)} \\
\Rightarrow \sum_{t\in T} \Delta_e(t) - \bar{\Delta}_e(t) &\leq f_e \log_{1+\varepsilon}\frac{l(e)}{\delta}
\end{aligned}$$

Since $X_1(l) = \max_e l(e)$ the lemma is proved. ■

Looking at the given restriction of the dual linear program, we have upper bounds on $l(e)$ after the algorithm has terminated:

Lemma 4.4
Let be $X_1 := (1+\varepsilon)\max\{\max_{k\in K_1}\{F_{1,k}\}, \max_{k\in K_2}\{g(D_k) - F_{2,k}\}\}$. At the end of the algorithm $X_1(l) \leq X_1$ holds.

Proof:
Just before any edge e is used for the last time in realizing the flow of a tree t (with $e \in t$), $cost_l(t,k) < 1$ must hold. So, since $g_{t,e} \geq 1$, it follows that $l(e) \leq F_{1,k}$ or resp.

$l(e) \leq g_{\bar{V}} - F_{2,k}$. Considering the last increase of $l(e)$ realizing the tree t, the lemma follows. ■

The next two lemmas together give bounds on the goodness of the computed solution of the algorithm.

Lemma 4.5
Using $\alpha(l) := \min_{k,t} cost_l(t,k)$, $X_2 = n\max\{\max_{k\in K_1}\{\frac{g_{D_k}}{F_{1,k}}\}, \max_{k\in K_2}\{F_{2,k}+1\}\}$ *and let* $l(i)$ *be the length function of the algorithm after i flow realizations. Then*

$$\alpha(l(i)-l(0)) \geq \alpha(l(i)) - \delta X_2$$

holds.

Proof:
First look at $k \in K_1$. Then we have

$$\begin{aligned}
\alpha(l(i)-l(0)) &= \min_{k\in K_1}\min_{t\in T_{k,D_k}} \frac{1}{factor(t,k)}\sum_{e\in t} g_{t,e}(l_e(i)-l_e(0)) \\
&= \min_{k\in K_1} \frac{1}{factor(t,k)}\sum_{v\in D_k} g_v \cdot dist_{k,v}(l_e(i)-l_e(0)) \\
&\geq \min_{k\in K_1} \frac{1}{factor(t,k)}\sum_{v\in D_k} g_v \cdot (dist_{k,v}(l_e(i)) - \delta n) \\
&= \alpha(l(i)) - \delta n \max_{k\in K_1} \frac{g_{D_k}}{F_{1,k}}.
\end{aligned}$$

Secondly we look at $k \in K_2$:

$$\begin{aligned}
\alpha(l(i)-l(0)) &= \min_{k\in K_2}\min_{\overline{V}\subseteq D_k}\min_{t\in T_{k,\overline{V}}} \frac{1}{factor(t,k)}\sum_{e\in t} g_{t,e}(l_e(i)-l_e(0)) \\
&= \min_{k\in K_2}\min_{\overline{V}\subseteq D_k} \frac{1}{g_{\bar{V}} - F_{2,k}}\sum_{v\in\overline{V}} g_v dist_{k,v}(l_e(i)-l_e(0)) \\
&\geq \min_{k\in K_2}\min_{\overline{V}\subseteq D_k} \frac{1}{g_{\bar{V}} - F_{2,k}}\sum_{v\in\overline{V}} g_v (dist_{k,v}(l_e(i)) - \delta n) \\
&\geq \alpha(l(i)) - \max_{k\in K_2}\max_{\overline{V}\in D_k} \frac{g_{\bar{V}}}{g_{\bar{V}} - F_{2,k}} n\delta \\
&\geq \alpha(l(i)) - \delta n \max_{k\in K_2} F_{2,k} + 1
\end{aligned}$$

Putting this together, the lemma is proved. ■

Now:

Lemma 4.6
Using $\delta = X_1 \cdot (X_1X_2)^{-1/\varepsilon}$*, the final flow scaled by* $\frac{1}{\sigma} = \frac{\varepsilon}{\log_{1+\varepsilon} X_1X_2}$ *is optimal with a relative error of at most* 3ε*.*

Proof:
We prove the desired accuracy of the solution computed by comparing it against the objective value D of a dual feasible solution, which our algorithm produces as a byproduct, and which gives us a valid upper bound on the primal optimal solution value Z_{opt}.

For any given length function $l : E \to \mathbb{R}_{\geq 0}$, denote with α the minimal routing costs of all commodities, i.e., $\alpha = \alpha(l) = \min_{k,t} cost_l(t,k)$. Further, denote with $D(l)$ the dual objective value corresponding to the current choice of l, i.e., $D(l) = \sum_{e \in E} f_e l(e)$. Then, the optimal dual objective value is $Z_{opt} = \min_l D(l)/\alpha(l)$.

Consider iteration i, and denote with $l(i)$, k, ϕ, and t the current choice of the length function, commodity, scaling factor, and routing tree, respectively. For ease of notation, we set $\alpha(i) = \alpha(l(i))$, $D(i) = D(l(i))$, and $dist_{k.v}(i) = dist_{k,v}(l(i))$. For any edge $e \in E$, define $\Gamma_e = \max\{\Delta_e(i) - \overline{\Delta}_e(i), 0\}$. Note, that in every iteration i it holds $\Gamma_e \leq \phi g_{t,e}$. Then,

$$\begin{aligned} D(i) &= D(i-1) + \varepsilon \sum_{e \in t} \Gamma_e l_e(i-1) \\ &\leq D(i-1) + \varepsilon \sum_{e \in t} \phi g_{t,e} l_e(i-1) \\ &= D(i-1) + \varepsilon\phi \cdot factor(t,k) \cdot cost_l(t,k) \\ &< D(i-1) + \varepsilon\phi \cdot factor(t,k)(1+\varepsilon)\alpha(i-1) \end{aligned}$$

Denote with $Z(i)$ the primal objective value (corresponding to the — possibly infeasible — flows h) after the iteration i ($Z(0) = 0$). Obviously, it holds $Z(i) \geq Z(i-1) + factor(t,k)\phi \Leftrightarrow \phi \leq \frac{Z(i)-Z(i-1)}{factor(t,k)}$. Thus, $D(i) < D(i-1) + \varepsilon(1+\varepsilon)(Z(i) - Z(i-1))\alpha(i-1)$, and therefore

$$D(i) < D(0) + \varepsilon(1+\varepsilon) \sum_{1 \leq h \leq i} (Z(i) - Z(i-1))\alpha(i-1). \tag{4.1}$$

Consider the length function $l(i) - l(0) = l(i) - \delta$. Then, for the dual objective we have $D(l(i) - l(0)) = D(i) - D(0)$. For the primal objective we have $\alpha(l(i) - l(0)) = \alpha(i) - \delta X_2$. Hence,

$$Z_{opt} \leq \frac{D(l(i) - l(0))}{\alpha(l(i) - l(0))} \leq \frac{D(t) - D(0)}{\alpha(i) - \delta X_2}$$

And thus,

$$\alpha(i) < \delta X_2 + \frac{\varepsilon(1+\varepsilon)}{Z_{opt}} \sum_{1 \leq h \leq i} (Z(h) - Z(h-1))\alpha(h-1) \tag{4.2}$$

Denote with $A(i)$ the right hand side of inequality 4.2. Then,

$$\begin{aligned} A(i) &= A(i-1) + \frac{\varepsilon(1+\varepsilon)}{Z_{opt}}(Z(i)-Z(i-1))\alpha(i-1) \\ &< A(i-1)(1+\frac{\varepsilon(1+\varepsilon)}{Z_{opt}}(Z(i)-Z(i-1))) \qquad \text{since } \alpha(i-1) < A(i-1) \\ &\leq A(i-1)e^{\frac{\varepsilon(1+\varepsilon)}{Z_{opt}}(Z(i)-Z(i-1))} \qquad \text{since } 1+x < e^x \forall x \in \mathbb{R} \\ &= A(0)e^{\frac{\varepsilon(1+\varepsilon)Z(i)}{Z_{opt}}} \qquad \text{since Z}(0)=0. \end{aligned}$$

Now consider the last iteration i. Then, $\alpha(i) \geq 1$. With $A(0) = \delta X_2$ we get

$$1 \leq \alpha(i) < A(i) < \delta X_2 e^{\frac{\varepsilon(1+\varepsilon)Z(i)}{Z_{opt}}} \tag{4.3}$$

And thus

$$Z(i) > \frac{Z_{opt}}{\varepsilon(1+\varepsilon)} \ln \frac{1}{\delta X_2} \tag{4.4}$$

In order to get $\frac{Z(i)}{\sigma} > Z_{opt}\frac{1-\varepsilon}{\varepsilon(1+\varepsilon)}\ln(1+\varepsilon)$ we look at

$$\begin{aligned} \frac{Z_{opt}}{\varepsilon(1+\varepsilon)} \ln \frac{1}{\delta X_2} &= \sigma Z_{opt}\frac{1-\varepsilon}{\varepsilon(1+\varepsilon)}\ln(1+\varepsilon) && (4.5) \\ \Leftrightarrow \quad \delta &= \frac{1}{X_2(1+\varepsilon)^{\sigma(1-\varepsilon)}} && (4.6) \\ & \qquad \text{with } \sigma = \log_{1+\varepsilon}\frac{X_1}{\delta} && (4.7) \\ \Leftrightarrow \quad \delta &= X_1(X_1X_2)^{-1/\varepsilon} && (4.8) \end{aligned}$$

So we have

$$\frac{Z(i)}{\sigma} > Z_{opt}\frac{1-\varepsilon}{\varepsilon(1+\varepsilon)}\ln 1+\varepsilon \quad \Rightarrow \quad \frac{Z(i)}{\sigma} > Z_{opt}(1-3\varepsilon), \tag{4.9}$$

for all $\varepsilon < 1$. ■

Therefore, so correctness of the approximation is shown. It remains to show the running time:

Lemma 4.7
The Algorithm 2 runs in time $O^*(\varepsilon^{-2}m^2)$, if $g(V) = poly(m)$ and $\forall k : F_k = poly(m)$.

Proof:
Obviously, the running time of the algorithm is dominated by the calculation of the

shortest path trees. Let p be the number of tree-computations without a following `route()`, and let q be the number of tree-computations with a following `route()`. Then the total algorithm needs time $O((p+q)T_{spt})$ where T_{spt} is the time for computing a shortest-path-tree.

Firstly, we look at p: The initial value $\hat{\alpha}_0$ of $\hat{\alpha}$ is known from Lemma 4.1. And at the end $\hat{\alpha} < (1+\varepsilon)$ is clear. So, for the number I of iterations of the outer repeat-until-loop holds

$$\begin{aligned} \hat{\alpha}_0(1+\varepsilon)^I &< 1+\varepsilon \\ \Rightarrow \quad I &< \log_{1+\varepsilon}\frac{1+\varepsilon}{\hat{\alpha}_0} = O^*(\varepsilon^{-2}). \end{aligned}$$

At a first glance we have $p = O^*(\varepsilon^{-2}|K|)$. But, by ordering the commodities according their origin and using the fact that succeeding calculations of shortest-path-tree from the same origin can be done together, we have $p = O^*(\varepsilon^{-2}n)$.

Secondly, we look at q: At the beginning we have $\forall e : l(e) = \delta$, and according Lemma 4.4 at the end of the algorithm $\forall e : l(e) \leq X_1$ holds. Furthermore, in every `route()` there is at least one edge with $l(e) = l(e)(1+\varepsilon)$. So we have

$$\begin{aligned} \delta(1+\varepsilon)^{q/m} &\leq X_1 \\ \Rightarrow \quad q &\leq m\log_{1+\varepsilon}\frac{X_1}{\delta} = O^*(\varepsilon^{-2}m) \end{aligned}$$

Finally, with $m = \Omega(n)$ and $T_{spt} = O(m\log m)$ using the Dijkstra-algorithm, a total running time of $O^*(\varepsilon^{-2}m^2)$ is shown. ■

The following theorem sums up the results:

Theorem 4.1
Algorithm 2 is able to compute an ε-approximation of the VarMC-bound and MVarMC-Bound in time $O^*(\varepsilon^{-2}m^2)$.

4.3.3 Implementation Details

Three main practical improvements have been suggested for approximation algorithms for multicommodity flow problems, see e.g. [GOPS98, Rad00, Fle00]:

- Firstly, it is suggested to use a better estimation for the ε which is used in the algorithm instead of using the theoretical approach. This makes sense if you want to compute a solution with a given maximal error as fast as possible. But in our context we have to select an ε which fits best for the branch&bound algorithm, this fitting is a practical trade-off and we will have to carry out experiments in order to quantify the best ε, anyway. Therefore, this suggestion is of no use for us.

- Secondly, in [Rad00, Fle00] it is suggested to use a different length-function-update: Instead of using $l(e) \leftarrow l(e)(1+\varepsilon\frac{\Delta}{f_e})$, as it is done in line 6 of Algorithm 3, you can also use $l(e) \leftarrow l(e)e^{\varepsilon\frac{\Delta}{f_e}}$. This modified length-function-update does not change the analysis of the algorithm but it is reported that the algorithm behaves better.

- Thirdly, it is proposed to use a variable ϕ in the realization of the actual routing tree instead of the fixed ϕ as it is given in the Algorithm 3 (line 1). This suggestion comes from approximation algorithms for multicommodity flow problems previous to [GK98] and [Fle00] which work slightly different. There it is suggested to choose ϕ such that a specific potential function is minimized. This corresponds to a minimization of the dual value of the problem. However, in our algorithm it is not possible to efficiently compute the ϕ which minimizes the dual value. Instead, it is possible to compute the ϕ which maximizes the primal value; so we have tested this variation.

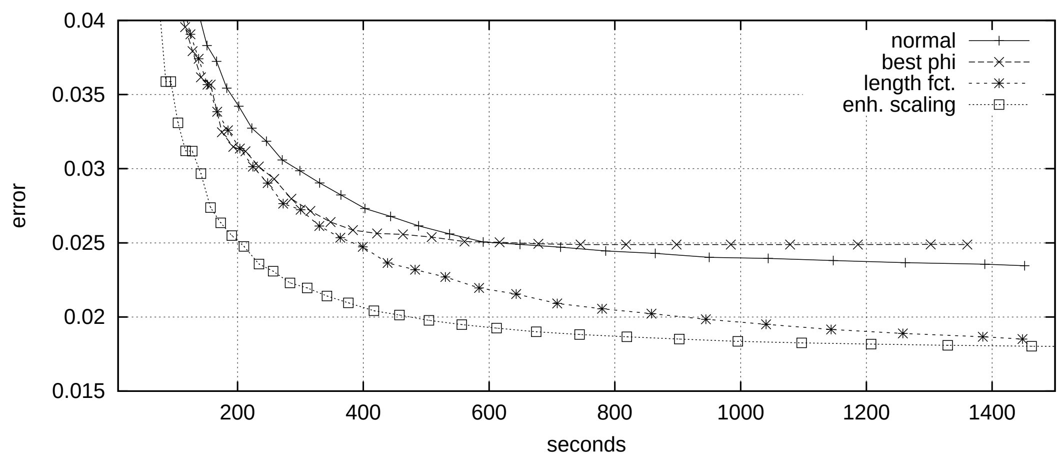
Beside these suggestions we present another practical improvement of the approximation algorithm, we call it *enhanced scaling*: During the algorithm we obtain for each commodity k a flow $h(o_k)$ and also a contribution to the Cut-Flow of $F(k)$. In order to construct a feasible flow, instead of scaling all $h(o_k)$ and $F(k)$ equally as it is done according to Lemma 4.6, we could also set up another optimization problem to find scalars $\lambda(k)$ that solve the following LP:

$$\begin{array}{lrcll} \text{Maximize} & \sum_k \lambda(k)F(k) & & & \\ \text{subject to} & \sum_k \lambda(k)(h_1(o_k,e)+h_2(o_k,e)) & \leq & f_e & \forall\, e \in E \\ & \lambda(k) & \geq & 0 & \forall\, k \end{array}$$

Like that, the bound obtained can be improved in practice. However, this gain has to be paid for by an additional computational effort that, in theory, dominates the overall running time. So the question how often we use this scaling is a trade off. Experiments have shown that using this scaling after every 100th iteration is a good choice. Notice that we use this scaling only in order to get a feasible solution value; the primal solution which is used while the algorithm goes on is not changed. Indeed, we have also tried to take over the scaled solution into the further flow of the algorithm, but this variant performs quite bad.

Figure 4.1 shows the effect of the three different improvements. In this example the VarMC-bound for the bisection-width of the DeBruijn graph of dimension 8 was computed and $\varepsilon = 0.1$ was used. We have carried out similar tests with a couple of different settings, so we can say that the behavior in Figure 4.1 is typical. It shows the error which comes from the difference of the actual primal and dual solution of the algorithm depending of the running time. For the application in a branch&bound algorithm we can stop the calculation as soon as the primal or dual value reaches a specific threshold, so the quality over the whole running time is of interest.

Figure 4.1: Comparison of different practical improvements of the approximation Algorithm



The main result is that the improvement of the enhanced scaling, as it is described above, generally gives the best results over the total run of the algorithm. We have also carried out experiments with all different combinations of the three improvements. But no combination is as good as the variant with enhanced scaling only is. Therefore, in all following experiments we use this variant. By the way, the results which are presented in Figure 4.1 also show, that the real error $\hat{\varepsilon}$ of the bound which is computed with the approximation algorithm is much smaller than the ε we have entered into the algorithm.

4.4 Experimental Evaluations

In this section we present experimental results regarding the different methods for computing the MC-bounds. Firstly, we will compare the different settings for the cost-decomposition approaches. Secondly, we will present the results of the experiments that we have carried out in order to quantify the best ε for the approximation algorithm when it is used inside the branch&bound algorithm. And thirdly, we show comparisons of the different methods.

Most of the experiments already use the branch&bound procedure we have developed for computing exact solutions of graph partitioning algorithms. Details on this branch&bound procedure are given in Chapter 5. Furthermore, the experiments concentrate on the bisection problem, since firstly, the problem is the most important one and secondly, the results regarding the comparison of the three different methods are typical for all graph partitioning problems.

Table 4.4: Average running times in seconds and average number of search nodes using cost-decomposition. (1): max-cut-flow-formulation, (2): min-congestion-formulation

		pure subgr.		Crowder		mod. CFM		Volume	
	graph	time	subp.	time	subp.	time	subp.	time	subp.
(1)	RandPlan	11318	85	**6626**	85	24986	85	7752	85
	RandReg	1192	38	589	28	737	29	**568**	32
	Random	748	117	**694**	119	722	116	849	127
	RandW	151	11	**128**	11	134	10	166	17
(2)	RandPlan	2032	85	**1276**	85	1831	85	1307	85
	RandReg	**3768**	64	17320	299	4374	72	17633	292
	Random	**1776**	160	3366	239	1941	169	6211	418
	RandW	1710	183	**1394**	139	1591	166	3661	588

4.4.1 Cost-Decomposition

In Section 4.2 we have presented cost-decomposition based methods for computing the VarMC-bound. There we have presented the use of Lagrangian relaxation for the generation of new columns. This relaxation can be done with a max-Cut-Flow-formulation or a min-congestion-formulation. Furthermore, four different methods for the computation of a new direction inside the subgradient algorithm have been presented: pure subgradient, Crowder rule, the modified Camerini-Fratta-Maffioli rule and the Volume algorithm.

Table 4.4 summarizes the results of the experiments which we have carried out in order to compare all combination of the different possibilities. Each given value is the average of the results with 20 generated graphs. The details on the experiments are given in Tables A.19-A.26. The bold-faced entry in every row is the one with minimal running time. In order to concentrate on the effect of the different cost-decomposition possibilities, no variable fixing is used inside the branch&bound procedure (see Chapter 5 for more details).

Results

Both, the running times and the sizes of the search-trees produced by the various subgradient algorithms and Lagrangian formulations differ considerably on the different benchmark sets. Thus, it is not an easy task to draw valid conclusions out of these experiments. As a tendency, the max-Cut-Flow formulation looks better than the min-congestion formulation (except for the random planar graphs). And when using the max-Cut-Flow formulation, the Crowder rule gives a good overall performance.

Table 4.5: Times and sizes of the search-trees of the branch&bound algorithm using the approximation algorithm with different ε's. (1): without variable fixing, (2): with variable fixing.

		$\varepsilon = 0.025$		$\varepsilon = 0.05$		$\varepsilon = 0.1$		$\varepsilon = 0.25$		$\varepsilon = 0.5$		$\varepsilon = 0.75$	
	graph	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.
(1)	RandPlan	6395	461	2158	461	962	463	587	557	**450**	1466	562	5273
	RandReg	2192	22	1107	23	561	26	196	37	**126**	90	163	489
	Random	2620	113	525	114	228	118	98	139	**80**	280	186	2188
	RandW	1383	37	472	46	176	62	**76**	150	85	788	1289	3859
(2)	RandPlan	3013	412	1009	406	587	381	133	239	54	117	**35**	78
	RandReg	2186	22	1083	23	622	25	181	34	**117**	67	160	173
	Random	2249	107	465	108	209	111	83	121	49	164	**47**	328
	RandW	737	14	283	17	122	25	44	54	**35**	188	41	582

4.4.2 Approximation Algorithm

The best choice of ε for the use in a branch&bound environment is a trade-off. If ε is too large, the computed bound is too bad, and the number of subproblems in the search-tree explodes. On the other hand, if ε is chosen too small, the computation of the bound is too time consuming.

Table 4.5 shows the resulting running times and the number of subproblems of the branch&bound algorithm using approximated bounds. The results are the averages on all 20 instances for every set of graphs. The results without variable fixing show the expected behavior for the choice of ε. The smaller ε is, the smaller is the number of search nodes. This rule is not strict when using variable-fixing: looking at the random planar graphs, we see that less good solutions of the VarMC-bound can result in stronger variable fixings so that the number of subproblems may even decrease. The table also shows that the effects of variable fixing are different for the different classes of graphs and also for changing ε's. Altogether, the experiments show that setting ε to 0.5 only is favorable.

4.4.3 Comparison of the different Methods

Finally, we give a comparison of the results of the branch&bound algorithm with variable fixing using the following three different methods for computing the VarMC-bound:

1. Standard barrier LP-solver (CPLEX, Version 7.0)
2. The approximation algorithm with $\varepsilon = 0.5$ and enhanced scaling
3. The cost-decomposition routine using the max-Cut-Flow formulation and the Crowder rule.

Table 4.6: Average running times (seconds) and sizes of the search-trees using the different methods for computing the VarMC-bound.

	CPlex		Approx.		Cost-Dec.	
graph	time	subp.	time	subp.	time	subp.
RandPlan	74	7	**54**	117	889	26
RandReg	993	21	**117**	67	557	28
Random	350	99	**49**	164	612	106
RandW	**30**	9	35	188	120	11

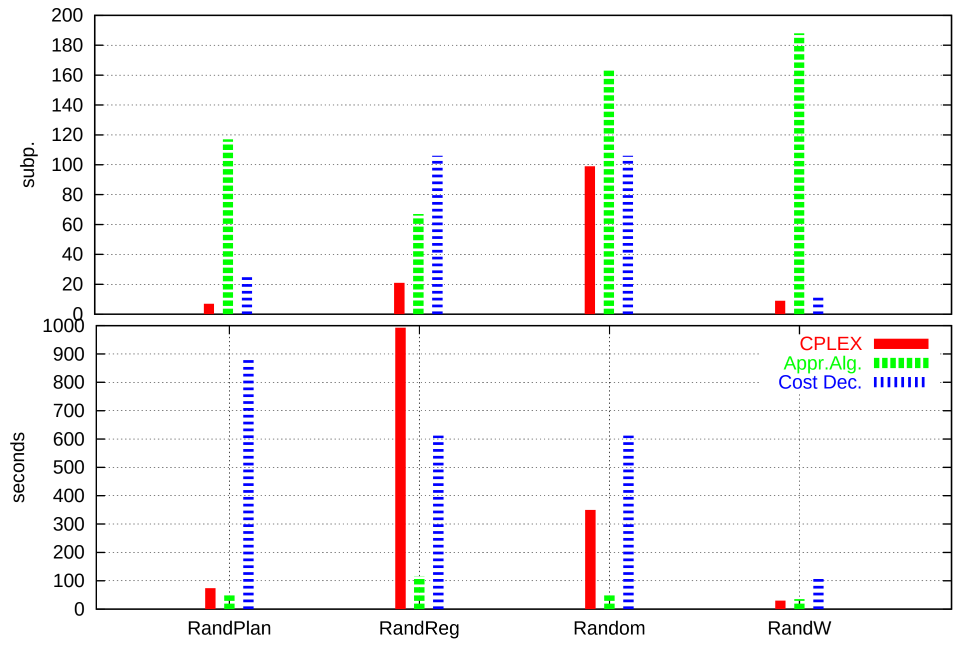


Table 4.6 shows the results on the four different benchmark sets. In general, the approximation algorithm with the enhanced scaling gives the best running times, even though the search-trees are largest. We also experimented with primal and dual simplex algorithms, but they take much more running time and cannot compete with the three approaches presented here.

Note that, with respect to the memory consumption, approximation and cost-decomposition are preferable to the barrier algorithm: When the graphs we consider become larger, for DeBruijn 9 of Shuffle-Exchange 9, for example, 2 GB main memory are not enough to allow the application of CPLEX. Thus, it cannot be applied to compute the corresponding bisection width, whereas both cost-decomposition and approximation allowed us to proof a bisection width of 92 and 48, respectively.

Table 4.7: Comparison of the different VarMC bounds with the semi-definite bound

Graph	CPLEX		Decomp.		Approx.		pSDP	
	Bound	Time	Bound	Time	Bound	Time	Bound	Time
Grid 10x11	11.00	13	11.00	54	10.59	2	8.84	51
Torus 10x11	20.17	12	20.17	48	19.66	2	17.33	47
SE 8	26.15	485	25.69	244	24.94	16	18.14	943
DB 8	49.54	613	49.05	322	46.95	21	35.08	1031
BCR m8	7.00	22	7.00	64	6.96	5	5.98	80

Moreover, recall from the formulation of the the master problem in Section 4.2, that the cost-decomposition approach was especially designed to be memory efficient, which makes it less competitive with respect to the running time. In any case, apart from being more memory efficient, we observe that cost-decomposition does not allow us to solve a generalized maximum multicommodity flow problem much faster than standard LP methods.

Finally, we present a comparison of the different methods for computing the VarMC-Bound with a lower bound on the graph bisection problem based on semi-definite programming. In Table 4.7 the results are presented. The results show the power of the approximation algorithm in the application of graph bisection. Its bounds are nearly as good as the optimal bounds of the VarMC method computed with CPLEX and much better then the semi-definite bounds. And the running times are clearly the smallest ones of the four compared methods.

4.4.4 Summary

To sum up the results of the experiments, we feel the following main conclusions can be drawn:

- The decomposition-approach gives nearly the same bounds as the bounds which are computed with CPLEX. The computation time of the decomposition-approach is longer on small instances and shorter on large instances.
- Surprisingly large ε's have to be used in order to get the best branch&bound performance when the approximation algorithm is used.
- If those ε's are used, the computed bounds are visibly smaller than the optimal bounds. However, the computation times are clearly shorter when compared to CPLEX.
- Both alternatives to CPLEX are clearly less memory-consuming than CPLEX.

Chapter 5

Branch & Bound Algorithm

In this chapter we present the algorithm for computing exact solutions of graph partitioning problems. This algorithm is a branch&bound algorithm which uses the presented MC-bounds for the bounding of subproblems. We assume that the reader is familiar with the concepts for branch&bound; for introduction we refer to [Mit70, Kum87, Iba88].

In implementing a branch&bound procedure we must make many design decisions. These design decisions include questions like:

- How can we obtain a good upper bound, i.e. a small feasible value for the cut-size?
- Which subproblem is computed next?
- How problems are divided into subproblems?
- How lower bounds on subproblems are computed, such that the lower bounds utilize the restrictions of the branchings?
- Which branching is selected?
- How can subproblems we further restricted without losing a possibly best solution?

In the following sections we will describe the design decisions we have made. In Section 5.1 we will present methods for calculation very good feasible partitions. In Section 5.2 we will introduce the technique that we have used for generating subproblems and computing good lower bounds on these restricted subproblems. In Section 5.3 we illustrate very effective methods which further restrict subproblems without losing possibly optimal solutions. And in Section 5.4 we explain a procedure which exactly decides which parts of the solutions are fixed in order to get subproblems.

5.1 Upper Bound and Depth-First-Search

The task of our branch&bound algorithm is to compute the optimal cut-size of the given graph partitioning problem. For the efficiency of every branch&bound algorithm it is crucial that there are very good upper bounds. The best what could happen is that the initial upper bound corresponds to the optimal cut-size. In this case the branch&bound “only” has to prove that the upper bound is indeed optimal.

Fortunately, there exists a large number of algorithms for computing good solutions of graph partitioning problems. These algorithms are very fast and are often designed such that they can handle graphs with thousand and millions of vertices. In contrast to this we can only look at graphs with at most some hundreds of vertices, since the problem to compute the exact cut-size is theoretically NP-hard and also practically very difficult. Therefore, we can use different heuristics for computing an upper bound at the beginning of the branch&bound algorithm. Doing this the upper bound we got from the heuristical algorithms are optimal in nearly all cases. We use the PARTY-library [Pre98, PD96] for the computation of the upper bound.

The goodness of the heuristical solution implies that we do not have to think on the order we use to work on the open subproblems. Therefore we can simply use depth-first-search. This simplifies the code and above all this drastically decreases the number of subproblems we have to store.

5.2 Realization of Branchings

A branch inside a branch&bound procedure means that the solution-space, which is represented by the actual search-node, is divided into at least two parts. These parts then represent new subproblems. It is crucial for the efficiency of the whole approach that the lower bounds which are used can take an advantage out of the restriction of the solution-space.

In our branch&bound algorithm we divide the solution-space of a search-node such that we select a pair of vertices and in the one newly generated subproblem we fix this pair of vertices to be in the same partition and in the other newly generated subproblem we fix this pair to be in different partitions. Therefore, every search-node which cannot be pruned because the lower bound is not large enough is divided into two new search-nodes. In the following we name the fixing of a pair of vertices to be in the same partition a *join* while the fixing of a pair to be in different partitions a *split*.

Utilization of a Join

In order to utilize a join of two vertices v_1 and v_2 for the computation of the lower bound we could insert a new edge $\{v_1, v_2\}$ into the graph with an infinite capacity.

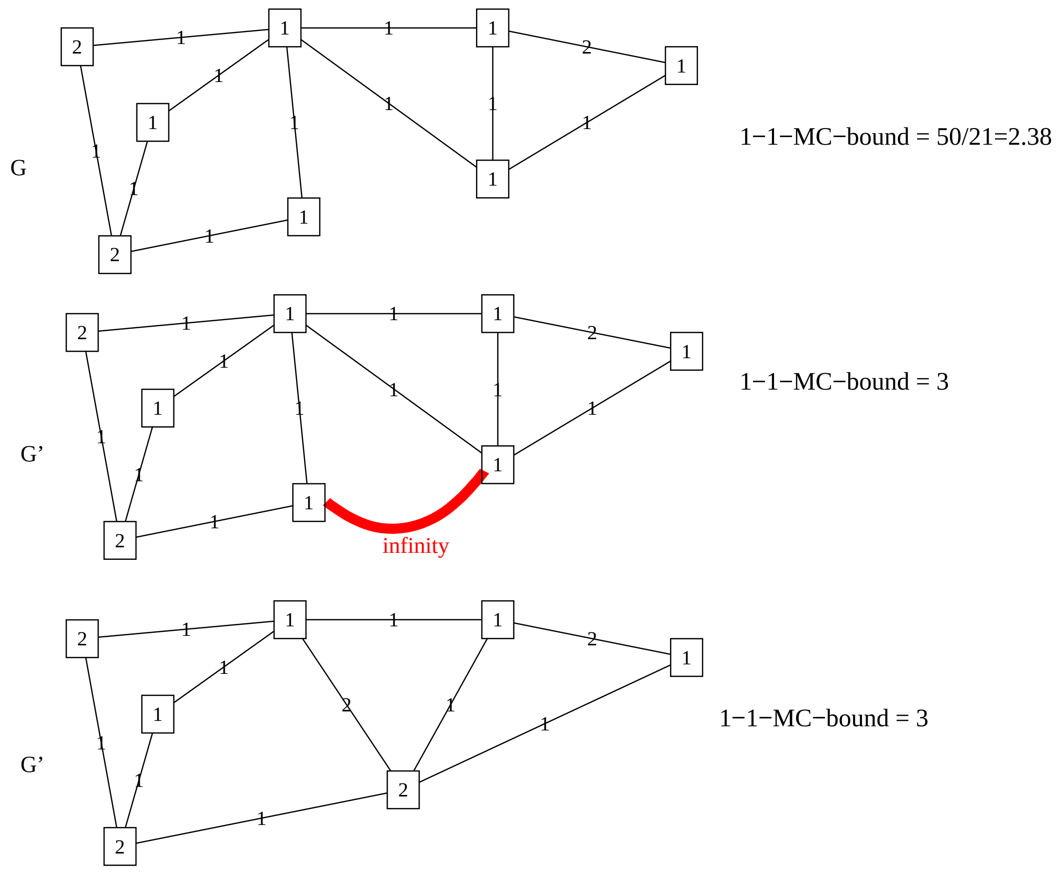


Figure 5.1: Example of a join of two vertices when computing the 1-1-MC-bound on the bisection width of a graph with vertex- and edge-weights

Obviously, since we restrict the search-space such that this pair of vertices are in the same partition, this does not change the cut-size. Furthermore, the insertion of this new edge probably increases the lower bound since this new edge can be used as an abbreviation which does not cause any costs for the flows inside the graph. In order to simplify the computation of the MC-bounds, we could left out the capacity-constraint of this inserted edge. However, we can even do something better to compute the MC-bounds for this subproblem:

It can be seen that inserting an edge with an infinite capacity gives the same MC-bound as if we modify the graph such that the two joined vertices are indeed joined into one vertex of the graph. I.e. let $G = (V, E, g, f)$ be the graph and we look at a join of vertices $v_1, v_2 \in V$. Then we construct a graph $G' = (V', E', g', f')$ with

$$\begin{array}{lrcl}
 & V' & = & V \setminus \{v_2\} \\
\wedge & \forall w \in V' : g'(w) & = & \begin{cases} g(w) & \text{if } w \neq v_1 \\ g(v_1) + g(v_2) & \text{else} \end{cases} \\
\wedge & E' & = & (E \setminus \{\{v_2, u\} \in E\}) \cup \{\{v_1, u\} | \{v_2, u\} \in E\} \\
\wedge & \forall \{u, w\} \in E' : f'(\{u, w\}) & = & \begin{cases} f(\{u, w\}) & \text{if } u \neq v_1 \wedge w \neq v_1 \\ f(\{v_1, u\}) + f(\{v_2, u\}) & \text{if } w = v_1 \vee w = v_2 \\ f(\{v_1, w\}) + f(\{v_2, w\}) & \text{if } u = v_1 \vee u = v_2. \end{cases}
\end{array}$$

Figure 5.1 gives an example of this processing for a small graph and demonstrates the possibility of an increment of the bound. Without the join, there is a bottle-neck inside the graph between the three rightmost vertices and the other vertices. Including the edge with infinite capacity or joining the corresponding vertices dissolves the bottleneck.

Utilization of a Split

Removing Split-Edges. In order to utilize a split of vertices $v_1, v_2 \in V$ for the computation of the lower bound we could remove the edge $\{v_1, v_2\}$ from the graph if $\{v_1, v_2\} \in E$. Then we can compute the bound on the remaining graph and add $f_{\{v_1, v_2\}}$ to the bound. Unfortunately, if we use this method there is no increase of the bound if the vertices v_1 and v_2 are not adjacent. In the following we present two other methods for the utilization of a split. On the one hand, these methods can increase the bound even if the vertices are not adjacent and on the other hand, these methods can possibly increase the bound more than just removing the edge.

Split-Commodities. The first thing we can do is based on the ideas of the MC-bounds. One main idea of these bound was the consideration that we send commodities through the graph and know that a specific portion of these commodities has to cross every possible feasible cut. Now, we are looking at a subproblem where we only want to consider solutions where the pair of vertices v_1 and v_2 are in different partitions. This means that all commodities which are routed from vertex v_1 to vertex v_2 have to cross the cut! Hence, we can include such a commodity of variable size s_{cut} into our MC-bounds. This commodity has the origin v_1 and the destination v_2 and it can totally be counted into the Cut-Flow. Let $P \subset V^2$ be a set of pairs of vertices which are split in the current subproblem. Then the linear program for the VarMC-bound as it has been given in Section 4.1 is expanded to

$$
\begin{array}{lrl}
\multicolumn{3}{l}{\text{maximize } 2\sum_{p\in P} s(p) + (N-M)\cdot \sum_v s(v)} \\
\multicolumn{3}{l}{\text{subject to}} \\
\multicolumn{3}{l}{\forall c,d \in V, c \neq d, \{c,d\} \notin P:} \\
 & s(c)\cdot g(d) - \sum_{\{v,d\}\in E} h(c,v,d) - h(c,d,v) & = 0 \\
\multicolumn{3}{l}{\forall c,d \in V, c \neq d, \{c,d\} \in P:} \\
 & s(\{c,d\}) + s(c)\cdot g(d) - \sum_{\{v,d\}\in E} h(c,v,d) - h(c,d,v) & = 0 \\
 & & \\
\forall \{v,w\} \in E: & \sum_{c\in V} h(c,v,w) + h(c,w,v) & \leq f(\{v,w\}).
\end{array}
$$

According expansions can be done for the 1-1-MC-bound and the MVarMC-bound. Furthermore, these expansions are obviously linear programs and more important, they are covered by the bound model for the approximation algorithm in Section 4.3.

Obviously, the use of these split-commodities is always at least as good as not using them, since the linear program always has the freedom of selecting $\forall p \in P : s(p) = 0$ and the standard MC-bounds are achieved. Furthermore, by using split-commodities we can show that not removing split-edges is always at least as good as removing split-edges:

Theorem 5.1
When split-commodities are used, the MC-bounds with retained split-edges are at least as good as the MC-bounds with removed split-edges. This holds for the general graph partitioning problem and therefore this also holds for the specialized versions.

Proof:
We look at an instance L of an MC-bound with maximal lower bound Bd where we have removed a split-edge e with weight f_e from the graph. Then we know that

$$Bd = \frac{CF}{C} + f_e$$

holds, where CF and C are the Cut-Flow and congestion, respectively. Now we look at possible MC-instances where edge e is not removed. Obviously, we can take over the flows of instance L and get the same Cut-Flow CF and congestion C. Furthermore, these flows do not use edge e, so we can increase the split-commodity of the corresponding split-pair by a value of $C \cdot f_e$ without increasing the total congestion C. The MC-instance which is constructed in this way has bound Bd' with

$$Bd' = \frac{CF + C\cdot f_e}{C} = \frac{CF}{C} + f_e = Bd$$

and the theorem is proved. ■

Decreasing Set of Standard-Destinations. The second thing we can do is restricted to graph partitioning problems with $k = 2$. Then it follows that if we look at a split pair $\{v, w\}$ and any vertex $u \in V$, exactly one of the vertices v, w must be in the same partition as u and exactly one must be in the other partition. Normally when calculating the Cut-Flow, we apply the consideration that if a vertex sends commodities of unit size to all other vertices, at least $n - M$ of these commodities has to cross any cut. Now, we also know that if a vertex u sends commodities of unit size to all other vertices except vertices v, w, at least $n - M - 1$ of these commodities has to cross any cut. Notice that even though the number of destinations is decreased by two, the commodities which are counted into the Cut-Flow are decreased by only one. The following linear program corresponds to the VarMC-bound with Split-Commodities and the decreased set of standard-destinations:

$$
\begin{array}{lrl}
\text{maximize } 2\sum_{p\in P} s(p) + (N - M - |P|)\cdot\sum_{v\in V} s(v) & & \\
\text{subject to} & & \\
\forall c,d \in V \text{ with } c \neq d \wedge \forall p \in P : d \notin p : & & \\
 & s(c)\cdot g(d) - \sum_{\{v,d\}\in E} h(c,v,d) - h(c,d,v) & = 0 \\
\forall c,d \in V \text{ with } \{c,d\} \in P : & & \\
 & s(\{c,d\}) - \sum_{\{v,d\}\in E} h(c,v,d) - h(c,d,v) & = 0 \\
 & & \\
\forall \{v,w\} \in E : & \sum_{c\in V} h(c,v,w) + h(c,w,v) & \leq f(\{v,w\}).
\end{array}
$$

The same adaptations can be applied to the 1-1-MC-bound and the MVarMC-bound. Again, we can show that the use of these decreased sets of destinations delivers bounds which are at least as good as the normal sets:

Theorem 5.2
When split-commodities are used and when we look at graph partitioning problems with $k = 2$, the MC-bounds with decreased sets of destinations as described above are at least as good as the MC-bounds without this decrease.

Proof:
We are going to prove the theorem of the VarMC-bound, but all the considerations used can also be applied to the 1-1-MC-bound and MVarMC-bound. We look at an instance L of an MC-bound with maximal lower bound Bd where the set of destinations is not decreased. Then we know that

$$Bd = \frac{CF}{C}$$

holds, where CF and C are the Cut-Flow and congestion, respectively. Now we look at possible MC-instances where the set of destinations are decreased. We can take over the flows of instance L if we omit all flows from all vertices u to all split pairs. Then we get the same congestion C while the Cut-Flow is decreased to

$$CF' = CF - |P| \sum_{v\in V} s(v).$$

Furthermore, there is some space left on the edges since we have omitted all flows to split pairs. To be more precise, we have omitted flows from all sources $u \in V$ to all split pairs $\{v,w\} \in P$. So, we can increase the Split-Commodities $s(\{v,w\})' = s(\{v,w\}) + \frac{1}{2}\sum_{u \in V} s(u)$ and use the space left on the edges for these commodities. Then we get

$$CF' = CF - |P| \sum_{v \in V} s(v) + 2 \cdot |P| \cdot \frac{1}{2} \sum_{u \in V} s(u) = CF$$

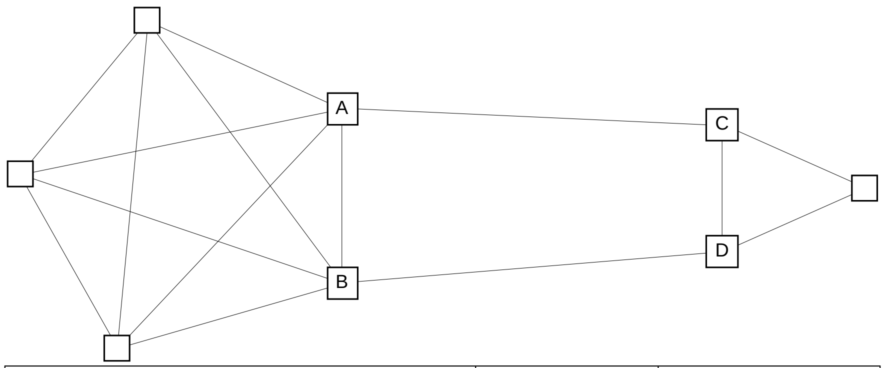
which proves the theorem. ■

For simplification purposes, we have not considered vertex-weights in the above calculations. If we take vertex-weights into consideration, the situation becomes more complex. Then, for any split-pair $\{v,w\} \in P$ we have to determine $gp(\{v,w\}) := \min\{g(v), g(w)\}$. The standard set of destinations is decreased such that for any $\{v,w\} \in P$, the vertex v with $g(v) = gp(\{v,w\})$ is totally removed from the set of destinations while for the other vertex w the portion of the normal commodity to this vertex is decreased from $s(c) \cdot g(w)$ to $s(c) \cdot (g(w) - gp(\{v,w\})$. Furthermore, the factor of the normal commodities inside the Cut-Flow has to be changed from $N - M - |P|$ to $N - M - \sum_{p \in P} gp(p)$. If all vertices have weight one, this procedure results in the same bound-computation as described above. The proof of Theorem 5.2 can therefore be adapted.

Summary. We have presented three different possibilities for the utilization of a split in order to increase the MC-bounds. The most effective possibility is the introduction of split-commodities. It is quite obvious that these split-commodities increase the bounds if there is no bottle-neck in the graph between the split vertices. Furthermore, if split-commodities are used we have shown that the simplest possibility which involves removing the split-edges from the graph and adding them to the bound after calculating the flows is not as effective as keeping the split-edges in the graph. Finally, if we look at graph partitioning problems with $k = 2$ we have presented the possibility of decreasing the set of destinations which is, if combined with the split-commodities, always at least as good as not decreasing them.

At last, we present an example which demonstrates that keeping split-edges in the graph can increase the VarMC-bound and that decreasing the set of destinations increases the VarMC-bound even more. In Figure 5.2 such a graph with its VarMC-bounds is illustrated. The VarMC-bound on this graph without any splits is $2\frac{2}{3}$. If split $\{A,C\}$ is introduced we see that removing the split-edge decreases the bound, keeping the split-edge does not change the bound and decreasing the sets of destinations increases the bound. This is typical behavior if there is a bottle-neck between the split edges. In contrast to this, if we split vertices $\{A,B\}$ there is no bottle-neck and the bound grows to the exact cut-size which is reached by using only the split-commodity.

Figure 5.2: Example of a graph where keeping split-edges and decreasing the sets of destinations improves the VarMC-bound for the bisection problem



situation	VarMC-Bd.	reached by
without any split	$\frac{8}{3} = 2\frac{2}{3}$	$s(A) = s(B) = 1$
split A,C and remove split-edge	$\frac{4}{3} + 1 = 2\frac{1}{3}$	$s(B) = 1$
split A,C and keep split-edge	$\frac{8}{3} = 2\frac{2}{3}$	$s(A) = s(B) = 1$
split A,C, keep split-edge and decrease sets of destinations	$\frac{6}{2} = 3$	$s(A) = s(B) = 1$
split A,B, keep split-edge	$\frac{5}{1} = 5$	$s(\{A,B\}) = 5$

5.3 Variable Fixing

In this section we will present some techniques which can be used inside the branch&bound search in order to restrict the search-space without losing optimal solutions. I.e. there are situations where a join or split of a pair of vertices leads to infeasible or non-optimal solutions. Then we can split or join this pair respectively without looking at the other branch. Firstly, we will present simple situations which would lead to infeasible solutions. Secondly, we will present methods which are based on the computed MC-bounds and which can show that specific splits cannot lead to optimal solutions.

5.3.1 Simple Considerations

There are some situations where a simple consideration shows that a pair of vertices has to be split or joined:

- If $\exists v, w \in V$ with $g(v) + g(w) > M$ then we can split v and w, as otherwise a partition which includes both vertex v and w violates the given maximal size of the partitions.

- If $\exists\{v,u_1\},\ldots,\{v,u_l\}\in P$ and $w\in V$ with $g(w)+\sum_{i=1}^{l}g(u_i)>(k-1)\cdot M$ we can join v and w, as otherwise vertex v has no possibility of being part of a partition with the minimal partition size. The minimal partition size M_{min} is not explicitly given in the graph partitioning problems but it follows from the given data with $M_{min}=N-(k-1)\cdot M$.
- If $k=2$ and $\exists v,u,w\in V$ with $\{v,w\}\in P$ and $\{u,w\}\in P$ then we can join v and u. This rule is obvious as otherwise it is impossible to partition the set of vertices into two sets with respect to the set of splits. Furthermore when this rule is used, it follows that there is at most one split for every vertex.
- The rule above can be generalized to arbitrary k's: If $\exists V'\subset V$ with $|V'|=k+1$ and $\forall 1\le i<j\le k+1:\ \big(\{v_i,v_j\}\in P\vee(i=1\wedge j=2)\big)$ with $V'=\{v_1,\ldots,v_{k+1}\}$, then we can join the vertices v_1 and v_2. Otherwise, we would have $k+1$ vertices where every pair is split, so we would not be able to generate a partition of the graph into k parts without violating at least one of the splits.

All these rules can be checked using a very small amount of computational power in comparison to the calculation of the MC-bounds. So in any node of the search-tree we check the rules and perform joins or splits accordingly.

5.3.2 MC-bounds based Methods

Here we present more powerful methods which are based on the computed MC-bound of the actual sub-problem. These methods utilize the fact that there is always a threshold L inside the branch&bound procedure and we are only interested in solutions with a cut-size of at most L. The methods detect the fact that a split of a pair of vertices would result in solutions with a cut-size greater than L. So, all those pairs of vertices can be joined without losing optimal solutions.

Edge based

The first method is more simple and is only based on edges. The following theorem outlines its basis:

Theorem 5.3
Let B be the MC-bound of the actual subproblem, C its congestion and $c(e)$ the congestion of the edges. Then, if vertices v,w are split with $\{v,w\}=e\in E$ it holds that any solution has a cut-size of at least

$$B+f_e\left(1-\frac{c(e)}{C}\right).$$

Proof:
We prove the theorem by constructing an MC-instance which gives a lower bound of $B + f_e(1 - \frac{c(e)}{C})$. Let I be the MC-instance which gives bound B on the actual subproblem. We take over instance I and add a flow of size $f_e(C - c(e))$ on edge e, this flow is counted as a split-commodity as we have presented it in Section 5.2. This addition of the flow results in a congestion of C on edge e. Therefore, this constructed instance has congestion C and the Cut-Flow CF is increased by $f_e(C - c(e))$ in comparison to instance I. Therefore, the bound B' of the constructed instance is

$$B' = \frac{CF + f_e(C - c(e))}{C} = B + f_e(1 - \frac{c(e)}{C})$$

which proves the theorem. ∎

The method for fixing a join is now obvious: We check if there is any edge $e = \{v, w\}$ with $B + f_e(1 - \frac{c(e)}{C}) > L$. If such an edge exists, we can join the vertices v, w without losing any solution with cut-size of at most L. We do this check after every bound-computation. The running time for this is negligible. If the calculated MC-bound B is very close to the threshold L, then the method is very strong. For example, in the extreme case of $B = L$, every edge e which is not totally loaded (i.e. $c(e) < C$) produces a join of the adjacent vertices.

Max-Flow based

The ideas behind the edge-based method above can be expanded to a method which not only looks at single edges but which computes the maximal flow between two vertices with reference to the space which is left on the edges while computing the MC-bound. The following theorem outlines the basis of this method:

Theorem 5.4
Let B be the MC-bound of the actual subproblem of a graph $G = (V, E, g, f)$, C its congestion and $c(e)$ the congestion of the edges. Let $G' = (V, E, g, f')$ be a graph with $f'(e) = f_e(1 - \frac{c(e)}{C})$ and $mf(v, w)$ be the maximal flow between the vertices v and w on graph G'. Then, if vertices v, w are split, it holds that any solution has a cut-size of at least

$$B + mf(v, w).$$

Proof:
We expand the MC-instance with bound B on the original subproblem by adding a flow of total size $C \cdot \mathrm{mf}(v, w)$ from vertex v to vertex w. Since this flow can be realized using only the space left on the edges, this can be done without increasing the congestion. The added flow is counted as split-commodity and so the Cut-Flow is increased by

Table 5.1: Examples of the effect of the MC-bound based variable fixing

graph	*B*	*bw*	without		edge-based		Max-Flow	
			time	subp.	time	subp.	time	subp.
BCR-6x10g	26.2	28	84	93	3	5	3	5
BCR-9x6m	789.8	792	938	78	433	25	310	21
Random-50-0.2-3	63.6	71	529	237	466	211	437	197
RandW-32-10-0	922.5	944	134	67	127	53	96	35
RandPlan-100-1-1	30.6	32	464	120	45	11	23	6
RandRegular-100-4-3	27.7	30	64	7	64	7	65	7

$C \cdot \mathrm{mf}(v,w)$. Therefore, if a split $\{v,w\}$ was added to the subproblem, a lower bound of $B + \mathrm{mf}(v,w)$ holds which therefore proves the theorem. ■

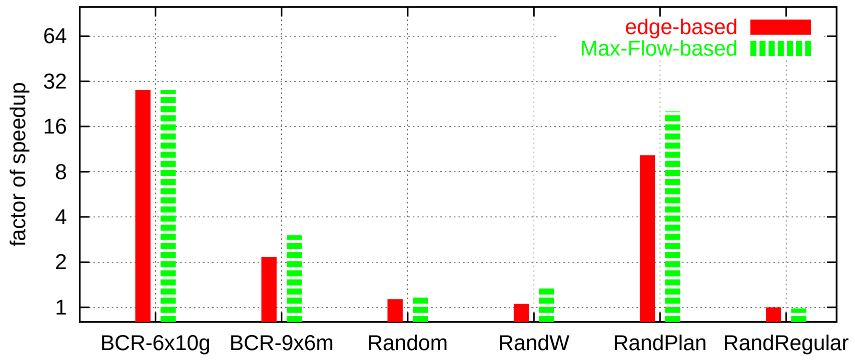
Now, the method for determining pairs of vertices which can be joined is obvious: We compute the max-flow values for all pairs of vertices on the graph G' and check if $B + \mathrm{mf}(v,w) > L$. However in contrast to the edge-based method, the computation of all $\binom{n}{2}$ max-flow values requires a notable amount of computational power. From the well known MaxFlow-MinCut-Theorem it is known that the maximal flow between two vertices is identical to a minimal cut which divides the two vertices. So we have to compute the minimal cut for all pairs of vertices of a graph. This problem was firstly introduced in 1961 by Gomory and Hu [GH61]. They called it the *cut-tree* problem, as the results for all pairs can be presented in a tree. Goldberg and Tsioutsiouliklis present a recent survey on different implementations of the cut-tree problem in [GT01]. We use an implementation which was introduced by Gusfield [Gus90].

Summary

We have presented two methods for the fixing of joins which are based on the computed MC-bound and which utilize the space left on the edges. Both are very effective if the computed bounds are very close to the actual threshold L. Of course, their success depends highly on the actual graph and MC-bound. Table 5.1 gives examples of the success of the methods for several graphs and the graph bisection problem, Figure 5.3 illustrates these results. The examples were computed with the MVarMC-bound using the approximation algorithm of Section 4.3 with $\varepsilon = 0.5$ and with the enhanced scaling. The column B gives the bound on the root problem, *bw* gives the exact bisection-width, *time* gives the total time of the branch&bound algorithm in seconds and *subp.* gives the number of nodes of the branch&bound-tree.

It is obvious that the effect of the variable fixing is drastic on some graphs, e.g. for the “BCR-6x10g” and the “RandPlan-100-1-1” graphs. On the other hand there are some

Figure 5.3: Visualization of the results of Table 5.1



graphs where the variable fixing has almost no influence. e.g. the “RandRegular-100-4-3” graph. Furthermore, the examples show that the edge-based method already gives good results. When compared to the step from no variable fixing to edge-based method, the step from edge-based method to the max-flow method improves the effect only slightly. But nevertheless, the max-flow based method gives the best results.

5.4 Branching Selection

One crucial aspect of every branch&bound algorithm is the branching: if a node of the search tree cannot be bounded, the corresponding feasible region has to be divided into at least two regions which become son-nodes of the original node in the search-tree. We do this branching by fixing the relation of a pair of vertices: two vertices have to be in the same partition (join) or they have to be in different partitions (split). For the size of the search-tree and therefore for the running time of the branch&bound algorithm, it is crucial that the selected pair of vertices leads to search-nodes with improved lower bounds such that they can be bounded quickly. So, in order to make a good selection, we try to predict the value of the lower bound in the case of a split or join of pairs of vertices respectively. Using these predictions, we finally have to select an appropriate pair. So in this section firstly we present methods of prediction and secondly we show how the selection can be carried out based on these predictions.

5.4.1 Prediction of the Impact of a Split on the Lower Bound

Firstly, we look at methods of predictions of the lower bound in the case of a split of a pair of vertices. We firstly give different ideas for heuristical values which could be used in our predictions and then we show their effectiveness.

Ideas for Values which can be used for Prediction

We have tested a number of different possible values of prediction. The most interesting or successful ones are the following:

Distance: A first obvious idea involves the use of the *distance* of two vertices as a method of prediction. It could be expected that the smaller the distance of two vertices, the bigger the increase of the lower bound if these two vertices are split. In order to consider edge-weights $w(e) : E \to \mathbb{N}$ of the given graph, we use edge lengths $l(e) := 1/w(e)$ for the calculation of the distance. This follows from the idea that an edge with a big weight can transport a lot of commodities so, as regards a flow problem, the two adjacent vertices have a small distance.

MaxFlow: A second more sophisticated idea involves the use of the *MaxFlow* of two vertices: the bigger the MaxFlow, the more commodities can be routed from one of the two split vertices to the other. And since in the case of a split the total amount of this commodity can be counted into the Cut-Flow, this should increase the lower bound.

rel. MaxFlow: Both of the ideas above have the disadvantage that they do not consider the possible impact of the split-flow on the rest of the multicommodity flow. Clearly, in both cases it can occur that the idea of increasing the lower bound does not work as the edges are heavily loaded by other commodities. Therefore we suggest a third possible method of prediction: The idea is to take the multicommodity flow which is used for the computation of the lower bound of the actual search-node into consideration. Using the maximal congestion of the multicommodity flow of the computed lower bound, we get a “free” amount for each edge. The idea now is to use the MaxFlow value on the graph with edge-weights equal to the reciprocal of the free amount for each edge. Pairs of edges with a big MaxFlow on this graph should clearly increase the Lower Bound since a commodity can be sent between them without disturbing the other commodities. We call this method *relative MaxFlow*.

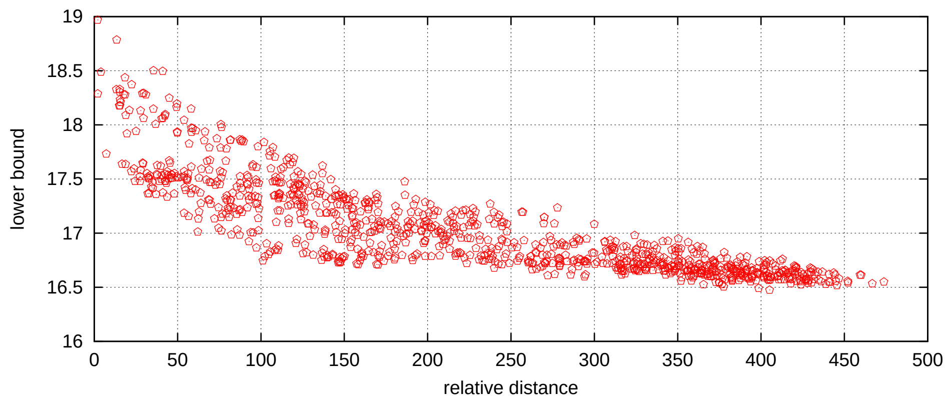
rel. distance: We can also use the idea of the relative MaxFlow for the calculation of the distance. So we call the distance of the vertices with regard to edge lengths which correspond to the reciprocal of the free amount of the computed multicommodity flow *relative distance* .

log(rel. distance): Finally, as you will see in the following experiments, very satisfactory results can be achieved if we use the logarithmic of the above rel. distance in order to achieve a linear correlation.

Experiments

In order to evaluate the effectiveness of the different values , we have carried out a number of experiments whose results are presented now. The main principle of the experiments is to take a set of pairs of vertices, compute the different values of prediction and then compute the lower bounds which would be reached if the actual pair of vertices were split. We use the MVarMC bound and the approximation algorithm of Section 4.3 for the computation of the lower bound. Figure 5.4 shows the results of 1000 randomly selected pairs of vertices of a DeBruijn graph of dimension 6 while using the relative distance as a method of prediction.

Figure 5.4: Predicting the lower bound of a split using the relative distance, with a DeBruijn graph of dimension 6 and 1000 randomly selected pairs of vertices



In this Figure we can see that the relative distance somehow correlates to the possible lower bound. However, for further comparison we need some type of measurement of this correlation. For the use of the prediction which follows it would be helpful if there was a linear correlation between the prediction value and the lower bound. So we use *Pearson's correlation coefficient* r as it is presented by Kenney and Keeping [KK54, p. 258] for this purpose. Given n pairs of x's and y's, r is defined as

$$r = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \overline{x}}{s_x}\right)\left(\frac{y_i - \overline{y}}{s_y}\right)$$

where $\overline{x}$ ($\overline{y}$) is the average of the x's (y's) and s_x (s_y) is the standard deviation. The value of r ranges from -1 to 1 . Large absolute values indicate strong linear correlation. (E.g. the correlation coefficient of the data in Figure 5.4 gives $r \approx -0.83$)

In order to consider all circumstances in the graph partitioning algorithm, we compare different situations: Firstly, we take bisectioning (p2) and four-partitioning (p4). Furthermore, we look at the situation of a graph at the root of the search-tree, a graph with

Table 5.2: Averages of the correlation coefficient of the different methods and graphs used to predict the impact of a split

	DB6	SE6	Grid	ex36	RPlan	Avg
dist.	-0.56	-0.57	-0.54	-0.18	-0.43	-0.46
MaxFlow	-0.02	-0.00	-0.01	-0.08	+0.13	+0.00
rel.MaxFlow	+0.50	+0.59	+0.73	-0.01	+0.73	+0.51
rel.dist.	-0.80	-0.81	-0.77	-0.38	-0.70	-0.69
log(rel.dist.)	-0.88	-0.91	-0.90	-0.41	-0.83	-0.79

5 randomly selected joins (J), with 5 randomly selected splits (S) and finally a graph with 5 randomly selected joins and splits (IN). In Table A.39 the detailed results of Pearson's correlation coefficient for 5 different graphs are given: the DeBruijn graph of dimension 6, the shuffle-exchange graph of dimension 6, an 6x10 grid, the ex36-graph and a random maximal planar graph with 60 vertices and 90% of edges. For each variation of the graph the predictor with the best correlation coefficient is printed in bold face. For better oversight the averages of the different methods and graphs are given in Table 5.2.

Some obvious conclusions can be drawn from the experiments:

- The distance already gives a good correlation
- The correlation of MaxFlow is very bad, whereas the correlation of the relative MaxFlow is better than that of the distance.
- The correlation of the relative distance is the best one and is even improved when the logarithmic is used.
- These rankings are nearly independent of the graph.
- The "ex36" graph is the most difficult one for prediction. But as you can see in the detailed results, the prediction becomes easier as we move down the search-tree (i.e. some joins and splits are performed).

Taking everything into account, the logarithmic of the relative distance is quite good predicting value for the lower bound, if a split is performed.

5.4.2 Prediction of the Impact of a Join on the Lower Bound

Here, we look at the methods of prediction of the lower bound in the case of a join of a pair of vertices. Again, firstly we give different ideas for heuristical values which could be used in our predictions and then we show their effectiveness.

Ideas for Values which can be used for Prediction

We have tested a number of different possible values of prediction. The most interesting or successful ones are the following:

Distance: As in the case of a split, we could use the distance between two vertices for the prediction of the lower bound. The bigger the distance, the bigger the increase of the lower bound in the case of a join of these two vertices. Edge-weights are considered in the same way as they are for split-prediction.

max11UB: Using the distances $d : V \times V \rightarrow \mathbb{R}_+$ of all pairs of vertices, a simple upper bound ub on the lower bound lb of the 1-1-MC-bound can be computed:

$$lb \leq ub = \frac{n^2/2}{\frac{1}{|E|}\sum_{v,w} d(v,w)}$$

The formula corresponds to the bisection problem without vertex- and edge-weights. The case of k-partitioning can be easily considered (use $\frac{k-1}{k}n^2$ instead of $n^2/2$). If vertex-weights are given, n has to be the sum of all vertex-weights and the distances have to be multiplied by the product of the two corresponding vertex-weights. Edge-weights can be considered such that the distance is calculated using the reciprocate of the edge-weights of each edge as its length (the same method which is used for the Distance above).
A value of prediction is computed for each pair of vertices a and b by computing the upper bound $ub(a,b)$ using distances which would be true if the two vertices were joined. The computation of $ub(a,b)$ for all $a,b \in V$ involves the calculations of all $\sum_{v,w} d_{a,b}(v,w)$ where $d_{a,b}(v,w)$ is the distance of the vertices v and w if the vertices a and b are joined. We do not have to compute all distances $d_{a,b}(v,w)$ totally new but can use

$$d_{a,b}(v,w) = \min\{d(v,w), d(v,a)+d(b,w), d(v,b)+d(a,w)\}.$$

Therefore in order to compute $ub(a,b)$ for all $a,b \in V$, we could compute all $d_{a,b}(v,w)$ which would result in an effort of $\Theta(n^4)$. Experiments show that this computational effort can be neglected if $n < 200$. However, if we look at graphs with 500 or 1000 vertices (DeBruijn 9 or 10), the computation of the $d_{a,b}(v,w)$'s dominate the overall running time of the branch&bound procedure. However, we can look at only those distances D' which differ, i.e. $D' = \{(a,b,v,w) | d(v,w) \neq d_{a,b}(v,w)\}$. Experiments show that we have $|D'| \approx n^3$. And in fact we can organize the computation of all the sums $\sum_{v,w} d_{a,b}(v,w)$ such that we need time $\Theta(|D'|)$.

rel. distance: Again, it is possible to utilize the multicommodity flow of the current search-node already calculated. We use the “free” amount of the edges as a reciprocal of the edge-length and compute the distances using these edge lengths (compare the predictors for split).

Table 5.3: Averages of the correlation coefficient for the different methods and graphs prediction a join

	DB6	SE6	Grid	ex36	RPlan	Avg
dist.	0.58	0.54	0.64	0.34	0.27	0.47
max11UB	0.65	0.59	0.72	0.40	0.52	0.58
rel.dist.	0.82	0.84	0.81	0.49	0.64	0.72
log(rel.dist.)	0.74	0.72	0.74	0.48	0.62	0.66
rel.max11UB	0.85	0.84	0.83	0.48	0.74	0.75
adv.rel.max11UB	0.88	0.89	0.88	0.55	0.81	0.80

log(rel. distance): As in the case of the split-prediction, we can also use the logarithmic of the relative distance as a method of prediction.

rel. max11UB: Of course, we can also use the edge-lengths of the rel. distance to compute the min11UB.

adv.rel.max11UB: One disadvantage of the rel. min11UB-value is the fact, that the use of only the free amount $a(e)$ of an edge for the edge-length $l(e)$ (until now: $l(e) = 1/a(e)$) leads to a lot of edges with very long lengths (which corresponds to a small amount of free space on this edge). This is somehow misleading, so it is a good idea to assign a small free amount to every edge. This means that we use the formula $l(e) = \frac{1}{0.01+a(e)}$ for the computation of the edge-lengths.

Experiments

In order to show the effectiveness of the different values of prediction, the same experiments as in the split-case are carried out. Table A.40 shows the detailed results while Table 5.3 shows a summary of the results.

Some conclusions can be drawn from the experiments:

- The distance on its own can already be used as a method of prediction.
- The max11UB has a better correlation coefficient than the simple distance-value.
- As in the case of join-predictions, the relative distance clearly improves the correlation coefficient clearly when compared to the simple distance. But, in the case of join-prediction, the use of the logarithmic of the relative distance is a worse method of prediction.
  This can be explained by the fact that in the case of a split, small distances give good lower bounds while the logarithmic has its main effect on the large values.

So, in the case of a split, the logarithmic mainly packs the large number of bad values. But, in case of a join, large distances give good lower bounds. Therefore the logarithmic packs the good ones and they are less distinguishable, what is bad of the correlation coefficient.

- The relative max11UB increases (as expected) the coefficient when compared to the simple max11UB. And the advanced relative max11UB is even better than all the other methods.

Taking everything into account, the logarithmic of the advanced relative max11UB is quite a good predicting value for the lower bound, if a join is performed.

5.4.3 Making the Branching Selection based on Predictions for the Lower Bound

As we have already said in the introduction of this section, the branching selection is very crucial to the running-time of a branch&bound algorithm. Our goal is to make good selections based on the prediction values which we have presented in the two previous Subsections. Here we present methods of making the selection decision based on the predicted values.

Literature

Surprisingly, relatively little work has been presented on the topic of making the selection decision based on prediction-values. Eckstein [Eck94] deals with mixed integer branch&bound algorithms and presents a formula where the predicted values of the two sub-nodes are combined using a linear combination and two parameters α_1 and α_2 which have to be fixed:

$$\sigma_j = \alpha_1 \min\{D_j^+, D_j^-\} + \alpha_2 \max\{D_j^+, D_j^-\} \tag{5.1}$$

where D_j^+ and D_j^- correspond to the predicted change of the objective-value; the branching-possibility j with maximal σ_j is selected. In fact, Eckstein uses an additional addend $\alpha_o \min\{d_j^+, d_j^-\}$, however we have omitted it as it is a mixed integer specific one. Unfortunately, Eckstein gives no evidence of the best set of parameters but simply states the parameters which he has used: $\alpha_0 = 1$, $\alpha_1 = 10$, and $\alpha_2 = 1$.

Linderoth and Savelsbergh [LS99] present some experiments which compare different parameter-settings of the above Equation 5.1 and they come to the conclusion that the setting $\alpha_1 = 2$ and $\alpha_2 = 1$ gives the best performance for the mixed integer instances tested. The same setting is used by Günlük [Gün99] who compares it to other, obviously worse, methods. Finally, Martin [Mar01] presents a recent survey on mixed integer programming including branching selection.

Own Approach

Of course, we can also use Equation 5.1 and in the following experiments we will test the combinations $(\alpha_1, \alpha_2) \in \{(8,1),(4,1),(2,1),(1,1),(\frac{1}{2},1)\}$.

Additionally, we present a new idea for the branching-selection using predicted lower bounds: We are somewhere inside the branch&bound-tree and the actual lower bound has a difference of Δ to the threshold at which we could prune the actual node. We introduce f as the expected number of leaves of this subtree. If we now use branchings with an increase of D_j^+ or D_j^- on the lower bound respectively, we define:

$$f(\Delta, D_j^+, D_j^-) := \begin{cases} f(\Delta - D_j^+, D_j^+, D_j^-) + f(\Delta - D_j^-, D_j^+, D_j^-) & \text{if } \Delta \geq 0 \\ 1 & \text{else} \end{cases}$$

So, $f(\Delta, D_j^+, D_j^-)$ is the exact number of leaves of the search tree, if we would use a branching with D_j^+ and D_j^- at every node of the tree. This function can be used for branching selection by selecting the one branching-possibility j with minimal $f(\Delta, D_j^+, D_j^-)$.

For practical purposes, we have to explain how we can obtain Δ, D_j^+, and D_j^- exactly. Assuming we use the approximation algorithm of Section 4.3 with the MVarMC, we have a correct upper bound (ub) and lower bound (lb) on the MVarMC. In order to get an estimation of Δ we now simply use

$$\Delta = thr - \begin{cases} \frac{ub+lb}{2} & \text{if } ub < thr \\ lb & \text{else} \end{cases}$$

with thr is the actual threshold value at which we can prune a subproblem. If we have not computed any bound, we simply guess that $\Delta = 10$.

Of course the estimations of the D_j's have to be based on the prediction-values. However, we cannot directly use the prediction-values, since they only fulfill the property to have a good linear correlation to the D_j's while their pure value can be different. Nevertheless, we know that there are probably many pairs of vertices which do not increase the lower bound at all. So we can assume that the worst prediction-value corresponds to $D_j = 0$. On the other hand, experience shows that even the best branchings only increase the lower bound by a value of about $1\frac{1}{2}$. Of course, this depends highly on the graph, but nevertheless the ratio of different values is most important for the branching decision, so the error which we make using this assumption is small. Therefore, we use $D_j = 1\frac{1}{2}$ for the pair with the best prediction-value and all other values are calculated linearly.

Efficient calculation of $f(\Delta, D_j^+, D_j^-)$

The efficient calculation of $f(\Delta, D_j^+, D_j^-)$ is not trivial. If we use the recursive structure for a simple recursive procedure, this procedure will have exponential complexity (e.g.

if $D_j^+ = 2$ and $D_j^- = 1$ then $f(\Delta, D_j^+, D_j^-)$ will correspond to the Fibonacci-number of Δ). Another possibility for the computation of $f(\Delta, D_j^+, D_j^-)$ involves the use of dynamic programming. If we use this we will get a running time of $\Theta(\frac{\Delta^2}{D_j^+ \cdot D_j^-})$. However, there are always tuples with $\min\{D_j^+, D_j^-\} \to 0$, so the complexity of the dynamic approach is still too large for these tuples. Our goal is to find a method for the computation of $f(\Delta, D_j^+, D_j^-)$ whose running time only depends on $\max\{D_j^+, D_j^-\}$. The following theorem provides us with the basic of such an approach:

Theorem 5.5
Let $h = \frac{\Delta}{D_j^-}$ and $k = \frac{\Delta}{D_j^+}$ and we assume $h \leq k$ without loss of generality. Then

$$f(\Delta, D_j^+, D_j^-) = \sum_{a=0}^{\lfloor h \rfloor + 1} \binom{a + \lfloor \frac{k}{h}(h - a + 1) \rfloor}{a}$$

holds.

Proof:
First of all, the parameters of the function f are scalable without changing the result. Therefore

$$f(\Delta, D_j^+, D_j^-) = f'(\frac{\Delta}{D_j^+}, \frac{\Delta}{D_j^-})$$

holds with

$$f'(h,k) := \begin{cases} f'(h-1, \frac{k}{h}(h-1)) + f'(\frac{h}{k}(k-1), k-1) & \text{if } h \geq 0 \wedge k \geq 0 \\ 1 & \text{else} \end{cases}$$

We prove the theorem by examining $f'(h,k)$. Without loss of generality we assume $h \leq k$ and use an induction over h. Firstly let $h < 1$, then we have

$$\begin{aligned} f'(h,k) &= 1 + f'(\frac{h}{k}(k-1), k-1) \\ &= 2 + f'(\frac{h}{k}(k-2), k-2) \\ &= i + f'(\frac{h}{k}(k-i), k-i) \quad \forall 0 \leq i \leq k+1, i \in \mathbb{N}_0 \\ \Rightarrow \qquad f'(h,k) &= \lfloor k+1 \rfloor + 1 = \lfloor k \rfloor + 2. \end{aligned}$$

Now, looking at $\sum_{a=0}^{\lfloor h \rfloor + 1} \binom{a + \lfloor k - \frac{k}{h}(a-1) \rfloor}{a}$ with $h \leq 1$ we have

$$\begin{aligned} \sum_{a=0}^{\lfloor h \rfloor + 1} \binom{a + \lfloor k - \frac{k}{h}(a-1) \rfloor}{a} &= 1 + \binom{1 + \lfloor k - \frac{k}{h}(1-1) \rfloor}{1} \\ &= 1 + (1 + \lfloor k \rfloor) = \lfloor k \rfloor + 2 \end{aligned}$$

and therefore, the equation of the theorem is proved if $h \leq 1$. Next, we look at $h > 1$. By using induction we have

$$\begin{aligned}
f'(h,k) &= f'(h-1,\frac{k}{h}(h-1)) + f'(\frac{h}{k}(k-1),k-1) \\
&= \sum_{a=0}^{\lfloor h \rfloor} \binom{a+\lfloor \frac{k}{h}(h-1)-\frac{k}{h}(a-1)\rfloor}{a} \\
&\quad + \sum_{a=0}^{\frac{h}{k}(k-1)+1} \binom{a+\lfloor k-1-\frac{k}{h}(a-1)\rfloor}{a} \\
&= \sum_{a=1}^{\lfloor h \rfloor+1} \binom{a-1+\lfloor k-\frac{k}{h}(a-1)\rfloor}{a-1} \\
&\quad + \sum_{a=0}^{\frac{h}{k}(k-1)+1} \binom{a-1+\lfloor k-\frac{k}{h}(a-1)\rfloor}{a} \\
&= \sum_{a=0}^{\lfloor h\frac{k-1}{k}\rfloor+1} \binom{a+\lfloor k-\frac{k}{h}(a-1)\rfloor}{a} \\
&\quad + \sum_{a=\lfloor h\frac{k-1}{k}\rfloor+2}^{\lfloor h \rfloor+1} \binom{a-1+\lfloor k-\frac{k}{h}(a-1)\rfloor}{a-1}
\end{aligned}$$

Now, the theorem is proved if

$$\sum_{a=\lfloor h\frac{k-1}{k}\rfloor+2}^{\lfloor h \rfloor+1} \binom{a-1+\lfloor k-\frac{k}{h}(a-1)\rfloor}{a-1} = \sum_{a=\lfloor h\frac{k-1}{k}\rfloor+2}^{\lfloor h \rfloor+1} \binom{a+\lfloor k-\frac{k}{h}(a-1)\rfloor}{a}$$

holds. If $k > h$ it is

$$\lfloor h\frac{k-1}{k}\rfloor+2 = \lfloor h(1-\frac{1}{k})\rfloor+2 > \lfloor h(1-\frac{1}{h})\rfloor+2 = \lfloor h \rfloor+1$$

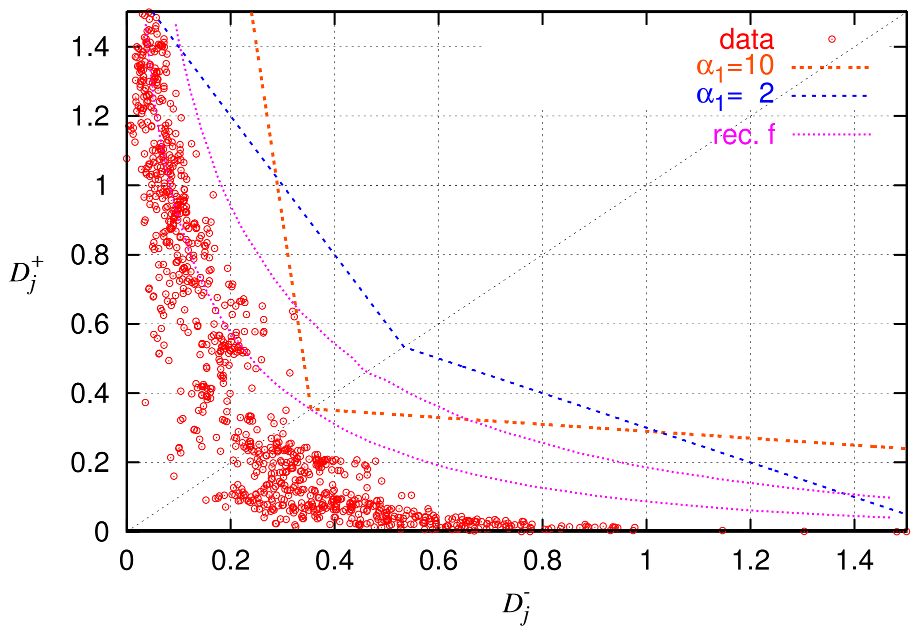
and so the sums above have no summand. It remains to look at $k = h$. Then, both sums have only one summand with $a = \lfloor h \rfloor + 1$ and the equality of the sums follows if $\lfloor k - \frac{k}{h}\lfloor h \rfloor \rfloor = 0$. With $k = h$ we have

$$\lfloor k-\frac{k}{h}\lfloor h\rfloor\rfloor = \lfloor h-\lfloor h\rfloor\rfloor = 0$$

and the theorem is proved. ■

If we use the theorem above for the computation of $f(\Delta, D_j^+, D_j^-)$, we will end up with a procedure which has to compute $\frac{\Delta}{\max\{D_j^+,D_j^-\}}$ binomial-coefficients. The computation of a binomial-coefficient $\binom{x}{y}$ can be done in time $O(\min\{y, x-y\})$, therefore:

Figure 5.5: Illustration of the insight into the branching selection



Corollary 5.1
The function $f(\Delta, D_j^+, D_j^-)$ *can be computed in time* $O(\frac{\Delta}{\max\{D_j^+, D_j^-\}})^2$.

Experimental Results

Figure 5.5 gives an insight into the differences between Ecksteins formula and our recursive function. It exemplary shows the situation of one specific graph with computed prediction-values. Every possible branching is plotted in this figure with one point corresponding to its predicted values of $D_j^+$and D_j^-. If we use Ecksteins formula with $\alpha_1 = 10$ we will always select that one point which first touches the plotted line when we move this line from the right upper corner to the left lower corner. As you can see in the figure, the use of $\alpha_1 = 2$ means that this line has a different gradient. So, with $\alpha_1 = 10$ we would select a point with $D_j^+ \approx 0.62$ and $D_j^- \approx 0.32$ whereas with $\alpha_1 = 2$ we would select a point with $D_j^+ \approx 1.49$ and $D_j^- \approx 0.06$.

In contrast to Ecksteins formula, the use of our recursive function means that instead of having straight lines, we have a curve. The use of a curve seems intuitively better than the use of straight lines because we do not have the corner at $D_j^+ = D_j^-$. Furthermore, this curve depends on Δ and it changes slightly with the distance of the point $(0,0)$. Unfortunately, there is no closed form which generally describes this curve; the curve

Table 5.4: Comparison of the different approaches to exact graph bisection (on different machines)

graph	MVarMC		[KRC00]		[BCR97]		[FMdS$^+$98]	
	time	subp.	time	subp.	time	subp.	time	subp.
BCR-6x10g	1	7	319	57	17,994	31	n.a.	
BCR-7x10t	<1	1	572	47	28,297	33	n.a.	
BCR-13x4m	172	26	34	5	5,105	5	n.a.	
BCR-m6.i	<1	1	37	1	5,737	55	103	1
DB-7	3	1	32,000	195	n.a.		n.a.	
ex36b	122	125	5	1	n.a.		n.a.	
cd47b	1	18	112	35	n.a.		n.a.	

in the figure was determined by experiments using $\Delta = 9$.

In Figure 5.6 the results of the experiments which we have carried out in order to determine the best selection-strategy are given. The detailed data is given in Tables A.41-A.43. The experiments were performed with the graph-bisection problem using the approximation algorithm with $\varepsilon = 0.5$ without variable fixing. You can see that the runs using the recursive function generally give the best results. For every α_1 there is also a type of graph where the recursive function is much better.

Finally, we can summarize by saying that the use of the recursive function presented is a good alternative to the simple formula presented by Eckstein. We have shown that the function can be computed very efficiently and the experiments show that the use of this function delivers very satisfactory results. This idea can generally be used if strong branching is used. Its main problem is the computation of the values Δ, $D_j^+$and D_j^-. On the other hand, if Ecksteins formula is used, $D_j^+$and D_j^- also have to be computed and furthermore an appropriate α_1 has to be fixed.

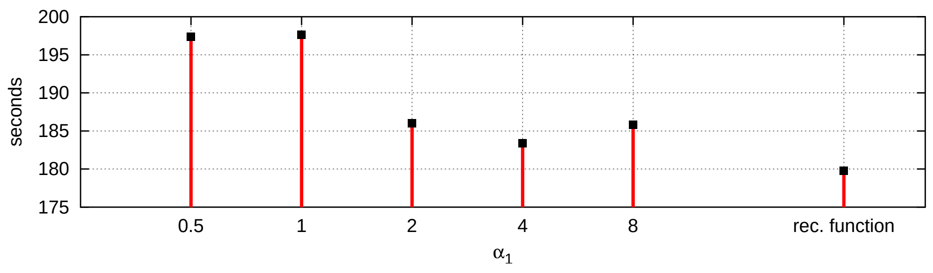
5.5 Experimental Evaluation

In this section we present experimental results which were achieved using the branch&bound-procedure previously described. In all the results which follow we use the setting which seems most promising: The approximation algorithm with $\varepsilon = 0.5$, the suggested alternative length-function-update and enhanced scaling are all used. The MVarMC-bound, the Max-Flow based variable fixing and the function $f(\Delta, D_j^+, D_j^-)$ are also used.

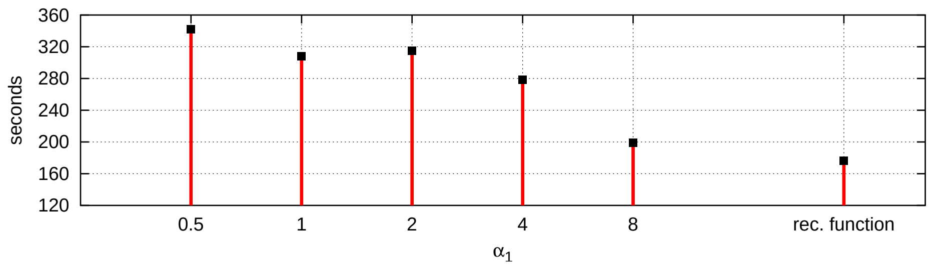
Firstly, we present results of graph bisection problems which have already been studied by other researchers. Tables A.44-A.48 give the detailed results, Table 5.4 gives

Figure 5.6: Experimental results comparing the different selection-methods with reference to the bisection problem. The upper number is the average running time in seconds, the lower number is the average size of the search-tree.

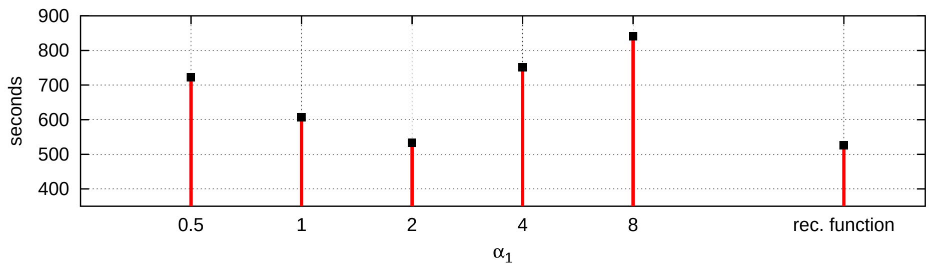
graph	rec. f	α_1				
		0.5	1	2	4	8
Random-30-0.5	180	197	198	186	183	186
	242	300	297	260	245	248
RandPlan-100-1	176	342	308	315	278	199
	182	272	252	270	195	171
RandReg-100-4	527	723	607	533	752	840
	126	235	195	149	187	210



(a) Random-graphs

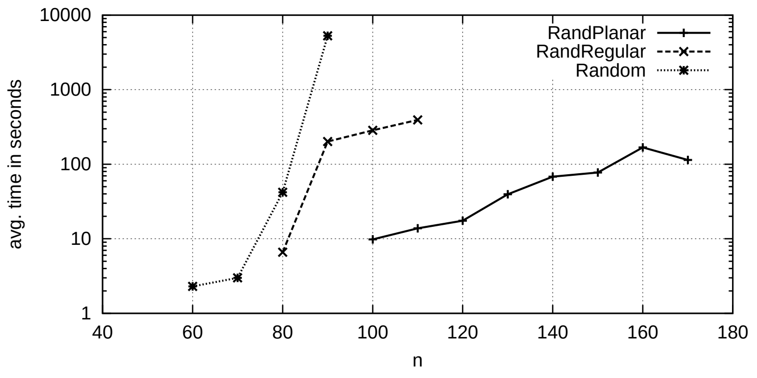


(b) RandPlan-graphs



(c) RandReg-graphs

Figure 5.7: What is possible as regards graph bisection problems



an excerpt which demonstrates the typical behavior. The quoted results of [KRC00], [BCR97] and [FMdS$^+$98] are the original values as they are presented in the corresponding publications. Therefore, the computational environments are all different, in [KRC00] a HP 9000/735 was used, in [BCR97] a SPARC 10/41 was used and in [FMdS$^+$98] a Sun 4/50 was used. We have used a SunFire 3800 system with 900 MHz UltraSparc-III processors. Therefore, it is difficult to produce reliable statements from the comparison between the running-times. Nevertheless, we feel that we can say safely that our approach is superior on the more sparse graphs such as the BCR-girds, BCR-tori and DeBruijn-graphs while the pSDP-bound-based approach is superior on the dense graphs such as the BCR-misti and ex36-graphs.

Secondly, we have performed a set of experiments in order to get an idea of the sizes of graphs which can be exactly solved in reasonable time using our MVarMC-based branch&bound procedure. Therefore, we have taken instances of RandPlan-graphs with $p = 1$, RandRegular graphs with $d = 4$ and Random graphs with $p = 0.05$ and we have solved the bisection-problem on always 10 instances. Tables A.49-A.51 give the details on the results, Figure 5.7 illustrates the averages of these runs.

Of course, these 10 runs are not sufficient for the production of reliable statements. However, in our opinion more runs are not useful, as firstly we get an idea of what is going on and secondly real instances will behave differently in any case. Therefore, the figure shows that the running times strongly depend on the class of graphs that we use. Totally random graphs prove to be the most difficult when our approach is used, random-regular graphs are easier and random-planar graphs are the easiest. If we have a running time of at most 100 seconds, we can solve random instances with up to 80 vertices, random-regular instances of degree 4 with up to 90 vertices and random-planar graphs with up to 150 vertices.

Thirdly, we can report that we were able to solve instances which to our knowledge had remained unsolvable until now. I.e. we have exactly computed the bisection-width

Table 5.5: Results on up to now unsolved problems

graph	n	$\|E\|$	$bw(G)$	time	subp.
DB-8	256	509	54	1:56:01	148
DB-9	512	1,025	92	82:44:00	3,719
SE-9	512	762	48	3:30:59	38
SE-10	1,028	1,534	82	5:09:58	19

of the DeBruijn-graphs of dimension 8 and 9 and of the Shuffle-Exchange-graphs of dimension 9 and 10. All four instances were unsolved until now. Table 5.5 gives the details of these computations. For both graphs we have taken advantage of the symmetry inside the graphs, e.g. if a pair of vertices $\{v_1, w_1\}$ is symmetric to $\{v_2, w_2\}$ and we have done a branch of split(v_1, w_1) on one side, we can fix a join of $\{v_1, w_1\}$ and $\{v_2, w_2\}$ on the other branch inside the branch&bound-algorithm. This symmetry-breaking saves about one half of the subproblems.

Chapter 6

Conclusion

In this elaboration we have introduced new lower bounds on graph partitioning problems. These new bounds are based on multicommodity-flows. The experiments presented show that the new bounds are superior when compared to other lower bounds on relatively sparse or structured graphs. The semidefinite-based bounds generally only give better results with dense random graphs.

Furthermore, we have shown that the new bounds can also be used for theoretical analyses. This is a big advantage in comparison to the semidefinite-based bounds. The analyses presented show that the VarMC-bound often gives improved results when the bisection problem is treated, while the MVarMC-bound is clearly superior for k-partitioning problems. Furthermore, we have also seen that these analyses often produce better results when compared to eigenvalue-based analyses.

In order to be able to quickly compute the bounds, we have compared linear programs, cost-decomposition based methods and an approximation algorithm. Experiments show that linear programs should be used in order to exactly compute the lower bounds. However the approximation algorithm is more effective if the bounds are used inside a branch&bound-procedure.

Finally, we have presented a branch&bound-procedure. Due to the very strong variable-fixing strategy that we have developed, the general usable branch-selection method that we have presented and of course due to the strength of the new bounds, the branch&bound-procedure is as a whole very efficient. In comparison to other approaches, our procedure is clearly superior on more dense or structured graphs. Finally, with the help of this program we were able to exactly compute the up to now unknown bisection width of the DeBruijn-graphs of dimension 8 and 9 and Shuffle-Exchange-graphs of dimension 9 and 10.

Bibliography

[Ali95] F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal on Optimization*, 5(1):13–51, 1995.

[AMO93] R.K. Ahuja, T.L. Magnati, and J.B. Orlin. *Network Flows*. Prentice Hall, 1993.

[BA00] F. Barahona and R. Anbil. The Volume Algorithm: producing primal solutions with a subgradient algorithm. *Mathematical Programming*, 87:385–399, 2000.

[BCLS87] T.N. Bui, S. Chaudhuri, F.T. Leighton, and M. Sipser. Graph bisection algorithms with good average case behavior. *Combinatorica*, 7(2):171–191, 1987.

[BCR97] L. Brunetta, M. Conforti, and G. Rinaldi. A branch-and-cut algorithm for the equicut problem. *Mathematical Programming*, 78:243–263, 1997.

[BHJS95] C. Barnhart, C.A. Hane, E.L. Johnson, and G. Sigismondi. A column generation and partitioning approach for multi-commodtiy flow problems. *Telecommunication Systems*, 3:239–258, 1995.

[BLM$^+$98] C. Bornstein, A. Litman, B. Maggs, R. Sitaraman, and T. Yatzkar. On the Bisection Width and Expansion of Butterfly Networks. In *Proceedings of the 1st Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing (IPPS/SPDP-98)*, pages 144–150. IEEE Computer Society, 1998.

[Bop87] R. B. Boppana. Eigenvalues and graph bisection: An average-case analysis. In Ashok K. Chandra, editor, *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, pages 280–285, Los Angeles, CA, October 1987. IEEE Computer Society Press.

[BR97] S. Bezrukov and B. Rovan. On partitioning grids into equal parts. *Computers and Artificial Intelligence*, 16(2):153–165, 1997.

[CDS79] D.M. Cvetkovic, M. Doob, and H. Sachs. *Spectra of Graphs: Theory and Application*. Academic Press, Inc., New York, 1979.

[CFM75] P. Camerini, L. Fratta, and F. Maffioli. On Improving Relaxation methods by Modified Gradient Techniques. *Math. Programming Studies*, 3:26–34, 1975.

[CHL$^+$94] S. Carney, M. A. Heroux, G. Li, R. Pozo, K. A. Remington, and K. Wu. A revised proposal for a sparse BLAS toolkit. Preprint 94-034, Army High Performance Computing Research Center, Minneapolis, Minnesota, 1994.

[Chu97] R.K. Chung. *Spectral Graph Theory*. American Mathematical Society, November 1997.

[Cro76] H. Crowder. Computational improvements for subgradient optimization. *Symposia Mathematica*, XIX:357–372, 1976.

[Dem89] J. Demmel. LAPACK: A portable linear algebra package for supercomputers. In *Proceedings 1989 IEEE Control Systems Society Workshop on Computer-Aided Control System Design*, pages 1–7, Tampa, Florida, December 1989.

[DH73] W. E. Donath and A. J. Hoffman. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, 17:420–425, 1973.

[Eck94] Jonathan Eckstein. Parallel Branch-and-Bound Algorithms for general Mixed Integer Programming on the CM-5. *SIAM Journal on Optimization*, 4(4):794–814, November 1994.

[ELM01] R. Elsässer, T. Lücking, and B. Monien. New Spectral Bounds on k-Partitioning of Graphs. In *Proc. of the Thirtheenth ACM Symposium on Parallel Algorithms and Architectures*, pages 255–262, 2001.

[Fan49] K. Fan. On a theorem of weyl concerning eigenvalues of linear transformations. In *International Proceedings of the National Academy of Sciences*, volume 35, pages 652–655, 1949.

[Fjä98] P.-O. Fjällström. Algorithms for graph partitioning: A survey. Linköping Electronic Articles in Computer and Information Science, 1998.

[FK02] Uriel Feige and Robert Krauthgamer. A Polylogarithmic Approximation of the Minimum Bisection. *SIAM Journal on Computing*, 31(4):1090–1118, 2002.

[Fle00] L. K. Fleischer. Approximating fractional multicommodity Flow independent of the Number of Commodities. *SIAM Journal on Discrete Mathematics*, 13(4):505–520, 2000.

[FMdS$^+$98] C. E. Ferreira, A. Martin, C. C. de Souza, R. Weismantel, and L. A. Wolsey. The node capacitated graph partitioning problem: a computational study. *Mathematical Programming*, 81:229–256, 1998.

[FRW94] Julie Falkner, Franz Rendl, and Henry Wolkowitz. A computational study of graph partitioning. *Mathematical Programming*, 66(2):211–239, 1994. Also University of Waterloo Tech.Report CORR-92-25, 1993.

[GH61] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, December 1961.

[GJS76] M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[GK98] Naveen Garg and Jochen Könemann. Faster and Simpler Algorithms for Multicommodity Flow and other Fractional Packing Problems. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 300–309, November 1998.

[GOPS98] A.V. Goldberg, J.D. Oldham, S.A. Plotkin, and C. Stein. An Implementation of a Combinatorial Approximation Algorithm for Minimum-Cost Multicommodity Flows. In *Proceedings of Integer Programming and Combinatorial Optimization (IPCO)*, volume 1412, pages 338–352, June 1998.

[GT01] Andrew V. Goldberg and Kostas Tsioutsiouliklis. Cut tree algorithms: an experimental study. *Journal of Algorithms*, 38(1):51–83, 2001.

[Gün99] Oktay Günlük. A Branch-and-Cut Algorithm for Capacitated Network Design Problems. *Mathematical Programming*, 86(1):17–39, 1999.

[Gup97] A. Gupta. Fast and effective algorithms for graph partitioning and sparse matrix ordering. *IBM Journal of Research and Development*, 41(1/2):171–184, 1997.

[Gus90] Dan Gusfield. Very simple methods for all pairs network flow analysis. *SIAM Journal on Computing*, 19(1):143–155, February 1990.

[HK00] Bruce Hendrickson and Tamara G. Kolda. Graph Partitioning Models for Parallel Computing. *Parallel Computing*, 26:1519–1534, 2000.

[HL94] B. Hendrickson and B. Leland. The chaco user's guide: Version 2.0. Technical Report SAND94-2692, Sandia National Laboratories, Albuquerque, 1994.

[HL95a] Bruce Hendrickson and Robert Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM Journal on Scientific Computing*, 16(2):452–469, 1995.

[HL95b] Bruce Hendrickson and Robert Leland. A multilevel algorithm for partitioning graphs. In *Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, page 28. ACM Press, 1995.

[HR73] L. Hyafil and R.L. Rivest. Graph partitioning and constructing optimal decision tress are polynomial complete problems. Technical Report 33, IRIA-Laboria, Rocquencourt, France, 1973.

[Iba88] T. Ibaraki. Enumerative approaches to combinatorial optimisation. *Annals of Operations Research*, 11(1–4), January 1988.

[ILO00] ILOG. *CPLEX 7.0 Reference Manual*, 2000.

[JMN93] E. Johnson, A. Mehrotra, and G. Nemhauser. Min-cut clustering. *Mathematical Programming*, 62:133–151, 1993.

[Kar98] S. Karisch. CUTSDP - A toolbox for a cutting-plane approach based on semidefinite programming, User's guide, Version 1.0. Technical Report 10/98, Department of Mathematical Modelling, Technical University of Denmark, www.imm.dtu.dk/ sk/cutsdp/, 1998.

[KK54] J.F. Kenney and E.S. Keeping. *Mathematics of Statistics - Part One*. D. van Nostrand Company, Inc., 3 edition, 1954.

[KK98a] G. Karypis and V. Kumar. METIS - A Software Package for Partitioning Unstructed Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices. Technical report, University of Minnesota, September 1998.

[KK98b] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. Technical Report TR 98-019, Dept. of Computer Science, Univ. of Minnesota, 1998.

[KR98] S. E. Karisch and F. Rendl. Semidefinite programming and graph equipartition. In P. M. Pardalos and H. Wolkowicz, editors, *Topics in Semidefinite and Interior-Point Methods*, volume 18, pages 77–95. AMS, 1998.

[KRC00] S. E. Karisch, F. Rendl, and J. Clausen. Solving graph bisection problems with semidefinite programming. *INFORMS Journal on Computing*, 12(3):177–191, 2000.

[Kum87] Vipin Kumar. BRANCH-AND-BOUND SEARCH. In Stuart C. Shapiro, editor, *Encyclopaedia of Artificial Intelligence: Vol 2*, pages 1000–1004. John Wiley and Sons, Inc., New York, 1987. Revised version appears in the second edition 1992.

[Lei92] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures*. Morgan Kaufman, 1992.

[Len90] T. Lengauer. *Combinatorial algorithms for integrated circuit layout*. Wiley - Teubner, 1990.

[LS99] Jeff T. Linderoth and Martin W.P. Savelsbergh. A Computational Study of Branch and Bound Search Strategies for Mixed Integer Programming. *INFORMS Journal on Computing*, 11(2):173–187, 1999.

[Mar01] Alexander Martin. General Mixed Integer Programming: Computational Issues for Branch- and-Cut Algorithms. In Michael Jünger and Denis Naddef, editors, *Computational Combinatorial Optimization, Optimal or Provably Near-Optimal Solutions*, volume 2241 of *Lecture Notes in Computer Science*, pages 1–25, 2001.

[Mit70] L. G. Mitten. Branch-and-bound methods: General formulation and properties. *Operations Research*, 18:24–34, 1970.

[MN95] K Mehlhorn and S. Nähler. LEDA: A Platform for Combinatorial and Geometric Computing. *Communications of the ACM*, 38(1):96–102, 1995.

[Moh97] Bojan Mohar. *Graph Symmetry: Algebraic Methods and Applications*, chapter Some Applications of Laplace Eigenvalues of Graphs, pages 225–275. NATO ASI Ser. C 497. Kluwer, 1997.

[MV80] S. Micali and V.V. Vazirani. An $o(\sqrt{n}m)$ algorithm for finding maximum matchings in general graphs. In *21st Annual Symposium on Foundation of Computer Science*, pages 17–27, 1980.

[Nog93] A. Noga. On the edge-expansion of graphs. *Combinatorics, Probability and Computing*, 11:1–10, 1993.

[PD96] R. Preis and R. Dieckmann. The PARTY Partitioning – Library User Guide – Version 1.1. SFB 376 tr-rsfb-96-024, University of Paderborn, 1996.

[PD97] R. Preis and R. Diekmann. PARTY - A Software Library for Graph Partitioning. In B.H.V. Topping, editor, *Advances in Computational Mechanics with Parallel and Distributed Processing*, pages 63–71. Civil-Comp Press, 1997.

[PR95] S. Poljak and F. Rendl. Nonpolyhedral relaxations of graph-bisection problems. *SIAM Journal on Optimization*, 5(3), 1995. 467-487.

[PR96] F. Pellegrini and J. Roman. SCOTCH: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *Proc. HPCN'96*, pages 493–498, 1996.

[Pre98] Robert Preis. The PARTY Graphpartitioning-Library, User Manual - Version 1.99, October 1998.

[Rad00] T. Radzik. Experimental study of a solution method for the multicommodity flow problem. In *Proceedings of the 2nd Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 79–102, January 2000.

[RW95] F. Rendl and H. Wolkowicz. A projection technique for partitioning the nodes of a graph. *Annals of Operations Research*, 58:155–180, 1995.

[SKK00] Kirk Schloegel, George Karypis, and Vipin Kumar. *CRPC Parallel Computing Handbook*, chapter Graph Partitioning for High Performance Scientific Simulations. Morgan Kaufmann, 2000.

[SW99] A. Steger and N.C. Wormald. Generating random regular graphs quickly. *Combinatorics, Probab. and Comput.*, 8:377–396, 1999.

[WCE95] C. Walshaw, M. Cross, and M. Everett. A localised algorithm for optimising unstructured mesh partitions. *Int. J. Supercomputer Appl.*, 9(4):280–295, 1995.

[WCE97] C. Walshaw, M. Cross, and M. G. Everett. Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes. *J. Parallel Distrib. Comput.*, 47(2):102–108, 1997. (originally published as Univ. Greenwich Tech. Rep. 97/IM/20).

[WCM00] C. Walshaw, M. Cross, and K. McManus. Multiphase Mesh Partitioning. *Appl. Math. Modelling*, 25(2):123–140, 2000. (originally published as Univ. Greenwich Tech. Rep. 99/IM/51).

[Whi84] A.T. White. *Graphs, Groups and Surfaces*. North-Holland, 1984.

Appendix A

Details of Experimental Results

graph	11-MC	VarMC	MVarMC	pSDP	BRW	DH	Opt.
DB-3	**4.0**	**4.0**	**4.0**	**4.0**	2.3	2.3	4
	<1	<1	<1	<1			
DB-4	5.9	**6.0**	**6.0**	**6.0**	3.1	3.1	6
	<1	<1	<1	<1			
DB-5	9.9	**10.0**	**10.0**	9.7	6.8	4.3	10
	<1	<1	<1	<1			
DB-6	15.9	**17.0**	**17.0**	14.9	8.4	6.3	18
	3	8	6	8			
DB-7	27.5	**29.0**	**29.0**	22.0	15.0	9.7	30
	30	53	1:00	1:24			
DB-8	48.5	**49.5**	**49.5**	35.0	22.8	15.4	54
	4:51	10:13	12:13	17:11			
SE-3	**2.0**	**2.0**	**2.0**	**2.0**	0.9	0.9	2
	<1	<1	<1	<1			
SE-4	3.0	3.1	3.1	**3.5**	1.2	1.2	4
	<1	<1	<1	<1			
SE-5	5.0	5.2	**5.2**	5.1	3.0	1.7	6
	<1	<1	1	<1			
SE-6	8.4	**8.9**	**8.9**	7.4	4.1	2.5	10
	2	6	6	9			
SE-7	14.3	**15.1**	**15.1**	10.9	7.1	4.0	16
	15	35	41	1:55			
SE-8	24.9	**26.1**	**26.1**	18.1	10.8	6.4	28
	2:54	8:05	9:43	15:43			

Table A.1: Different bounds and their computing-time for graph bisection problems with the DeBruijn and Shuffle-Exchange networks (Section 2.3.2)

graph	11-MC	VarMC	MVarMC	pSDP	BRW	DH	Opt.
Grid-8x10	**8.0** 4	**8.0** 8	**8.0** 3	7.8 22	3.6	2.0	8
Grid-9x10	**9.0** 5	**9.0** 13	**9.0** 5	7.9 31	3.8	2.2	9
Grid-10x10	**10.0** 7	**10.0** 18	**10.0** 7	8.0 38	3.9	2.4	10
Grid-11x10	10.1 9	**11.0** 13	**11.0** 16	8.8 51	4.0	2.2	11
Grid-12x10	**10.0** 11	**10.0** 10	**10.0** 12	8.6 1:09	2.0	2.0	10
Torus-8x10	**16.0** 4	**16.0** 10	**16.0** 4	15.3 15	7.6	7.6	16
Torus-9x10	**18.0** 7	**18.0** 17	**18.0** 6	17.2 24	8.6	8.6	18
Torus-10x10	**20.0** 9	**20.0** 8	**20.0** 8	17.8 34	9.5	9.5	20
Torus-11x10	**20.2** 12	**20.2** 12	**20.2** 11	17.3 47	8.7	8.7	22
Torus-12x10	**20.0** 12	**20.0** 12	**20.0** 14	16.6 1:05	8.0	8.0	20

Table A.2: Different Bounds and their computing-time for graph bisection problems with several grids and tori (Section 2.3.2)

graph	11-MC	VarMC	MVarMC	pSDP	BRW	DH	Opt.
BCR-9x4g	13.0	17.6	17.6	**19.2**	11.1	4.0	20
	<1	<1	1	<1			
BCR-5x10g	13.6	21.6	**21.8**	20.5	11.1	4.3	22
	<1	2	4	2			
BCR-6x10g	19.0	25.0	**26.2**	23.4	13.4	5.2	28
	1	4	6	4			
BCR-7x10g	13.6	18.3	**20.1**	19.3	10.9	4.5	23
	2	6	5	9			
BCR-9x4t	25.1	25.4	25.4	**25.7**	14.5	12.6	26
	<1	<1	1	<1			
BCR-10x4t	25.0	28.8	**28.8**	28.7	14.2	12.5	30
	<1	1	2	1			
BCR-10x5t	13.0	13.5	13.6	**14.1**	8.7	3.8	15
	1	2	3	2			
BCR-10x6t	38.5	40.7	**40.8**	36.4	19.5	18.2	42
	2	7	6	5			
BCR-10x8t	37.5	**41.0**	**41.0**	39.7	22.7	19.1	43
	4	17	9	14			
BCR-9x4m	360.4	364.5	364.5	**369.0**	350.3	333.8	369
	27	30	20	<1			
BCR-9x6m	783.7	789.8	789.8	**790.4**	755.6	743.7	792
	5:15	6:45	3:28	3			
BCR-10x5m	670.0	**670.0**	**670.0**	**670.0**	652.8	636.0	670
	2:38	2:48	2:02	2			
BCR-10x6m	**954.0**	**954.0**	**954.0**	**954.0**	913.4	913.2	954
	7:55	7:59	5:26	3			
BCR-10x7m	**1288.0**	**1288.0**	**1288.0**	**1288.0**	1258.5	1240.4	1288
	19:04	17:49	11:41	8			

Table A.3: Different bounds and their computing-time for graph bisection problems with several randomized grids and tori from [BCR97] (Section 2.3.2)

graph	11-MC	VarMC	MVarMC	pSDP	BRW	DH	Opt.
BCR-m4.i	-	-	**5.8** 1	5.7 <1	3.6	-	6
BCR-ma.i	-	-	1.9 2	**1.9** 3	0.7	-	2
BCR-me.i	**3.0** 1	**3.0** 2	**3.0** 2	**3.0** 4	1.2	0.4	3
BCR-m6.i	5.3 2	6.4 7	**6.6** 9	5.8 9	2.9	1.4	7
BCR-mb.i	3.0 2	3.1 7	**3.2** 5	3.1 14	1.1	0.4	4
BCR-mc.i	3.7 1	5.1 7	**5.7** 8	5.4 13	2.3	0.6	6
BCR-md.i	3.0 3	3.2 8	**3.3** 8	3.2 20	1.1	0.4	4
BCR-mf.i	3.0 4	3.2 10	**3.4** 11	2.9 26	1.1	0.3	4
BCR-m1.i	-	-	**2.9** 13	2.5 37	0.9	-	4
BCR-m8.i	**7.0** 21	**7.0** 19	**7.0** 19	6.0 2:52	2.7	1.0	7

Table A.4: Different bounds and their computing-time for graph bisection problems with some real-world instances from [BCR97] (Section 2.3.2)

graph	11-MC	VarMC	MVarMC	pSDP	BRW	DH	Opt.
Random-100-0.05-0	-	-	41.2 1:13	**42.3** 34	32.8	-	47
Random-100-0.05-1	25.3 21	**42.2** 1:34	**42.2** 56	41.2 31	32.9	11.9	43
Random-100-0.05-2	-	-	42.3 1:02	**43.3** 30	30.8	-	(48)
Random-100-0.05-3	12.8 12	25.0 35	**34.0** 1:31	32.5 30	22.6	7.2	34
Random-100-0.05-4	-	-	42.3 59	**43.8** 35	33.8	-	48
Random-50-0.2-0	38.3 6	64.0 16	64.0 21	**66.6** 2	59.8	33.4	67
Random-50-0.2-1	51.0 7	71.7 19	71.7 21	**76.6** 2	69.2	40.8	78
Random-50-0.2-2	51.0 7	68.0 16	68.0 23	**73.2** 2	66.6	42.4	75
Random-50-0.2-3	38.3 5	63.6 16	63.6 17	**69.0** 2	62.7	31.6	71
Random-50-0.2-4	25.5 8	50.0 8	67.2 19	**71.9** 2	65.3	22.4	74
Random-32-0.5-0	76.5 3	87.2 5	87.2 5	**95.3** <1	90.6	67.1	96
Random-32-0.5-1	82.5 3	87.3 6	87.3 10	**94.6** <1	89.5	68.8	96
Random-32-0.5-2	66.1 3	84.4 5	84.4 10	**92.7** <1	88.3	58.7	93
Random-32-0.5-3	79.0 2	83.3 10	83.3 13	**89.7** <1	85.4	66.1	90
Random-32-0.5-4	74.3 3	80.3 7	80.3 12	**85.0** <1	80.7	61.8	85
RandW-32-10-0	883.6 12	922.5 13	922.5 15	**944.0** <1	921.6	812.0	944
RandW-32-10-1	743.2 10	919.5 15	919.5 16	**955.0** <1	928.2	707.2	959
RandW-32-10-2	879.9 10	930.5 16	930.5 17	**962.0** <1	940.9	791.5	962
RandW-32-10-3	801.0 10	949.8 15	949.8 19	**977.0** <1	956.8	748.2	977
RandW-32-10-4	866.1 10	977.8 13	977.8 20	**1020.3** <1	997.5	826.8	1026

Table A.5: Different bounds and their computing-time for graph bisection problems with some randomly generated graphs (Section 2.3.2)

graph	11-MC	VarMC	MVarMC	pSDP	BRW	DH	Opt.
RandPlan-100-1-0	24.7 11	35.1 18	**35.1** 25	34.7 34	21.1	12.1	38
RandPlan-100-1-1	20.4 10	**30.6** 29	**30.6** 26	29.8 36	19.8	11.3	32
RandPlan-100-1-2	18.2 10	25.9 13	27.9 22	**28.8** 36	18.1	8.8	31
RandPlan-100-1-3	19.0 11	27.8 15	**29.5** 39	28.6 36	18.6	8.9	30
RandPlan-100-1-4	25.0 13	30.2 22	**30.2** 40	28.9 31	17.5	11.2	32
RandRegular-100-3-0	12.1 8	**12.9** 39	**12.9** 29	11.0 31	7.1	5.4	14
RandRegular-100-3-1	15.2 16	**15.4** 55	**15.4** 35	12.5 31	8.5	6.7	16
RandRegular-100-3-2	14.3 11	**14.9** 57	**14.9** 31	12.1 31	8.3	6.2	16
RandRegular-100-3-3	14.6 14	**14.8** 1:04	**14.8** 37	12.1 31	7.9	6.7	16
RandRegular-100-3-4	11.9 7	**13.0** 36	**13.0** 17	10.8 32	6.9	5.1	14
RandRegular-100-4-0	27.7 26	**27.9** 49	**27.9** 48	26.6 31	20.4	18.0	32
RandRegular-100-4-1	27.0 24	**27.2** 40	**27.2** 38	25.4 32	18.9	16.2	30
RandRegular-100-4-2	27.6 27	**27.8** 47	**27.8** 51	26.6 31	20.1	15.3	32
RandRegular-100-4-3	27.6 24	**27.7** 45	**27.7** 36	26.5 32	20.3	16.1	30
RandRegular-100-4-4	25.7 20	**26.7** 39	**26.7** 40	24.9 32	18.7	13.6	28

Table A.6: Different bounds and their computing-time for graph bisection problems with some randomly generated planar or respectively regular graphs (Section 2.3.2)

graph	11-MC	VarMC	MVarMC	GP$_{R1}$	DH	Opt.
DB-3	6.0	6.0	**9.0**	5.7	7.2	9
	<1	<1	<1	<1		
DB-4	8.9	9.0	**12.0**	7.5	7.9	14
	<1	<1	<1	<1		
DB-5	14.8	15.0	**17.5**	10.3	9.9	19
	<1	<1	1	<1		
DB-6	23.8	25.4	**27.2**	15.4	13.6	32
	3	9	10	<1		
SE-3	3.0	3.0	**6.0**	3.0	3.4	6
	<1	<1	<1	<1		
SE-4	4.6	4.7	**7.2**	3.3	3.3	8
	<1	<1	<1	<1		
SE-5	7.5	7.8	**10.2**	4.6	4.5	11
	<1	<1	1	<1		
SE-6	12.6	13.4	**15.2**	7.2	6.3	18
	2	6	5	<1		

Table A.7: Different bounds and their computing-time for 4-partitioning problems with the DeBruijn and Shuffle-Exchange networks (Section 2.3.3)

graph	11-MC	VarMC	MVarMC	GP$_{R1}$	DH	Opt.
Grid-8x13	12.1	13.0	**20.5**	5.5	5.5	(23)
	8	11	42	1		
Grid-10x10	15.0	15.0	**20.0**	6.0	4.9	20
	7	16	29	1		
Grid-12x9	13.5	13.5	**21.1**	5.8	5.1	(23)
	7	8	33	1		
Torus-8x13	24.1	24.1	**31.4**	8.9	13.6	(38)
	8	8	13	<1		
Torus-10x10	30.0	30.0	**32.7**	14.3	14.3	(40)
	8	8	9	<1		
Torus-12x9	27.0	27.0	**33.4**	10.8	13.6	(36)
	11	11	14	1		

Table A.8: Different bounds and their computing-time for 4-partitioning problems with several grids and tori (Section 2.3.3)

graph	11-MC	VarMC	MVarMC	GP_{R1}	DH	Opt.
BCR-6x10g	28.6 1	37.5 3	**51.2** 4	20.4 <1	16.7	(59)
BCR-10x4t	37.5 <1	43.2 1	**65.0** 4	21.4 <1	34.6	(69)
BCR-10x6t	57.8 2	61.1 7	**75.9** 10	29.3 <1	34.0	(79)
BCR-9x4m	540.7 25	546.8 33	**591.8** 30	525.5 <1	533.6	(621)
BCR-10x6m	1431.0 8:02	1431.0 7:52	**1490.5** 10:10	1398.5 <1	1399.4	(1494)

Table A.9: Different bounds and their computing-time for 4-partitioning problems with several randomized grids and tori from [BCR97] (Section 2.3.3)

graph	11-MC	VarMC	MVarMC	GP_{R1}	DH	Opt.
BCR-m4.i	-	-	**11.4** 1	5.5 <1	2.4	12
BCR-me.i	4.5 1	4.5 2	**10.0** 5	1.9 <1	2.6	10
BCR-md.i	4.5 2	4.7 8	**9.8** 16	1.8 <1	2.1	11
BCR-m1.i	-	-	**8.5** 27	1.4 1	0.8	11
BCR-m8.i	10.5 22	10.5 19	**20.7** 2:53	4.1 6	3.5	22

Table A.10: Different bounds and their computing-time for 4-partitioning problems with some real-world instances from [BCR97] (Section 2.3.3)

graph	11-MC	VarMC	MVarMC	GP$_{R1}$	DH	Opt.
Random-60-0.05-0	-	-	**16.6** 8	10.9 <1	-	(18)
Random-60-0.05-1	-	-	**18.8** 6	11.4 <1	-	(22)
Random-60-0.05-2	-	-	**16.4** 5	10.2 <1	-	(20)
Random-60-0.05-3	-	-	**17.2** 7	10.9 <1	-	(20)
Random-60-0.05-4	-	-	**21.7** 6	14.0 <1	-	(26)
Random-32-0.5-0	114.8 3	130.8 6	135.5 7	**137.0** <1	108.1	(153)
Random-32-0.5-1	123.7 3	130.9 6	134.0 10	**135.5** <1	114.3	(152)
Random-32-0.5-2	99.1 3	126.7 6	130.5 12	**132.7** <1	99.4	(149)
Random-32-0.5-3	118.5 2	124.9 8	128.0 9	**128.6** <1	108.2	(146)
Random-32-0.5-4	111.5 3	120.5 6	**122.8** 11	121.2 <1	102.0	(141)
RandW-32-10-0	1325.4 12	1383.8 13	**1402.0** 17	1385.5 <1	1249.0	(1477)
RandW-32-10-1	1114.8 10	1379.3 15	**1399.0** 20	1395.4 <1	1192.7	(1487)
RandW-32-10-2	1319.8 11	1395.7 14	1412.8 19	**1413.0** <1	1262.6	(1506)
RandW-32-10-3	1201.5 10	1424.7 16	**1441.0** 15	1436.8 <1	1245.9	(1519)
RandW-32-10-4	1299.2 10	1466.8 13	1491.5 21	**1498.6** <1	1296.5	(1586)

Table A.11: Different bounds and their computing-time for 4-partitioning problems with some randomly generated graphs (Section 2.3.3)

graph	11-MC	VarMC	MVarMC	GP_{R1}	DH	Opt.
RandPlan-60-1-0	26.8 2	38.1 11	**45.5** 11	31.4 <1	18.8	(47)
RandPlan-60-1-1	23.4 2	37.5 10	**43.7** 11	30.5 <1	17.5	(45)
RandPlan-60-1-2	27.3 2	36.8 11	**42.3** 13	31.5 <1	18.2	(44)
RandPlan-60-1-3	30.9 2	36.2 9	**47.3** 10	32.4 <1	20.8	(51)
RandPlan-60-1-4	29.2 3	32.6 9	**39.7** 8	26.7 <1	16.1	(42)
RandRegular-60-3-0	10.6 <1	11.7 4	**15.1** 6	7.1 <1	6.6	(18)
RandRegular-60-3-1	14.3 1	15.0 6	**16.5** 7	8.5 <1	7.9	(19)
RandRegular-60-3-2	13.9 1	15.0 11	**16.2** 5	9.0 <1	8.2	(19)
RandRegular-60-3-3	14.5 2	14.7 7	**15.7** 4	8.6 <1	7.3	(18)
RandRegular-60-3-4	12.0 1	12.0 3	**16.3** 6	7.7 <1	8.3	(18)
RandRegular-60-4-0	28.6 5	28.8 10	**30.1** 8	21.2 <1	20.5	(37)
RandRegular-60-4-1	27.8 3	28.7 11	**30.0** 7	20.4 <1	19.5	(35)
RandRegular-60-4-2	24.7 2	27.3 11	**29.4** 8	19.4 <1	19.3	(34)
RandRegular-60-4-3	27.9 4	28.2 11	**29.2** 9	19.6 <1	18.4	(34)
RandRegular-60-4-4	25.0 3	26.3 13	**27.9** 7	18.5 <1	16.8	(32)

Table A.12: Different bounds and their computing-time for 4-partitioning problems with some randomly generated planar or respectively regular graphs (Section 2.3.3)

graph	11-MC	VarMC	MVarMC	pSDP	DH	Opt.
DB-3	**3.7**	**3.8**	**3.8**	**3.8**	1.8	4
	<1	<1	<1	<1		
DB-4	5.5	5.5	5.5	**5.6**	2.3	6
	<1	<1	<1	<1		
DB-5	8.9	8.9	**9.0**	8.1	2.9	(10)
	<1	<1	<1	<1		
DB-6	14.3	14.4	**14.4**	12.3	4.4	(16)
	4	14	7	6		
DB-7	**24.5**	**24.5**	**24.5**	18.9	6.5	26
	30	32	28	1:19		
DB-8	43.2	43.3	**43.4**	30.8	10.4	(48)
	6:37	8:47	16:25	14:31		
SE-3	**1.9**	**1.9**	**1.9**	**1.9**	0.7	2
	<1	<1	<1	<1		
SE-4	**2.9**	**2.9**	**2.9**	2.8	0.9	3
	<1	<1	<1	<1		
SE-5	**4.5**	**4.5**	**4.5**	3.9	1.1	5
	<1	<1	<1	<1		
SE-6	7.6	7.6	**7.6**	6.2	1.7	8
	2	4	3	6		
SE-7	**12.7**	**12.7**	**12.7**	9.3	2.7	(14)
	15	13	21	1:20		
SE-8	**22.2**	**22.2**	**22.2**	14.9	4.3	(24)
	2:24	2:32	3:25	15:03		

Table A.13: Different bounds and their computing-time for partitioning problems using $k = 2$ and $M = \lfloor \frac{2}{3} n \rfloor$ with the DeBruijn and Shuffle-Exchange networks (Section 2.3.4)

graph	11-MC	VarMC	MVarMC	pSDP	DH	Opt.
Grid-8x10	**7.2**	**7.2**	**7.2**	6.4	1.3	8
	4	8	4	16		
Grid-9x10	**8.0**	**8.0**	**8.0**	7.2	1.5	(9)
	5	12	5	23		
Grid-10x10	**9.0**	**9.0**	**9.0**	6.8	1.7	(10)
	7	19	7	35		
Grid-11x10	**9.0**	**9.0**	**9.0**	7.1	1.5	10
	9	8	10	43		
Grid-12x10	**8.9**	**8.9**	**8.9**	6.8	1.4	(10)
	11	11	14	1:00		
Torus-8x10	**14.3**	**14.3**	**14.3**	13.5	5.2	(16)
	5	14	5	15		
Torus-9x10	**16.0**	**16.0**	**16.0**	14.5	5.7	(18)
	7	22	7	23		
Torus-10x10	**18.0**	**18.0**	**18.0**	16.3	6.5	(20)
	8	9	9	30		
Torus-11x10	**18.0**	**18.0**	**18.0**	15.8	5.9	(20)
	12	12	11	39		
Torus-12x10	**17.8**	**17.8**	**17.8**	14.5	5.4	(20)
	13	13	15	52		

Table A.14: Different bounds and their computing-time for partitioning problems using $k = 2$ and $M = \lfloor \frac{2}{3} n \rfloor$ with several grids and tori (Section 2.3.4)

graph	11-MC	VarMC	MVarMC	pSDP	DH	Opt.
BCR-9x4g	11.5 <1	12.0 1	12.2 1	**12.3** <1	2.7	(14)
BCR-5x10g	12.2 1	15.3 3	**16.9** 2	15.6 1	2.9	(19)
BCR-6x10g	16.9 1	17.4 3	**17.6** 3	17.0 4	3.5	(20)
BCR-7x10g	12.3 2	12.5 8	**12.8** 7	12.6 7	3.1	14
BCR-9x4t	**22.3** <1	**22.3** 1	**22.3** 1	21.4 <1	8.4	23
BCR-10x4t	**22.8** <1	**22.8** 1	**22.8** 1	22.7 <1	8.8	(24)
BCR-10x5t	**11.7** <1	**11.7** 2	**11.7** 2	11.4 2	2.6	(13)
BCR-10x6t	**34.3** 2	**34.3** 7	**34.3** 3	30.0 3	12.2	(37)
BCR-10x8t	**33.5** 6	**33.5** 18	**33.5** 6	31.0 14	12.9	(36)
BCR-9x4m	**320.4** 26	**320.4** 27	320.4 17	320.1 <1	222.5	(324)
BCR-9x6m	**696.6** 5:22	**696.6** 5:40	**696.6** 2:50	696.1 2	495.8	(702)
BCR-10x5m	**601.4** 2:59	**601.4** 3:07	**601.4** 1:52	601.1 2	432.5	(615)
BCR-10x6m	**848.0** 8:18	**848.0** 8:07	**848.0** 5:04	846.8 4	608.8	(863)
BCR-10x7m	**1160.8** 19:15	**1160.8** 20:59	**1160.8** 12:19	1157.0 7	850.6	(1176)

Table A.15: Different bounds and their computing-time for partitioning problems using $k = 2$ and $M = \lfloor \frac{2}{3} n \rfloor$ with several randomized grids and tori from [BCR97] (Section 2.3.4)

graph	11-MC	VarMC	MVarMC	pSDP	DH	Opt.
BCR-m4.i	-	-	**5.0**	4.8	0.0	(6)
			<1	<1		
BCR-ma.i	-	-	**1.6**	1.3	-	2
			2	2		
BCR-me.i	**2.7**	**2.7**	**2.7**	2.1	0.3	3
	1	2	1	4		
BCR-m6.i	**4.8**	**4.8**	**4.8**	4.3	0.9	5
	1	5	3	7		
BCR-mb.i	**2.7**	**2.7**	**2.7**	2.0	0.3	3
	2	4	3	11		
BCR-mc.i	3.3	3.5	**3.6**	2.8	0.4	4
	2	7	4	10		
BCR-md.i	**2.7**	**2.7**	**2.7**	1.9	0.3	3
	2	7	4	14		
BCR-mf.i	**2.7**	**2.7**	**2.7**	1.6	0.2	3
	4	11	7	21		
BCR-m1.i	-	-	**2.1**	1.3	0.0	3
			12	33		
BCR-m8.i	**6.3**	**6.3**	**6.3**	4.1	0.7	7
	23	22	22	2:05		

Table A.16: Different bounds and their computing-time for partitioning problems using $k = 2$ and $M = \lfloor \frac{2}{3}n \rfloor$ with some real-world instances from [BCR97] (Section 2.3.4)

graph	11-MC	VarMC	MVarMC	pSDP	DH	Opt.
Random-100-0.05-0	-	-	31.3	**33.7**	0.0	(38)
			1:42	28		
Random-100-0.05-1	22.7	30.3	34.0	**34.4**	8.1	(38)
	20	1:40	1:53	27		
Random-100-0.05-2	-	-	32.7	**35.0**	0.0	(41)
			1:38	28		
Random-100-0.05-3	11.4	17.0	**26.8**	26.6	4.9	32
	13	52	1:31	29		
Random-100-0.05-4	-	-	31.5	**34.5**	0.0	(39)
			1:22	29		
Random-50-0.2-0	34.3	45.9	53.7	**57.6**	22.7	(62)
	6	17	13	1		
Random-50-0.2-1	45.8	54.8	60.3	**64.9**	27.7	(68)
	8	20	15	2		
Random-50-0.2-2	45.8	53.1	57.1	**62.1**	28.9	(64)
	7	17	17	1		
Random-50-0.2-3	34.3	45.7	51.8	**58.1**	21.5	(63)
	5	16	15	1		
Random-50-0.2-4	22.9	34.0	55.3	**60.9**	15.3	(65)
	7	11	19	1		
Random-32-0.5-0	69.0	72.3	73.5	**81.8**	46.1	82
	3	6	6	<1		
Random-32-0.5-1	74.4	74.4	74.4	**83.2**	47.3	(87)
	3	3	6	<1		
Random-32-0.5-2	59.6	67.1	71.3	**80.0**	40.3	80
	4	6	6	<1		
Random-32-0.5-3	71.3	71.3	71.3	**78.3**	45.4	79
	3	3	5	<1		
Random-32-0.5-4	67.1	68.3	68.8	**74.5**	42.5	75
	3	4	6	<1		
RandW-32-10-0	797.3	811.5	820.0	**845.9**	558.3	(860)
	12	18	14	<1		
RandW-32-10-1	670.6	750.5	807.6	**834.9**	486.2	835
	11	20	15	<1		
RandW-32-10-2	794.0	810.3	814.9	**841.4**	544.2	(844)
	11	16	21	<1		
RandW-32-10-3	722.8	789.7	836.8	**865.4**	514.4	(867)
	11	21	17	<1		
RandW-32-10-4	781.5	830.0	862.0	**889.9**	568.4	(894)
	10	18	19	<1		

Table A.17: Different bounds and their computing-time for partitioning problems using $k = 2$ and $M = \lfloor \frac{2}{3} n \rfloor$ with some randomly generated graphs (Section 2.3.4)

graph	11-MC	VarMC	MVarMC	pSDP	DH	Opt.
RandPlan-100-1-0	22.2 13	24.2 21	24.4 34	**24.5** 32	8.3	27
RandPlan-100-1-1	18.3 11	21.3 24	21.9 28	**22.4** 30	7.7	(25)
RandPlan-100-1-2	16.3 11	17.6 18	18.3 25	**18.7** 30	6.0	(22)
RandPlan-100-1-3	17.1 11	18.9 24	**19.8** 33	19.5 31	6.1	(21)
RandPlan-100-1-4	22.4 13	23.9 25	**23.9** 30	22.7 34	7.6	(25)
RandRegular-100-3-0	**10.8** 8	**10.8** 22	**10.8** 13	9.7 28	3.7	(12)
RandRegular-100-3-1	**13.7** 16	**13.7** 38	**13.7** 21	10.7 28	4.6	14
RandRegular-100-3-2	12.8 11	12.9 51	**12.9** 33	10.4 29	4.2	(14)
RandRegular-100-3-3	**13.1** 15	**13.1** 33	**13.1** 18	10.4 29	4.5	14
RandRegular-100-3-4	10.7 7	10.8 29	**10.8** 17	9.5 28	3.4	(12)
RandRegular-100-4-0	**24.9** 29	**24.9** 28	**24.9** 31	23.3 27	12.2	(28)
RandRegular-100-4-1	**24.3** 30	**24.3** 26	**24.3** 29	22.3 27	11.0	(26)
RandRegular-100-4-2	**24.8** 26	**24.8** 29	**24.8** 29	23.3 30	10.4	(28)
RandRegular-100-4-3	**24.8** 27	**24.8** 30	**24.8** 28	23.1 29	10.9	(28)
RandRegular-100-4-4	23.1 21	23.1 29	**23.1** 24	21.4 34	9.3	24

Table A.18: Different bounds and their computing-time for partitioning problems using $k = 2$ and $M = \lfloor \frac{2}{3} n \rfloor$ with some randomly generated planar or respectively regular graphs (Section 2.3.4)

	pure subgr.		Crowder		mod. CFM		Volume	
graph	time	subp.	time	subp.	time	subp.	time	subp.
RandPlan-0	40:52	15	22:37	15	1:15:02	15	50:40	15
RandPlan-1	19:50	19	17:15	19	32:22	19	18:27	19
RandPlan-2	2:56:20	51	2:18:02	51	6:34:11	51	4:03:19	51
RandPlan-3	39:54	11	23:03	11	1:11:02	11	15:40	11
RandPlan-4	7:51	7	4:09	7	9:30	7	5:44	7
RandPlan-5	48	1	31	1	50	1	25	1
RandPlan-6	21:52:27	443	11:06:40	443	>48:00:00	443	13:38:02	443
RandPlan-7	4:21:26	91	2:36:35	91	8:41:30	91	2:48:38	91
RandPlan-8	7:35	5	3:28	5	7:36	5	3:34	5
RandPlan-9	48	1	1:07	1	1:24	1	1:13	1
RandPlan-10	1:30	1	41	1	1:32	1	52	1
RandPlan-11	16:10:39	581	10:03:31	581	40:28:25	581	6:33:41	581
RandPlan-12	38:30	13	17:06	13	45:53	13	20:27	13
RandPlan-13	33:09	13	12:29	13	22:54	13	40:51	13
RandPlan-14	54	1	33	1	44	1	47	1
RandPlan-15	34	1	45	1	47	1	35	1
RandPlan-17	6:12:15	219	3:47:48	219	11:52:41	219	7:58:43	219
RandPlan-18	4:44:18	147	3:14:13	147	11:34:59	147	3:04:06	147
RandPlan-19	14:31	5	7:38	5	11:02	5	9:04	5
Median	1989	13	1026	13	1942	13	1107	13
Maximum	78747	581	40000	581	172800	581	49082	581
Minimum	34	1	31	1	44	1	25	1
Avg.	**11318.5**	**85.5**	**6625.8**	**85.5**	**24986.5**	**85.5**	**7752.0**	**85.5**
Std. Deviation	21512.6	162.7	11933.5	162.7	49657.7	162.7	13138.3	162.7
rel. Dev. [%]	190.1	190.2	180.1	190.2	198.7	190.2	169.5	190.2
Skewness	2.5	2.4	2.2	2.4	2.4	2.4	2.2	2.4

Table A.19: Results of Cost-Decomposition with the max-Cut-Flow formulation (Section 4.4.1)

	pure subgr.		Crowder		mod. CFM		Volume	
graph	time	subp.	time	subp.	time	subp.	time	subp.
RandRegular-0	23:45	49	12:23	43	14:31	43	13:50	45
RandRegular-1	7:15	13	1:55	7	2:25	7	1:31	7
RandRegular-2	30:04	57	12:29	37	18:04	41	14:43	47
RandRegular-3	6:30	9	1:57	5	1:55	5	2:42	7
RandRegular-4	3:13	3	21	1	23	1	21	1
RandRegular-5	5:33	9	2:18	5	2:25	5	1:41	5
RandRegular-6	12:18	21	4:37	13	6:13	13	4:12	11
RandRegular-7	24:21	49	12:35	37	15:48	41	9:16	39
RandRegular-8	20:43	39	11:11	29	15:30	31	9:28	35
RandRegular-9	45:48	95	27:36	69	31:23	75	25:56	87
RandRegular-10	14:23	27	7:50	21	10:11	23	7:30	27
RandRegular-11	24:04	41	12:37	25	15:26	29	9:18	27
RandRegular-12	11:13	23	3:39	13	4:07	13	4:50	17
RandRegular-13	24:47	43	9:03	25	15:05	31	9:34	31
RandRegular-14	3:52	7	2:13	3	2:51	5	43	3
RandRegular-15	15:13	35	9:48	25	13:08	25	9:04	29
RandRegular-16	24:36	41	8:09	25	12:31	27	6:53	25
RandRegular-17	5:32	7	1:59	5	3:36	7	2:31	7
RandRegular-18	4:56	9	1:25	5	2:30	5	1:06	5
RandRegular-19	1:29:23	191	52:11	165	57:40	159	54:20	189
Median	863	27	470	21	611	23	413	25
Maximum	5363	191	3131	165	3460	159	3260	189
Minimum	193	3	21	1	23	1	21	1
Avg.	**1192.5**	**38.4**	**588.8**	**27.9**	**737.1**	**29.3**	**568.5**	**32.2**
Std. Deviation	1183.9	42.5	711.9	36.5	792.6	35.6	733.3	42.3
rel. Dev. [%]	99.3	110.6	120.9	130.7	107.5	121.6	129.0	131.4
Skewness	2.5	2.7	2.7	3.1	2.4	2.8	2.9	3.0

Table A.20: Results of Cost-Decomposition with the max-Cut-Flow formulation (Section 4.4.1)

graph	pure subgr.		Crowder		mod. CFM		Volume	
	time	subp.	time	subp.	time	subp.	time	subp.
Random-0	12:33	87	9:48	87	11:34	85	23:04	87
Random-1	1:36	7	31	7	33	7	35	7
Random-2	1:00	7	54	7	53	7	37	7
Random-3	5:42	37	5:37	41	4:44	37	5:56	51
Random-4	2:45	27	2:12	29	1:37	27	2:53	37
Random-5	10:18	61	11:26	69	8:51	59	9:32	93
Random-6	56	9	32	9	31	9	32	9
Random-7	7:44	53	8:09	61	7:25	49	6:43	65
Random-8	3:05	17	3:04	19	2:08	15	2:35	23
Random-9	3:50	39	3:08	39	4:00	37	3:53	45
Random-10	7:13	35	4:51	35	6:01	33	5:11	47
Random-11	2:23:10	1625	2:21:27	1625	2:32:51	1625	2:58:43	1625
Random-12	5:54	35	5:04	35	3:58	31	5:07	43
Random-13	22	5	21	5	21	5	19	5
Random-14	8:17	69	6:49	69	6:30	63	7:20	91
Random-15	3:36	19	2:15	17	2:39	17	2:29	23
Random-16	5:01	31	4:09	31	4:11	31	4:49	39
Random-17	11:02	71	10:09	77	9:28	67	9:58	93
Random-18	5:02	33	2:54	33	4:04	33	3:32	37
Random-19	10:21	81	8:11	87	8:24	77	9:23	109
Median	302	35	249	35	244	33	289	43
Maximum	8590	1625	8487	1625	9171	1625	10723	1625
Minimum	22	5	21	5	21	5	19	5
Avg.	**748.4**	**117.4**	**694.5**	**119.1**	**722.1**	**115.7**	**849.5**	**126.8**
Std. Deviation	1858.3	355.7	1845.5	355.5	1998.3	356.1	2344.1	354.1
rel. Dev. [%]	248.3	303.0	265.7	298.5	276.7	307.7	275.9	279.3
Skewness	4.4	4.4	4.4	4.4	4.4	4.4	4.4	4.4

Table A.21: Results of Cost-Decomposition with the max-Cut-Flow formulation (Section 4.4.1)

graph	pure subgr.		Crowder		mod. CFM		Volume	
	time	subp.	time	subp.	time	subp.	time	subp.
RandW-0	2:53	13	2:35	13	3:03	13	2:52	19
RandW-1	1:27	3	1:08	3	52	3	1:17	5
RandW-2	1:52	1	57	1	58	1	51	3
RandW-3	2:56	5	1:00	3	1:06	3	2:15	7
RandW-4	2:32	11	1:44	11	2:43	11	3:31	17
RandW-5	1:44	11	1:26	11	1:26	11	2:21	13
RandW-6	36	1	28	1	9	1	35	1
RandW-7	9	1	14	1	18	1	7	1
RandW-8	4:32	19	3:43	21	4:08	21	5:15	37
RandW-9	1:40	3	1:50	3	1:35	3	1:26	5
RandW-10	1:05	9	58	9	55	9	52	9
RandW-11	2:22	9	2:00	9	2:18	9	3:29	17
RandW-12	2:41	23	3:07	23	3:25	23	5:08	35
RandW-13	1:44	3	1:36	3	1:54	3	1:30	5
RandW-14	4:00	19	3:08	19	3:58	17	4:29	29
RandW-15	3:57	21	3:54	27	3:46	21	5:32	37
RandW-16	4:12	19	2:57	19	3:24	15	3:25	23
RandW-17	3:46	13	3:05	15	3:13	13	2:58	23
RandW-18	4:13	17	4:12	15	3:49	15	3:46	23
RandW-19	1:57	11	2:44	13	1:50	11	3:34	23
Median	142	11	110	11	114	11	172	17
Maximum	272	23	252	27	248	23	332	37
Minimum	9	1	14	1	9	1	7	1
Avg.	**150.9**	**10.6**	**128.3**	**11.0**	**134.5**	**10.2**	**165.7**	**16.6**
Std. Deviation	76.8	7.3	70.8	8.1	77.9	7.1	98.0	12.0
rel. Dev. [%]	50.9	68.9	55.2	73.2	57.9	70.1	59.1	72.2
Skewness	-0.0	0.1	0.1	0.3	-0.0	0.2	0.1	0.3

Table A.22: Results of Cost-Decomposition with the max-Cut-Flow formulation (Section 4.4.1)

	pure subgr.		Crowder		mod. CFM		Volume	
graph	time	subp.	time	subp.	time	subp.	time	subp.
RandPlan-0	8:12	15	12:09	15	8:18	15	10:05	15
RandPlan-1	12:06	19	12:45	19	11:43	19	11:38	19
RandPlan-2	31:57	51	17:56	51	29:51	51	15:33	51
RandPlan-3	7:57	11	7:08	11	7:19	11	6:03	11
RandPlan-4	4:35	7	6:34	7	5:04	7	6:16	7
RandPlan-5	56	1	55	1	38	1	46	1
RandPlan-6	3:25:40	443	1:23:08	443	3:16:58	443	1:53:56	443
RandPlan-7	41:30	91	21:02	91	38:53	91	23:34	91
RandPlan-8	3:38	5	3:18	5	3:13	5	2:36	5
RandPlan-9	44	1	55	1	35	1	39	1
RandPlan-10	54	1	46	1	21	1	31	1
RandPlan-11	2:39:35	581	1:57:37	581	2:06:15	581	1:45:46	581
RandPlan-12	9:10	13	8:52	13	7:27	13	7:15	13
RandPlan-13	8:34	13	6:04	13	6:26	13	5:31	13
RandPlan-14	46	1	43	1	30	1	37	1
RandPlan-15	51	1	36	1	27	1	28	1
RandPlan-17	1:23:10	219	1:05:26	219	1:15:05	219	59:52	219
RandPlan-18	1:00:16	147	34:05	147	57:03	147	38:56	147
RandPlan-19	2:59	5	4:07	5	3:53	5	3:43	5
Median	492	13	428	13	439	13	376	13
Maximum	12340	581	7057	581	11818	581	6836	581
Minimum	44	1	36	1	21	1	28	1
Avg.	**2032.1**	**85.5**	**1276.1**	**85.5**	**1831.5**	**85.5**	**1306.6**	**85.5**
Std. Deviation	3456.7	162.7	1944.6	162.7	3122.0	162.7	2072.3	162.7
rel. Dev. [%]	170.1	190.2	152.4	190.2	170.5	190.2	158.6	190.2
Skewness	2.2	2.4	2.1	2.4	2.4	2.4	2.0	2.4

Table A.23: Results of Cost-Decomposition with the min-congestion formulation (Section 4.4.1)

	pure subgr.		Crowder		mod. CFM		Volume	
graph	time	subp.	time	subp.	time	subp.	time	subp.
RandRegular-0	1:31:37	91	7:47:27	477	1:50:24	107	7:45:13	447
RandRegular-1	18:10	19	1:37:58	99	19:21	19	1:19:54	71
RandRegular-2	1:26:04	91	7:39:05	483	1:48:16	109	7:22:46	427
RandRegular-3	10:08	9	49:11	47	10:13	9	33:33	29
RandRegular-4	1:29	1	18:59	19	1:37	1	9:41	9
RandRegular-5	15:01	15	1:02:44	61	14:37	13	52:01	45
RandRegular-6	22:24	21	1:42:43	105	24:36	23	1:21:39	79
RandRegular-7	1:22:22	83	6:51:21	415	1:35:14	91	7:06:02	413
RandRegular-8	1:09:50	69	6:11:52	373	1:16:18	73	5:41:41	329
RandRegular-9	2:29:54	155	10:35:20	661	2:49:36	173	11:06:13	687
RandRegular-10	46:37	47	4:32:35	273	54:08	51	4:02:56	227
RandRegular-11	33:30	37	2:11:44	141	41:20	43	2:07:28	129
RandRegular-12	28:43	31	2:31:36	157	33:37	33	2:19:45	133
RandRegular-13	47:54	51	3:08:45	201	50:53	55	2:56:50	181
RandRegular-14	7:35	7	42:01	43	8:11	7	24:45	21
RandRegular-15	52:27	51	4:16:58	253	1:06:27	63	4:00:43	233
RandRegular-16	43:15	45	3:21:01	211	50:17	51	2:57:07	177
RandRegular-17	7:38	7	49:21	51	8:19	7	34:45	33
RandRegular-18	15:13	15	1:11:44	73	17:35	17	50:22	45
RandRegular-19	7:06:06	431	28:50:53	1839	8:17:03	501	34:24:16	2135
Median	2010	37	9096	157	2480	43	8385	133
Maximum	25566	431	103853	1839	29823	501	123856	2135
Minimum	89	1	1139	19	97	1	581	9
Avg.	**3767.8**	**63.8**	**17319.9**	**299.1**	**4374.1**	**72.3**	**17633.0**	**292.5**
Std. Deviation	5594.2	94.5	22879.3	404.4	6534.8	110.0	27271.0	470.3
rel. Dev. [%]	148.5	148.2	132.1	135.2	149.4	152.2	154.7	160.8
Skewness	3.4	3.4	3.1	3.2	3.4	3.4	3.4	3.5

Table A.24: Results of Cost-Decomposition with the min-congestion formulation (Section 4.4.1)

	pure subgr.		Crowder		mod. CFM		Volume	
graph	time	subp.	time	subp.	time	subp.	time	subp.
Random-0	8:08	89	12:52	107	9:00	93	18:58	125
Random-1	4:14	11	12:01	33	5:42	13	25:24	77
Random-2	4:24	13	11:42	33	5:03	17	20:54	61
Random-3	28:25	97	1:01:02	195	30:02	101	1:52:34	391
Random-4	18:34	63	43:21	133	19:46	67	1:22:37	271
Random-5	43:05	149	1:27:04	283	51:32	173	2:56:52	661
Random-6	4:53	15	13:07	43	5:05	15	21:15	67
Random-7	33:34	105	1:14:06	233	36:54	113	2:04:55	437
Random-8	9:29	31	25:49	73	10:39	31	51:23	161
Random-9	24:01	75	55:50	177	29:36	91	1:43:29	365
Random-10	28:06	91	55:56	175	30:52	101	1:49:16	375
Random-11	2:09:44	1625	2:24:41	1633	2:08:17	1625	3:13:47	1717
Random-12	28:36	87	1:06:32	197	31:19	97	2:15:50	455
Random-13	3:33	11	9:35	31	3:52	11	17:22	53
Random-14	43:33	165	1:30:58	327	49:42	183	2:53:18	697
Random-15	13:17	39	30:07	87	12:55	41	57:27	187
Random-16	25:51	81	52:46	155	28:44	87	1:37:59	301
Random-17	1:01:30	183	1:53:33	323	1:10:15	207	3:41:23	697
Random-18	26:14	91	56:54	189	28:40	97	1:54:56	417
Random-19	52:41	181	1:43:59	345	58:59	211	3:30:49	853
Median	1551	87	3350	175	1724	93	6209	365
Maximum	7784	1625	8681	1633	7697	1625	13283	1717
Minimum	213	11	575	31	232	11	1042	53
Avg.	**1775.6**	**160.1**	**3365.8**	**238.6**	**1940.7**	**168.7**	**6211.4**	**418.4**
Std. Dev.	1733.1	349.1	2288.2	343.2	1775.8	348.4	4016.1	386.6
rel. Dev. [%]	97.6	218.0	68.0	143.9	91.5	206.5	64.7	92.4
Skewness	2.3	4.3	0.6	3.9	1.9	4.2	0.3	2.1

Table A.25: Results of Cost-Decomposition with the min-congestion formulation (Section 4.4.1)

	pure subgr.		Crowder		mod. CFM		Volume	
graph	time	subp.	time	subp.	time	subp.	time	subp.
RandW-0	27:23	175	21:44	139	27:53	179	57:53	571
RandW-1	24:53	143	18:39	99	20:13	115	54:04	509
RandW-2	18:36	103	14:16	77	15:57	85	40:35	331
RandW-3	23:38	137	19:28	103	20:50	113	53:49	479
RandW-4	34:43	225	26:56	145	30:02	189	1:09:33	679
RandW-5	32:45	211	25:51	155	30:15	191	1:03:55	635
RandW-6	19:46	113	16:21	85	18:20	97	46:43	413
RandW-7	15:03	79	11:43	57	14:43	73	34:46	283
RandW-8	31:42	219	24:04	143	29:45	195	1:03:19	603
RandW-9	20:50	125	15:49	87	17:39	95	42:43	403
RandW-10	23:13	141	19:01	109	22:34	131	49:47	459
RandW-11	29:10	195	24:08	151	27:07	179	59:37	595
RandW-12	43:36	313	38:59	273	43:45	305	1:35:13	981
RandW-13	15:00	83	12:29	67	14:25	77	33:50	283
RandW-14	34:39	223	28:53	173	32:44	211	1:15:16	717
RandW-15	35:34	249	31:37	199	35:28	243	1:28:02	917
RandW-16	32:28	215	26:09	161	31:35	205	1:04:47	635
RandW-17	32:17	215	25:25	165	27:44	179	1:09:15	705
RandW-18	41:54	295	35:17	229	37:54	255	1:23:42	851
RandW-19	32:45	209	28:00	169	31:34	197	1:13:25	707
Median	1750	195	1444	143	1664	179	3577	595
Maximum	2616	313	2339	273	2625	305	5713	981
Minimum	900	79	703	57	865	73	2030	283
Avg.	**1709.8**	**183.4**	**1394.5**	**139.3**	**1591.3**	**165.7**	**3660.7**	**587.8**
Std. Dev.	491.1	66.1	443.8	55.5	490.0	64.8	1026.6	197.5
rel. Dev. [%]	28.7	36.1	31.8	39.8	30.8	39.1	28.0	33.6
Skewness	-0.0	0.1	0.3	0.6	0.2	0.2	0.2	0.2

Table A.26: Results of Cost-Decomposition with the min-congestion formulation (Section 4.4.1)

	$\varepsilon = 0.025$		$\varepsilon = 0.05$		$\varepsilon = 0.1$		$\varepsilon = 0.25$		$\varepsilon = 0.5$		$\varepsilon = 0.75$	
graph	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.
RandPlan-0	3:00	15	5:35	15	3:05	15	1:34	23	1:40	189	5:43	2355
RandPlan-1	1:17:19	19	29:32	19	9:40	19	4:14	95	1:00	131	41	313
RandPlan-2	1:00:20	51	14:57	51	3:54	51	3:11	101	1:15	169	1:01	421
RandPlan-3	3:12	11	2:13	11	1:51	11	2:19	37	2:37	351	2:25	1199
RandPlan-4	1:27	7	56	7	1:37	7	1:36	23	1:04	127	57	361
RandPlan-5	19	1	13	1	18	1	3:27	79	55	139	53	583
RandPlan-6	2:29:15	443	41:18	443	16:11	443	7:57	491	4:40	957	4:45	2153
RandPlan-7	1:23:27	101	40:33	111	15:11	123	3:59	171	1:38	247	4:30	1729
RandPlan-8	46	5	30	5	1:27	5	2:30	51	1:21	217	42	463
RandPlan-9	1:51	1	33:57	15	9:01	17	3:51	79	49	95	36	241
RandPlan-10	16	1	9	1	6	1	3:07	89	46	101	20	207
RandPlan-11	4:37:12	581	2:24:56	585	48:41	597	38:51	1555	24:27	5199	17:49	13207
RandPlan-12	26:19	13	7:19	13	2:51	13	2:46	43	1:36	183	1:21	529
RandPlan-13	1:29	13	57	13	40	13	3:32	81	1:04	123	55	363
RandPlan-14	22	1	12	1	8	1	29	7	21	29	21	129
RandPlan-15	1:56	1	5:54	3	22:30	65	3:29	81	41	87	16	151
RandPlan-16	23:24:31	7597	7:07:58	7597	3:22:40	7611	1:53:55	8139	1:19:34	19303	1:24:27	65541
RandPlan-17	46:52	219	16:10	219	8:17	219	8:03	331	14:41	2111	40:42	21243
RandPlan-18	29:00	147	8:44	147	2:51	147	1:15	151	4:43	835	4:57	3103
RandPlan-19	1:59	5	1:19	5	1:37	5	28	5	1:12	133	1:21	513
Median	180	13	354	13	171	15	191	81	75	169	61	513
Maximum	84271	7597	25678	7597	12160	7611	6835	8139	4774	19303	5067	65541
Minimum	16	1	9	1	6	1	28	5	21	29	16	129
Avg.	**6512.6**	**461.6**	**2350.1**	**463.1**	**1057.8**	**468.2**	**631.6**	**581.6**	**438.2**	**1536.3**	**524.1**	**5740.2**
Std. Deviation	18755.2	1687.0	5832.5	1686.6	2701.6	1688.8	1540.8	1811.6	1079.2	4346.5	1207.4	15023.4
rel. Dev. [%]	288.0	365.5	248.2	364.2	255.4	360.7	243.9	311.5	246.3	282.9	230.4	261.7
Skewness	4.1	4.4	3.8	4.4	4.0	4.4	3.9	4.2	3.8	4.0	3.3	3.7

Table A.27: Results of the Approximation Algorithm without scaling and variable fixing (Section 4.4.2)

	$\varepsilon = 0.025$		$\varepsilon = 0.05$		$\varepsilon = 0.1$		$\varepsilon = 0.25$		$\varepsilon = 0.5$		$\varepsilon = 0.75$	
graph	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.
RandRegular-0	1:27:49	39	43:50	43	22:29	55	12:47	143	13:13	761	27:04	5253
RandRegular-1	9:17	7	13:24	9	6:17	13	3:29	39	3:54	221	9:14	1781
RandRegular-2	1:10:26	35	41:05	39	22:19	55	11:38	133	12:42	737	28:52	5663
RandRegular-3	17:11	5	5:32	5	3:52	7	2:09	25	2:30	139	7:03	1433
RandRegular-4	44	1	25	1	15	1	40	7	1:07	65	2:44	531
RandRegular-5	18:40	5	7:30	5	4:54	9	2:09	23	3:03	177	7:16	1395
RandRegular-6	30:52	9	20:14	13	10:13	21	4:27	51	5:31	415	10:41	2357
RandRegular-7	1:37:37	35	41:53	39	20:46	47	11:19	125	12:32	725	26:05	5061
RandRegular-8	1:15:45	27	40:31	35	18:01	39	10:09	115	10:58	631	23:54	4637
RandRegular-9	2:06:22	53	1:07:14	67	35:08	81	18:30	219	19:50	1181	43:42	9071
RandRegular-10	50:19	21	22:46	25	14:24	31	7:12	79	8:36	491	19:52	3835
RandRegular-11	55:18	13	35:11	23	20:03	39	8:55	113	8:32	617	16:43	4363
RandRegular-12	35:48	11	17:25	15	8:05	21	4:55	55	5:25	307	12:08	2365
RandRegular-13	42:33	25	29:47	25	18:30	41	8:41	109	9:46	651	22:15	5363
RandRegular-14	8:01	3	8:48	5	2:53	5	1:40	19	2:10	123	5:30	1053
RandRegular-15	45:23	21	28:40	25	17:50	35	9:59	121	9:15	569	19:10	3907
RandRegular-16	56:33	23	27:58	25	16:33	35	8:06	95	8:16	513	18:35	3967
RandRegular-17	18:33	5	6:37	5	4:12	7	2:13	27	3:17	231	6:46	1433
RandRegular-18	11:15	5	6:29	7	4:52	11	2:23	27	3:07	177	7:36	1463
RandRegular-19	6:24:15	143	2:48:37	161	1:22:13	191	42:39	493	39:48	2317	1:22:57	16325
Median	2553	13	1366	23	864	31	432	79	496	491	1003	3835
Maximum	23055	143	10117	161	4933	191	2559	493	2388	2317	4977	16325
Minimum	44	1	25	1	15	1	40	7	67	65	164	531
Avg.	**3728.1**	**24.3**	**1901.8**	**28.6**	**1001.5**	**37.2**	**522.0**	**100.9**	**550.6**	**552.4**	**1194.3**	**4062.8**
Std. Deviation	4972.6	31.4	2187.4	35.4	1068.3	41.8	555.5	107.4	518.6	503.1	1079.9	3575.1
rel. Dev. [%]	133.4	129.2	115.0	123.8	106.7	112.3	106.4	106.4	94.2	91.1	90.4	88.0
Skewness	3.4	3.1	3.0	3.0	2.8	2.8	2.8	2.8	2.5	2.4	2.5	2.3

Table A.28: Results of the Approximation Algorithm without scaling and variable fixing (Section 4.4.2)

	$\varepsilon = 0.025$		$\varepsilon = 0.05$		$\varepsilon = 0.1$		$\varepsilon = 0.25$		$\varepsilon = 0.5$		$\varepsilon = 0.75$	
graph	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.
Random-0	19:30	83	5:10	85	2:07	89	1:05	133	50	329	1:58	2338
Random-1	7:29	7	2:25	7	1:13	11	46	37	55	227	3:08	2861
Random-2	3:44	7	1:27	7	53	11	33	29	46	223	3:04	3049
Random-3	22:50	35	10:00	41	5:10	51	3:23	175	4:39	1333	15:43	18067
Random-4	7:47	27	7:36	31	3:27	39	1:47	95	2:30	679	9:22	10139
Random-5	38:27	53	16:45	61	8:10	81	4:39	235	6:37	1911	21:38	25935
Random-6	3:54	9	1:28	9	59	11	40	37	48	209	2:27	2477
Random-7	17:38	39	12:08	47	7:24	69	4:05	205	5:40	1675	16:33	20333
Random-8	12:02	15	4:26	17	2:31	25	1:26	73	2:11	591	7:52	8419
Random-9	18:59	37	9:14	43	4:18	51	2:28	133	3:23	929	12:21	13585
Random-10	25:45	31	9:41	35	4:47	45	2:29	121	3:23	835	12:38	12797
Random-11	9:55:20	1625	1:10:20	1625	25:46	1625	9:09	1639	6:30	2581	19:15	23321
Random-12	23:14	33	11:34	45	5:27	53	3:15	159	5:05	1333	18:37	20091
Random-13	50	5	40	5	39	7	22	21	28	127	1:46	1807
Random-14	36:13	65	18:21	73	8:55	95	5:00	289	6:29	2081	22:18	29621
Random-15	12:25	17	5:16	19	2:11	21	1:26	65	2:12	607	6:54	7349
Random-16	21:42	33	8:52	35	4:31	43	2:29	117	3:09	755	10:49	10469
Random-17	25:39	59	19:45	75	9:18	95	5:34	267	8:45	2329	31:39	36705
Random-18	19:14	31	8:01	35	3:41	37	2:19	121	4:11	1253	15:16	18563
Random-19	39:33	75	19:12	89	9:48	115	5:22	293	8:16	2483	27:05	32833
Median	1154	33	532	35	258	45	148	121	203	835	741	12797
Maximum	35720	1625	4220	1625	1546	1625	549	1639	525	2581	1899	36705
Minimum	50	5	40	5	39	7	22	21	28	127	106	1807
Avg.	**2856.8**	**114.3**	**727.0**	**119.2**	**333.8**	**128.7**	**174.8**	**212.2**	**230.3**	**1124.5**	**781.1**	**15038.0**
Std. Deviation	7764.2	356.3	895.4	355.4	334.8	353.6	132.5	346.4	155.1	809.5	522.4	10766.4
rel. Dev. [%]	271.8	311.7	123.2	298.1	100.3	274.8	75.8	163.3	67.3	72.0	66.9	71.6
Skewness	4.4	4.4	3.4	4.4	2.7	4.4	1.2	4.0	0.4	0.5	0.4	0.5

Table A.29: Results of the Approximation Algorithm without scaling and variable fixing (Section 4.4.2)

	$\varepsilon = 0.025$		$\varepsilon = 0.05$		$\varepsilon = 0.1$		$\varepsilon = 0.25$		$\varepsilon = 0.5$		$\varepsilon = 0.75$	
graph	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.
RandW-0	22:10	53	7:49	64	3:33	94	1:45	310	1:42	1980	2:01	11193
RandW-1	15:11	37	4:30	37	1:36	41	1:03	146	1:08	1077	1:33	7487
RandW-2	31:03	45	9:21	56	2:38	65	58	159	1:02	1094	1:25	7131
RandW-3	22:12	35	7:34	43	2:59	61	1:32	212	1:31	1523	1:49	9537
RandW-4	17:52	31	6:58	39	2:42	55	1:56	263	2:11	2666	2:24	13960
RandW-5	29:02	60	10:17	81	4:00	113	1:40	272	1:39	1781	2:04	11143
RandW-6	18:35	35	4:42	37	1:18	39	46	105	1:00	919	1:26	6612
RandW-7	17:50	33	4:30	35	1:12	39	37	87	48	696	1:10	5161
RandW-8	30:53	56	9:22	67	4:01	117	2:08	366	2:05	2707	2:22	15306
RandW-9	51:01	70	13:55	86	3:37	96	1:04	176	47	769	1:00	4474
RandW-10	11:40	35	5:08	47	2:37	63	1:18	199	1:19	1401	1:33	8202
RandW-11	22:51	37	7:19	47	3:16	78	1:48	262	1:41	1744	1:50	9609
RandW-12	21:31	47	9:55	64	5:11	116	3:04	530	2:53	3985	3:07	22318
RandW-13	36:21	57	9:51	75	2:36	92	58	182	58	1024	1:20	6710
RandW-14	23:24	45	10:11	68	3:48	84	2:16	331	2:26	3080	2:51	18846
RandW-15	30:56	80	12:25	97	5:25	139	2:47	514	2:39	3592	2:42	17797
RandW-16	25:51	47	9:30	57	5:24	120	2:27	434	1:53	2393	1:57	11524
RandW-17	22:49	37	11:20	65	3:57	93	1:46	301	1:28	1610	1:42	8947
RandW-18	27:36	52	14:37	82	7:02	150	3:27	633	2:35	3623	2:42	17780
RandW-19	16:53	41	8:47	54	5:32	122	2:44	449	2:21	3023	2:33	16221
Median	1369	45	561	57	213	92	105	263	99	1744	110	9609
Maximum	3061	80	877	97	422	150	207	633	173	3985	187	22318
Minimum	700	31	270	35	72	39	37	87	47	696	60	4474
Avg.	**1487.0**	**46.6**	**534.0**	**60.0**	**217.2**	**88.8**	**108.2**	**296.6**	**102.3**	**2034.3**	**118.5**	**11497.9**
Std. Deviation	525.7	13.1	175.3	18.0	91.6	33.3	48.1	150.9	39.3	1032.6	36.2	5075.9
rel. Dev. [%]	35.3	28.1	32.8	30.0	42.2	37.5	44.5	50.9	38.4	50.8	30.5	44.1
Skewness	1.4	1.1	0.2	0.3	0.4	0.0	0.4	0.7	0.2	0.5	0.3	0.6

Table A.30: Results of the Approximation Algorithm without scaling and variable fixing (Section 4.4.2)

graph	$\varepsilon = 0.025$		$\varepsilon = 0.05$		$\varepsilon = 0.1$		$\varepsilon = 0.25$		$\varepsilon = 0.5$		$\varepsilon = 0.75$	
	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.
RandPlan-0	3:00	15	4:27	10	1:27	10	31	5	14	6	19	14
RandPlan-1	17:31	8	6:10	6	1:38	9	25	6	22	7	12	9
RandPlan-2	10:18	37	2:54	37	1:02	37	28	35	11	13	10	11
RandPlan-3	3:11	11	2:13	11	1:34	9	15	3	8	5	8	10
RandPlan-4	1:26	7	56	7	1:24	7	28	5	22	9	14	13
RandPlan-5	19	1	13	1	18	1	11	1	4	3	7	4
RandPlan-6	44:23	403	16:10	401	8:46	381	2:49	271	1:08	132	50	119
RandPlan-7	18:24	91	4:35	88	1:52	86	40	74	51	69	19	33
RandPlan-8	46	5	30	5	1:13	4	13	3	5	3	5	5
RandPlan-9	1:51	1	5:39	1	1:21	1	16	2	7	3	6	4
RandPlan-10	16	1	9	1	6	1	10	1	3	2	2	2
RandPlan-11	1:35:04	511	27:33	457	8:49	425	3:29	309	57	82	12	60
RandPlan-12	14:09	9	4:11	9	1:11	7	19	5	8	8	7	11
RandPlan-13	1:28	13	56	13	40	13	35	13	8	5	8	12
RandPlan-14	21	1	12	1	7	1	11	1	4	2	4	5
RandPlan-15	1:55	1	5:14	1	1:15	1	10	1	4	2	3	2
RandPlan-16	12:28:06	6769	4:04:33	6705	1:36:14	6347	28:53	3705	13:08	1921	8:35	1196
RandPlan-17	21:48	209	9:39	207	4:46	197	2:20	161	44	55	53	129
RandPlan-18	12:52	139	4:39	137	1:50	137	55	131	32	53	11	28
RandPlan-19	1:59	5	1:19	5	1:07	2	13	2	16	6	10	7
Median	180	9	251	9	81	9	25	5	11	6	10	11
Maximum	44886	6769	14673	6705	5774	6347	1733	3705	788	1921	515	1196
Minimum	16	1	9	1	6	1	10	1	3	2	2	2
Avg.	**2997.3**	**411.9**	**1026.6**	**405.1**	**410.0**	**383.8**	**130.6**	**236.7**	**58.8**	**119.3**	**38.8**	**83.7**
Std. Deviation	9948.7	1503.0	3235.7	1488.8	1271.3	1409.2	381.5	821.6	172.8	425.6	112.9	264.3
rel. Dev. [%]	331.9	365.0	315.2	367.5	310.1	367.2	292.2	347.1	293.8	356.7	291.5	315.8
Skewness	4.3	4.4	4.4	4.4	4.4	4.4	4.3	4.4	4.4	4.4	4.4	4.3

Table A.31: Results of the Approximation Algorithm with scaling and without variable fixing (Section 4.4.2)

graph	$\varepsilon = 0.025$		$\varepsilon = 0.05$		$\varepsilon = 0.1$		$\varepsilon = 0.25$		$\varepsilon = 0.5$		$\varepsilon = 0.75$	
	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.
RandRegular-0	1:26:47	39	42:05	43	19:14	47	9:53	110	8:58	470	19:26	2452
RandRegular-1	9:14	7	13:05	9	6:27	13	2:34	31	2:34	138	6:22	784
RandRegular-2	1:07:58	35	40:01	37	19:00	51	9:16	107	8:25	461	19:19	2389
RandRegular-3	17:08	5	5:33	5	3:28	5	1:19	17	1:44	95	4:24	581
RandRegular-4	44	1	26	1	15	1	30	5	32	30	1:25	200
RandRegular-5	18:45	5	7:30	5	4:53	9	1:51	21	1:57	95	5:12	643
RandRegular-6	29:49	9	17:39	11	7:55	17	3:10	32	2:41	138	6:13	823
RandRegular-7	1:15:45	31	41:12	39	17:56	43	8:41	98	8:33	431	18:43	2301
RandRegular-8	1:15:19	27	39:14	35	17:00	37	8:05	87	7:39	409	17:13	2199
RandRegular-9	2:06:27	53	1:03:40	63	33:33	79	14:01	163	12:10	644	24:07	3076
RandRegular-10	47:24	21	22:13	25	14:00	31	6:04	69	6:02	310	15:10	1858
RandRegular-11	54:04	13	25:52	19	14:03	27	4:55	62	3:47	209	7:38	972
RandRegular-12	35:37	11	16:37	15	7:27	19	3:25	39	3:33	176	7:37	984
RandRegular-13	41:47	25	25:24	23	11:54	26	5:22	65	4:25	237	9:19	1221
RandRegular-14	7:59	3	8:53	5	2:53	5	1:17	15	1:30	73	3:56	478
RandRegular-15	45:05	21	23:06	21	16:05	33	6:55	81	6:11	307	13:45	1732
RandRegular-16	54:02	23	25:11	25	14:32	35	4:43	57	4:31	244	9:34	1239
RandRegular-17	15:46	3	6:33	5	4:11	7	1:34	19	1:30	79	4:36	560
RandRegular-18	11:14	5	6:28	7	4:32	9	1:59	21	2:11	113	5:22	668
RandRegular-19	6:14:31	141	2:45:47	159	1:16:01	182	33:58	391	27:55	1461	58:55	7210
Median	2507	13	1333	19	714	26	283	57	227	209	458	984
Maximum	22471	141	9947	159	4561	182	2038	391	1675	1461	3535	7210
Minimum	44	1	26	1	15	1	30	5	32	30	85	200
Avg.	**3586.2**	**23.9**	**1789.5**	**27.6**	**886.0**	**33.8**	**388.6**	**74.5**	**350.4**	**306.0**	**774.8**	**1618.5**
Std. Deviation	4838.5	31.0	2147.6	34.9	986.1	39.9	443.3	84.9	364.1	318.3	756.7	1545.2
rel. Dev. [%]	134.9	129.7	120.0	126.4	111.3	118.2	114.1	114.0	103.9	104.0	97.7	95.5
Skewness	3.4	3.1	3.1	3.1	3.0	2.9	3.0	3.0	2.7	2.7	2.7	2.7

Table A.32: Results of the Approximation Algorithm with scaling and without variable fixing (Section 4.4.2)

graph	ε = 0.025		ε = 0.05		ε = 0.1		ε = 0.25		ε = 0.5		ε = 0.75	
	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.
Random-0	18:01	76	4:01	78	1:52	80	39	91	26	117	43	377
Random-1	7:29	7	2:27	7	1:07	9	30	19	33	95	1:09	569
Random-2	3:24	5	1:29	7	49	9	24	19	24	75	51	433
Random-3	22:42	35	9:43	37	4:58	47	2:23	111	1:52	379	3:19	1864
Random-4	7:48	27	7:02	27	3:17	35	1:25	69	1:15	258	2:11	1209
Random-5	38:07	51	16:10	59	7:20	69	3:12	141	2:39	553	4:56	2948
Random-6	3:54	9	1:28	9	58	11	28	25	27	89	43	380
Random-7	17:39	39	11:38	45	6:37	55	2:38	113	2:12	448	3:32	2160
Random-8	12:04	15	4:12	15	2:05	19	53	39	58	182	1:53	981
Random-9	18:58	37	8:54	41	4:11	47	2:06	106	1:33	324	2:54	1687
Random-10	24:50	29	9:26	35	4:18	41	1:54	77	1:26	275	2:34	1474
Random-11	7:48:26	1517	52:04	1527	21:01	1531	7:53	1519	4:10	1653	5:08	3253
Random-12	23:09	33	10:44	37	5:23	49	2:27	111	2:17	431	4:13	2366
Random-13	50	5	40	5	39	7	17	13	15	59	25	245
Random-14	32:31	61	15:26	67	7:59	83	3:12	163	2:21	523	3:54	2512
Random-15	12:20	17	5:16	19	2:12	21	1:05	47	54	165	1:44	886
Random-16	21:04	33	8:39	33	4:14	39	1:59	84	1:39	301	3:08	1670
Random-17	31:39	59	18:38	67	9:05	87	4:03	171	3:37	706	6:44	4015
Random-18	19:17	31	7:58	33	3:42	37	1:35	79	1:32	330	3:25	2168
Random-19	41:07	71	18:56	82	8:48	95	4:05	195	3:09	671	6:24	3895
Median	1138	33	519	35	251	41	114	84	92	301	174	1670
Maximum	28106	1517	3124	1527	1261	1531	473	1519	250	1653	404	4015
Minimum	50	5	40	5	39	7	17	13	15	59	25	245
Avg.	**2475.9**	**107.8**	**644.5**	**111.5**	**301.8**	**118.5**	**129.4**	**159.6**	**101.0**	**381.7**	**179.5**	**1754.6**
Std. Deviation	6070.6	332.4	671.7	334.0	277.0	333.6	107.4	324.3	66.8	357.1	111.4	1161.1
rel. Dev. [%]	245.2	308.2	104.2	299.6	91.8	281.4	83.0	203.2	66.2	93.6	62.1	66.2
Skewness	4.4	4.4	2.8	4.4	2.3	4.4	1.8	4.3	0.7	2.5	0.5	0.5

Table A.33: Results of the Approximation Algorithm with scaling and without variable fixing (Section 4.4.2)

	$\varepsilon = 0.025$		$\varepsilon = 0.05$		$\varepsilon = 0.1$		$\varepsilon = 0.25$		$\varepsilon = 0.5$		$\varepsilon = 0.75$	
graph	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.
RandW-0	12:59	18	5:13	24	2:17	36	57	94	46	421	1:18	1956
RandW-1	9:08	13	3:25	17	1:13	23	34	52	36	285	59	1315
RandW-2	12:56	9	5:42	19	1:42	21	36	57	33	267	58	1284
RandW-3	13:51	11	4:46	13	1:47	17	46	61	44	364	1:16	1815
RandW-4	10:40	13	4:37	13	2:09	29	1:07	95	1:06	593	1:39	2479
RandW-5	10:34	15	5:34	27	2:29	48	52	93	53	460	1:25	2051
RandW-6	10:20	11	3:51	19	1:06	19	30	53	29	244	56	1268
RandW-7	8:41	9	2:16	9	44	11	25	39	25	187	47	1011
RandW-8	23:12	28	7:17	36	2:43	41	1:10	124	1:03	598	1:39	2643
RandW-9	24:56	18	7:16	23	2:20	33	35	48	24	177	37	800
RandW-10	5:51	13	2:39	13	1:47	24	43	64	43	368	1:08	1647
RandW-11	14:30	15	4:58	17	1:51	23	1:01	85	45	362	1:12	1718
RandW-12	16:22	27	7:36	35	4:19	67	1:49	196	1:29	938	2:11	3718
RandW-13	21:25	23	6:40	29	2:05	43	35	73	31	276	53	1267
RandW-14	15:54	21	7:40	31	2:43	37	1:27	135	1:16	731	1:56	3005
RandW-15	19:06	31	8:50	45	3:55	63	1:47	190	1:29	964	1:58	3245
RandW-16	13:40	19	6:01	23	3:14	44	1:20	146	1:02	599	1:26	2165
RandW-17	16:12	19	6:30	24	2:58	48	59	97	49	402	1:03	1482
RandW-18	20:45	21	9:25	33	4:10	55	1:50	184	1:24	820	1:55	3126
RandW-19	10:28	15	5:35	19	3:10	43	1:33	155	1:07	637	1:44	2740
Median	820	15	335	23	137	36	57	93	46	402	76	1815
Maximum	1496	31	565	45	259	67	110	196	89	964	131	3718
Minimum	351	9	136	9	44	11	25	39	24	177	37	800
Avg.	**874.5**	**17.4**	**347.6**	**23.4**	**146.1**	**36.2**	**61.8**	**102.0**	**52.7**	**484.6**	**81.0**	**2036.8**
Std. Deviation	310.2	6.3	115.3	9.3	59.2	15.5	27.7	50.2	20.9	240.2	26.7	825.7
rel. Dev. [%]	35.5	36.0	33.2	39.5	40.5	42.8	44.9	49.2	39.7	49.6	33.0	40.5
Skewness	0.5	0.6	-0.0	0.6	0.4	0.3	0.5	0.7	0.4	0.7	0.3	0.5

Table A.34: Results of the Approximation Algorithm with scaling and without variable fixing (Section 4.4.2)

	ε = 0.025		ε = 0.05		ε = 0.1		ε = 0.25		ε = 0.5		ε = 0.75	
graph	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.
RandPlan-0	2:47	15	4:42	10	1:27	9	25	5	9	5	17	12
RandPlan-1	19:56	8	6:22	6	1:42	7	18	7	8	7	14	10
RandPlan-2	10:21	37	2:56	37	1:01	37	29	35	10	11	5	7
RandPlan-3	2:39	11	1:56	11	1:37	9	13	3	4	3	5	5
RandPlan-4	49	7	34	7	1:35	7	23	5	14	5	15	8
RandPlan-5	12	1	8	1	5	1	3	1	3	2	1	2
RandPlan-6	44:52	403	14:56	403	8:43	381	2:55	273	1:05	131	46	118
RandPlan-7	6:44	91	4:39	88	1:47	86	38	74	30	66	12	25
RandPlan-8	46	5	28	5	1:20	4	13	3	3	3	1	4
RandPlan-9	35	1	19	1	12	1	12	1	3	2	2	2
RandPlan-10	8	1	4	1	3	1	1	1	2	1	1	2
RandPlan-11	1:36:11	511	28:09	455	10:40	421	3:50	319	28	44	9	45
RandPlan-12	14:21	9	4:07	9	1:00	7	16	5	3	5	8	8
RandPlan-13	1:04	13	40	13	28	13	33	13	7	4	9	7
RandPlan-14	9	1	5	1	2	1	4	1	3	1	5	2
RandPlan-15	7	1	3	1	2	1	1	1	2	1	1	1
RandPlan-16	12:46:11	6771	4:10:25	6727	2:36:00	6293	30:16	3739	13:12	1949	8:21	1170
RandPlan-17	21:56	209	9:52	207	4:51	197	2:19	161	49	51	34	105
RandPlan-18	12:59	139	4:44	137	1:51	137	58	131	32	53	6	14
RandPlan-19	1:43	5	1:11	5	1:07	3	11	2	8	3	7	5
Median	159	9	116	9	80	7	18	5	8	5	7	7
Maximum	45971	6771	15025	6727	9360	6293	1816	3739	792	1949	501	1170
Minimum	7	1	3	1	2	1	1	1	2	1	1	1
Avg.	**3013.5**	**411.9**	**1009.0**	**406.2**	**586.6**	**380.8**	**132.9**	**239.0**	**53.8**	**117.3**	**35.0**	**77.6**
Std. Deviation	10201.6	1503.5	3323.7	1493.7	2072.0	1397.2	401.1	829.2	174.6	432.4	110.3	259.2
rel. Dev. [%]	338.5	365.0	329.4	367.7	353.2	366.9	301.8	346.9	324.9	368.5	315.5	334.1
Skewness	4.3	4.4	4.4	4.4	4.4	4.4	4.3	4.4	4.4	4.4	4.4	4.4

Table A.35: Results of the Approximation Algorithm with scaling and variable fixing (Section 4.4.2)

	$\varepsilon = 0.025$		$\varepsilon = 0.05$		$\varepsilon = 0.1$		$\varepsilon = 0.25$		$\varepsilon = 0.5$		$\varepsilon = 0.75$	
graph	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.
RandRegular-0	47:57	35	27:47	35	12:09	41	4:53	53	2:56	104	3:56	262
RandRegular-1	2:55	7	6:21	7	7:00	7	1:20	13	46	25	1:14	74
RandRegular-2	42:52	33	22:48	33	21:42	37	4:32	53	2:53	98	3:46	253
RandRegular-3	5:17	3	4:58	5	5:26	5	44	7	32	17	43	50
RandRegular-4	29	1	16	1	23	1	6	1	9	6	17	17
RandRegular-5	18:34	5	5:20	5	4:33	5	52	9	35	25	50	58
RandRegular-6	23:48	9	12:59	9	5:42	11	1:37	17	57	31	1:27	76
RandRegular-7	32:52	29	24:01	29	11:18	35	4:00	45	2:38	85	3:52	253
RandRegular-8	47:04	25	25:59	27	11:36	29	3:13	37	2:03	76	3:05	208
RandRegular-9	1:33:26	53	33:43	53	21:32	53	6:37	82	4:19	147	5:42	380
RandRegular-10	8:16	17	5:11	17	8:37	21	2:46	29	1:42	55	2:20	157
RandRegular-11	28:30	11	12:14	11	7:13	16	2:33	28	1:31	51	2:06	125
RandRegular-12	22:31	11	7:34	11	3:41	13	1:43	19	1:24	38	2:15	97
RandRegular-13	33:40	23	18:12	23	8:24	23	2:51	30	1:38	62	2:12	167
RandRegular-14	2:04	3	5:37	3	2:15	3	32	5	22	13	39	39
RandRegular-15	26:01	19	14:46	19	8:55	22	2:58	33	2:10	71	2:33	174
RandRegular-16	31:05	19	18:59	23	8:58	23	2:24	30	1:53	67	2:33	157
RandRegular-17	4:26	3	2:56	3	2:45	5	43	7	29	16	39	49
RandRegular-18	2:57	5	1:50	5	2:40	7	59	9	33	21	57	62
RandRegular-19	4:13:52	131	1:49:33	137	52:35	147	14:59	177	9:33	337	12:23	803
Median	1428	11	734	11	433	16	144	28	91	51	132	125
Maximum	15232	131	6573	137	3155	147	899	177	573	337	743	803
Minimum	29	1	16	1	23	1	6	1	9	6	17	17
Avg.	**2185.8**	**22.1**	**1083.2**	**22.8**	**622.2**	**25.2**	**181.1**	**34.2**	**117.2**	**67.2**	**160.4**	**173.1**
Std. Deviation	3349.5	29.0	1418.0	30.1	685.6	32.1	196.6	39.3	125.0	73.3	160.7	175.9
rel. Dev. [%]	153.2	131.2	130.9	132.1	110.2	127.3	108.6	114.8	106.7	108.9	100.2	101.6
Skewness	3.4	3.1	3.3	3.1	2.9	3.1	2.8	2.8	2.8	2.9	2.7	2.6

Table A.36: Results of the Approximation Algorithm with scaling and variable fixing (Section 4.4.2)

graph	$\varepsilon = 0.025$		$\varepsilon = 0.05$		$\varepsilon = 0.1$		$\varepsilon = 0.25$		$\varepsilon = 0.5$		$\varepsilon = 0.75$	
	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.
Random-0	18:07	76	3:46	76	1:25	78	32	81	12	48	12	87
Random-1	7:09	7	1:57	7	34	7	15	9	10	21	11	61
Random-2	1:08	5	55	5	31	7	12	11	10	23	11	63
Random-3	16:13	33	6:30	35	3:13	37	1:09	49	47	115	53	320
Random-4	4:43	27	4:58	27	1:51	27	43	35	28	69	30	177
Random-5	28:07	49	11:22	49	4:45	55	1:43	73	1:03	158	1:15	502
Random-6	3:33	9	1:07	9	41	9	15	11	12	31	10	59
Random-7	10:31	37	6:22	39	4:08	41	1:33	65	55	141	1:05	460
Random-8	9:01	11	3:19	15	1:26	17	32	23	21	47	21	123
Random-9	12:42	35	6:05	37	2:32	39	1:03	49	41	103	40	247
Random-10	17:29	27	7:29	27	2:51	33	1:01	44	35	79	34	205
Random-11	8:17:26	1527	45:53	1531	19:14	1531	8:37	1517	4:51	1556	2:59	1690
Random-12	13:33	31	5:52	31	2:59	37	1:10	49	47	111	49	285
Random-13	24	5	16	5	25	6	11	9	7	19	7	48
Random-14	25:10	58	11:29	60	4:56	64	1:56	91	1:03	183	1:17	534
Random-15	10:45	17	4:03	17	1:23	17	32	21	22	49	24	136
Random-16	13:19	29	5:27	29	2:29	33	1:00	39	37	79	38	217
Random-17	19:23	55	10:21	59	5:33	63	2:11	89	1:13	169	1:25	522
Random-18	14:34	29	6:01	29	2:48	33	50	39	33	87	41	283
Random-19	26:24	71	11:42	71	5:49	81	2:16	109	1:17	200	1:24	548
Median	799	29	352	29	152	33	60	44	35	79	38	217
Maximum	29846	1527	2753	1531	1154	1531	517	1517	291	1556	179	1690
Minimum	24	5	16	5	25	6	11	9	7	19	7	48
Avg.	**2249.1**	**106.9**	**464.7**	**107.9**	**208.7**	**110.8**	**83.0**	**120.7**	**49.2**	**164.4**	**47.3**	**328.4**
Std. Deviation	6512.7	334.9	577.3	335.6	244.3	335.1	109.3	330.0	60.9	332.3	40.3	364.3
rel. Dev. [%]	289.6	313.3	124.2	311.1	117.1	302.5	131.6	273.5	123.8	202.1	85.3	110.9
Skewness	4.4	4.4	3.6	4.4	3.3	4.4	3.6	4.4	3.6	4.3	1.9	3.0

Table A.37: Results of the Approximation Algorithm with scaling and variable fixing (Section 4.4.2)

	$\varepsilon = 0.025$		$\varepsilon = 0.05$		$\varepsilon = 0.1$		$\varepsilon = 0.25$		$\varepsilon = 0.5$		$\varepsilon = 0.75$	
graph	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.
RandW-0	9:09	15	4:38	19	2:28	20	42	53	33	177	42	603
RandW-1	7:32	10	2:56	12	1:01	14	27	29	26	116	28	359
RandW-2	10:13	7	3:29	9	1:35	18	26	30	20	103	28	353
RandW-3	11:06	7	3:35	9	1:29	16	31	31	29	151	39	545
RandW-4	10:22	13	4:37	15	1:45	21	45	52	40	207	46	633
RandW-5	10:07	13	4:00	15	1:46	24	43	55	38	192	43	582
RandW-6	9:55	9	2:33	9	56	13	19	22	22	103	32	414
RandW-7	8:05	7	2:04	9	40	9	17	17	16	70	20	240
RandW-8	20:58	23	6:41	28	2:11	33	55	76	39	214	47	672
RandW-9	21:54	13	6:34	16	1:52	23	32	40	19	100	22	267
RandW-10	4:51	12	2:07	13	1:10	13	34	40	28	144	32	430
RandW-11	14:35	15	4:34	15	1:46	22	42	46	33	156	33	450
RandW-12	13:34	27	6:28	29	3:01	41	1:21	106	1:00	370	1:09	1114
RandW-13	15:03	11	4:28	14	1:28	21	25	33	20	105	24	320
RandW-14	15:12	21	6:35	27	2:37	33	59	65	46	249	58	821
RandW-15	14:21	23	7:28	35	2:41	37	1:09	91	49	287	1:01	886
RandW-16	12:33	17	5:09	19	2:47	38	56	66	43	241	48	684
RandW-17	13:15	14	5:41	21	2:28	35	49	71	36	202	40	554
RandW-18	15:45	17	7:08	23	3:21	37	1:14	87	53	313	1:01	929
RandW-19	7:02	12	3:42	13	3:32	29	57	70	47	265	52	788
Median	666	13	274	15	106	22	42	52	33	177	40	554
Maximum	1314	27	448	35	212	41	81	106	60	370	69	1114
Minimum	291	7	124	9	40	9	17	17	16	70	20	240
Avg.	**736.6**	**14.3**	**283.4**	**17.5**	**121.7**	**24.9**	**44.1**	**54.0**	**34.9**	**188.2**	**41.2**	**582.2**
Std. Deviation	261.4	5.6	101.6	7.5	48.9	9.8	18.3	24.6	12.4	81.1	14.1	237.8
rel. Dev. [%]	35.5	39.5	35.9	43.1	40.2	39.4	41.3	45.5	35.7	43.1	34.1	40.9
Skewness	0.6	0.7	0.0	0.8	0.2	0.2	0.4	0.4	0.2	0.6	0.3	0.5

Table A.38: Results of the Approximation Algorithm with scaling and variable fixing (Section 4.4.2)

	p2				p4			
		J	S	IN		J	S	IN
DeBruijn 6								
dist.	-0.53	-0.51	-0.46	-0.42	-0.55	-0.65	-0.70	-0.62
MaxFlow	-0.12	0.11	-0.03	0.02	-0.15	0.04	-0.07	0.00
rel MaxFlow	0.58	0.65	0.62	0.45	0.52	0.39	0.44	0.38
rel. dist.	-0.83	-0.82	-0.80	-0.76	-0.72	-0.79	-0.83	-0.81
log(rel. dist.)	**-0.88**	**-0.88**	**-0.83**	**-0.83**	**-0.87**	**-0.92**	**-0.91**	**-0.92**
Shuffle-Exchange 6								
dist.	-0.56	-0.52	-0.46	-0.48	-0.68	-0.64	-0.64	-0.61
MaxFlow	-0.07	0.09	-0.02	0.02	-0.08	0.06	-0.05	0.01
rel MaxFlow	0.65	0.65	0.78	0.66	0.43	0.54	0.55	0.48
rel. dist.	-0.80	-0.81	-0.88	-0.85	-0.80	-0.74	-0.76	-0.80
log(rel. dist.)	**-0.92**	**-0.88**	**-0.91**	**-0.89**	**-0.93**	**-0.92**	**-0.93**	**-0.93**
6x10 Grid								
dist.	-0.60	-0.48	-0.57	-0.42	-0.67	-0.52	-0.60	-0.49
MaxFlow	0.07	0.04	-0.01	0.03	-0.02	-0.07	-0.07	-0.07
rel MaxFlow	0.82	0.85	0.63	0.76	0.58	0.81	0.67	0.73
rel. dist.	-0.91	-0.82	-0.68	-0.80	-0.78	-0.71	-0.79	-0.65
log(rel. dist.)	**-0.93**	**-0.92**	**-0.84**	**-0.87**	**-0.92**	**-0.93**	**-0.94**	**-0.89**
ex36								
dist.	0.55	0.28	-0.16	-0.57	-0.37	-0.53	-0.64	-0.01
MaxFlow	0.08	-0.13	-0.06	0.21	-0.31	-0.06	-0.21	-0.17
rel MaxFlow	**0.23**	-0.10	-0.24	-0.21	-0.11	0.09	-0.11	-0.08
rel. dist.	0.10	0.10	-0.15	-0.50	-0.53	-0.65	-0.71	-0.71
log(rel. dist.)	0.16	**0.07**	**-0.17**	**-0.58**	**-0.54**	**-0.74**	**-0.73**	**-0.73**
random planar								
dist.	-0.42	-0.44	-0.41	-0.36	-0.45	-0.47	-0.47	-0.39
MaxFlow	0.17	0.18	0.16	0.19	0.06	0.11	0.10	0.10
rel MaxFlow	0.62	0.79	0.73	0.77	0.73	0.70	0.70	0.79
rel. dist.	-0.72	-0.78	-0.46	-0.83	-0.71	-0.63	-0.72	-0.75
log(rel. dist.)	**-0.78**	**-0.87**	**-0.76**	**-0.84**	**-0.85**	**-0.83**	**-0.85**	**-0.84**

Table A.39: Pearson’s correlation coefficient for the split-prediction with different graphs and predictors (Section 5.4.1)

	p2				p4			
		J	S	IN		J	S	IN
DeBruijn 6								
dist.	0.64	0.54	0.39	0.41	0.81	0.62	0.76	0.51
max11UB	0.80	0.60	0.51	0.23	0.86	0.76	0.80	0.63
rel.dist.	0.81	0.85	0.79	0.77	0.81	0.83	0.88	0.82
log(rel.dist.)	0.70	0.77	0.71	0.63	0.76	0.77	0.81	0.75
rel.max11UB	0.84	0.86	0.85	0.87	0.84	0.85	0.82	0.84
adv.rel.max11UB	0.90	0.89	0.86	0.86	0.89	0.90	0.86	0.86
Shuffle-Exchange 6								
dist.	0.64	0.37	0.36	0.28	0.80	0.62	0.71	0.54
max11UB	0.83	0.37	0.42	0.23	0.89	0.67	0.73	0.57
rel.dist.	0.83	0.80	0.89	0.87	0.85	0.84	0.79	0.84
log(rel.dist.)	0.63	0.63	0.77	0.72	0.76	0.75	0.74	0.74
rel.max11UB	0.92	0.89	0.92	0.91	0.82	0.73	0.70	0.80
adv.rel.max11UB	0.93	0.92	0.92	0.90	0.90	0.86	0.85	0.81
6x10 Grid								
dist.	0.65	0.58	0.64	0.38	0.79	0.69	0.75	0.62
max11UB	0.89	0.72	0.71	0.50	0.84	0.70	0.80	0.63
rel.dist.	0.90	0.82	0.61	0.81	0.85	0.88	0.84	0.73
log(rel.dist.)	0.87	0.67	0.60	0.67	0.80	0.79	0.79	0.70
rel.max11UB	0.94	0.90	0.84	0.90	0.71	0.85	0.69	0.81
adv.rel.max11UB	0.95	0.92	0.82	0.91	0.85	0.90	0.87	0.82
ex36								
dist.	0.46	0.31	0.42	-0.07	0.59	0.52	0.56	-0.04
max11UB	0.68	0.23	0.51	0.12	0.66	0.11	0.64	0.25
rel.dist.	0.46	0.46	0.55	0.49	0.57	0.49	0.58	0.29
log(rel.dist.)	0.48	0.46	0.54	0.47	0.57	0.50	0.58	0.28
rel.max11UB	0.23	0.38	0.58	0.58	0.69	0.06	0.67	0.64
adv.rel.max11UB	0.63	0.42	0.63	0.60	0.70	0.07	0.70	0.66
random planar								
dist.	0.31	0.35	0.25	0.12	0.32	0.29	0.28	0.24
max11UB	0.51	0.45	0.32	0.28	0.67	0.66	0.71	0.56
rel.dist.	0.73	0.74	0.34	0.67	0.68	0.68	0.58	0.70
log(rel.dist.)	0.69	0.62	0.52	0.58	0.66	0.65	0.56	0.65
rel.max11UB	0.75	0.72	0.64	0.88	0.66	0.61	0.82	0.80
adv.rel.max11UB	0.82	0.77	0.68	0.89	0.80	0.79	0.84	0.88

Table A.40: Pearson's correlation coefficient for the join-prediction with different graphs and predictors (Section 5.4.2)

graph	rek F.		$\alpha_1 = 0.5$		$\alpha_1 = 1$		$\alpha_1 = 2$		$\alpha_1 = 4$		$\alpha_1 = 8$	
	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.
Random-30-0.5-0	291	417	318	523	324	529	308	467	301	429	309	441
Random-30-0.5-1	99	121	95	135	102	147	105	133	104	131	97	119
Random-30-0.5-2	249	343	283	455	285	439	250	361	276	369	243	329
Random-30-0.5-3	197	273	233	381	228	355	226	333	216	301	221	305
Random-30-0.5-4	44	55	46	61	41	49	34	43	34	41	34	39
Random-30-0.5-5	62	71	74	101	65	89	60	73	63	73	63	73
Random-30-0.5-6	146	205	162	251	148	225	158	225	150	205	154	215
Random-30-0.5-7	429	585	429	657	459	675	435	621	415	553	454	605
Random-30-0.5-8	58	71	61	95	67	101	60	75	54	65	51	63
Random-30-0.5-9	272	373	320	457	303	447	276	389	258	349	288	381
Random-30-0.5-10	54	63	57	79	45	61	59	69	51	59	51	59
Random-30-0.5-11	578	767	643	959	641	971	592	819	603	809	554	769
Random-30-0.5-12	6	9	6	9	6	9	6	9	6	9	6	9
Random-30-0.5-13	128	161	143	207	153	211	132	171	140	181	144	181
Random-30-0.5-14	313	447	345	565	329	521	318	477	307	439	313	437
Random-30-0.5-15	228	313	245	381	243	411	224	343	217	295	237	315
Random-30-0.5-16	75	95	80	107	66	85	85	109	75	95	82	103
Random-30-0.5-17	148	193	156	229	188	255	160	199	162	205	186	235
Random-30-0.5-18	94	123	106	149	100	135	99	131	104	131	99	123
Random-30-0.5-19	126	153	146	195	158	217	133	161	132	161	131	153
Median	128	161	146	207	153	217	133	171	140	181	144	181
Maximum	578	767	643	959	641	971	592	819	603	809	554	769
Minimum	6	9	6	9	6	9	6	9	6	9	6	9
Avg.	**179.8**	**241.9**	**197.4**	**299.8**	**197.6**	**296.6**	**186.0**	**260.4**	**183.4**	**245.0**	**185.8**	**247.7**
rel. Std. Dev.[%]	77.9	79.9	77.6	79.3	78.5	81.1	76.9	79.9	77.9	80.2	75.6	79.4
Skewness	-1.33	-1.20	-1.31	-1.16	-1.28	-1.19	-1.29	-1.13	-1.36	-1.28	-1.09	-1.11

Table A.41: Results of the different branching-selection strategies with random graphs and bisection (Section 5.4.3)

graph	rek F.		$\alpha_1 = 0.5$		$\alpha_1 = 1$		$\alpha_1 = 2$		$\alpha_1 = 4$		$\alpha_1 = 8$	
	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.
RandPlan-100-1-0	28	7	45	9	24	5	16	3	27	7	78	21
RandPlan-100-1-1	244	123	266	169	267	171	262	155	278	181	245	139
RandPlan-100-1-2	77	21	298	167	231	105	229	105	235	111	394	165
RandPlan-100-1-3	16	3	13	3	13	3	13	3	16	3	16	3
RandPlan-100-1-4	46	11	66	19	138	45	91	27	44	9	82	21
RandPlan-100-1-5	13	3	12	3	12	3	12	3	12	3	12	3
RandPlan-100-1-6	98	169	56	135	50	135	50	135	222	195	226	149
RandPlan-100-1-7	273	135	125	161	126	161	142	157	441	265	309	269
RandPlan-100-1-8	33	9	227	141	227	141	227	141	211	107	27	7
RandPlan-100-1-9	31	7	242	171	242	171	239	147	237	139	232	155
RandPlan-100-1-10	2	1	2	1	2	1	2	1	2	1	2	1
RandPlan-100-1-11	242	249	353	233	350	233	203	169	142	169	120	223
RandPlan-100-1-12	34	9	243	127	233	145	232	139	423	253	65	17
RandPlan-100-1-13	22	5	211	91	230	91	229	95	240	107	65	17
RandPlan-100-1-14	10	3	162	55	162	55	162	55	190	71	10	3
RandPlan-100-1-15	186	91	222	109	231	137	227	135	181	109	10	3
RandPlan-100-1-16	537	1245	2197	1919	1634	1573	1422	1401	1356	1311	1063	1481
RandPlan-100-1-17	1434	1457	1877	1815	1772	1755	2316	2429	1089	757	752	609
RandPlan-100-1-18	198	87	219	109	219	109	220	109	219	109	264	131
RandPlan-100-1-19	3	1	3	1	3	1	3	1	3	1	3	1
Median	34	9	211	109	219	109	203	109	211	109	78	21
Maximum	1434	1457	2197	1919	1772	1755	2316	2429	1356	1311	1063	1481
Minimum	2	1	2	1	2	1	2	1	2	1	2	1
Avg.	**176.3**	**181.8**	**341.9**	**271.9**	**308.3**	**252.0**	**314.9**	**270.5**	**278.4**	**195.4**	**198.8**	**170.9**
rel. Std. Dev.[%]	179.8	218.4	168.7	197.3	154.5	189.0	173.1	212.5	122.6	155.4	134.4	194.7
Skewness	-3.39	-2.78	-2.75	-2.80	-2.65	-2.80	-3.07	-3.24	-2.29	-2.93	-2.18	-3.36

Table A.42: Results of the different branching-selection strategies with planar Graphs and bisection (Section 5.4.3)

	rek F.		$\alpha_1 = 0.5$		$\alpha_1 = 1$		$\alpha_1 = 2$		$\alpha_1 = 4$		$\alpha_1 = 8$	
graph	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.
RandRegular-100-4-0	1097	263	942	291	933	275	1032	275	1227	303	1373	341
RandRegular-100-4-1	142	33	140	35	88	21	72	17	196	47	161	37
RandRegular-100-4-2	793	189	932	297	872	273	743	207	1294	321	1151	287
RandRegular-100-4-3	41	9	50	13	51	13	55	13	103	23	125	29
RandRegular-100-4-4	10	3	16	5	16	5	10	3	13	3	13	3
RandRegular-100-4-5	70	17	51	13	44	11	51	13	72	17	64	15
RandRegular-100-4-6	143	33	279	127	275	121	272	105	218	51	215	49
RandRegular-100-4-7	711	167	884	249	714	211	563	131	1052	271	983	251
RandRegular-100-4-8	752	183	648	191	661	199	701	169	812	199	805	201
RandRegular-100-4-9	1064	255	1772	597	1621	535	1517	429	1691	421	1676	423
RandRegular-100-4-10	490	115	420	119	350	95	622	149	792	199	822	205
RandRegular-100-4-11	319	69	357	157	319	153	302	155	452	101	588	133
RandRegular-100-4-12	255	61	320	103	270	85	154	37	252	61	304	75
RandRegular-100-4-13	296	67	386	157	360	165	392	153	468	109	588	141
RandRegular-100-4-14	56	13	58	15	37	9	30	7	73	17	66	15
RandRegular-100-4-15	462	107	405	115	277	67	475	113	573	139	621	149
RandRegular-100-4-16	291	65	320	125	317	119	300	117	535	127	472	111
RandRegular-100-4-17	83	19	91	23	87	23	63	15	92	21	110	25
RandRegular-100-4-18	89	21	158	47	137	39	58	13	145	35	153	37
RandRegular-100-4-19	3370	829	6232	2015	4709	1485	3252	859	4981	1267	6521	1683
Median	291	65	320	119	277	95	300	113	452	101	472	111
Maximum	3370	829	6232	2015	4709	1485	3252	859	4981	1267	6521	1683
Minimum	10	3	16	5	16	5	10	3	13	3	13	3
Avg.	**526.7**	**125.9**	**723.0**	**234.7**	**606.9**	**195.2**	**533.2**	**149.0**	**752.0**	**186.6**	**840.5**	**210.5**
rel. Std. Dev.[%]	138.9	142.9	184.1	183.5	167.8	164.7	137.1	130.1	143.1	146.9	164.5	169.8
Skewness	-3.12	-3.18	-3.81	-3.80	-3.53	-3.49	-2.81	-2.69	-3.23	-3.27	-3.70	-3.74

Table A.43: Results of the different branching-selection strategies with regular graphs and bisection (Section 5.4.3)

graph	MVarMC		[KRC00]		[BCR97]	
	time	subp.	time	subp.	time	subp.
BCR-10x2g	<1	3	1	1	6	1
BCR-5x6g	<1	6	3	1	61	1
BCR-2x16g	<1	3	4	1	108	3
BCR-18x2g	<1	3	2	1	262	3
BCR-2x19g	<1	12	55	49	788	3
BCR-5x8g	<1	1	2	1	851	7
BCR-3x14g	<1	3	18	5	1256	5
BCR-5x10g	<1	1	9	1	2843	3
BCR-6x10g	1	7	319	57	17994	31
BCR-7x10g	<1	5	557	61	n.a.	

Table A.44: Results on the exact bisection width of the BCR-grid graphs (Section 5.5)

graph	MVarMC		[KRC00]		[BCR97]	
	time	subp.	time	subp.	time	subp.
BCR-4x5t	<1	1	1	1	3	1
BCR-6x5t	<1	3	3	1	45	1
BCR-8x5t	<1	1	6	1	494	7
BCR-21x2t	<1	1	5	1	1061	3
BCR-23x2t	<1	13	125	33	2788	3
BCR-4x12t	<1	1	17	3	3012	5
BCR-5x10t	1	3	6	1	2031	13
BCR-10x6t	2	3	350	43	6517	3
BCR-7x10t	<1	1	572	47	28297	33
BCR-10x8t	4	3	944	45	n.a.	

Table A.45: Results on the exact bisection width of the BCR-tori graphs (Section 5.5)

graph	MVarMC		[KRC00]		[BCR97]	
	time	subp.	time	subp.	time	subp.
BCR-2x10m	<1	1	1	1	3	1
BCR-6x5m	<1	1	1	1	28	1
BCR-2x17m	22	29	29	21	235	3
BCR-10x4m	2	1	2	1	315	1
BCR-5x10m	7	2	2	1	3212	5
BCR-4x13m	225	36	34	5	5702	7
BCR-13x4m	172	26	34	5	5105	5
BCR-9x6m	201	26	12	1	n.a.	
BCR-10x6m	13	2	8	1	12920	9
BCR-10x7m	21	3	14	1	127297	13

Table A.46: Results on the exact bisection width of the BCR-mixed-grid graphs (Section 5.5)

graph	MVarMC		[KRC00]		[BCR97]		[FMdS$^+$98]	
	time	subp.	time	subp.	time	subp.	time	subp.
BCR-m4.i	<1	1	1	1	56	1	5	1
BCR-ma.i	<1	1	3	1	3218	29	n.a.	
BCR-me.i	<1	1	4	1	6505	37	n.a.	
BCR-m6.i	<1	1	37	1	5737	55	103	1
BCR-mb.i	<1	1	28	1	3772	33	n.a.	
BCR-mc.i	<1	1	46	1	27712	53	n.a.	
BCR-md.i	<1	1	29	1	86140	57	n.a.	
BCR-mf.i	<1	1	24	1	14659	47	n.a.	
BCR-m1.i	4	3	1095	15	97271	101	n.a.	
BCR-m8.i	<1	1	321	1	n.a.		851	1

Table A.47: Results on the exact bisection width of the BCR-real-world graphs (Section 5.5)

graph	MVarMC		[KRC00]	
	time	subp.	time	subp.
DB-5	<1	1	<6	<3
DB-6	<1	1	<469	<55
DB-7	3	1	32,000	195
DB-8	6,961	148	n.a.	
ex36a	250	279	3	1
ex36b	122	125	5	1
ex36c	98	101	2	1
cd30	<1	1	2	1
cd45	<1	3	7	1
cd47a	<1	3	10	1
cd47b	1	18	112	35
cd61	1	8	20	1

Table A.48: Comparison with [KRC00] with respect to the graph bisection problem (Section 5.5)

n	120		130		140		150		160		170	
graph	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.	time	subp.
0	12	3	33	8	20	5	56	11	41	9	40	6
1	26	16	43	21	76	34	47	10	27	3	4	1
2	40	15	24	7	35	12	76	20	215	63	335	71
3	1	1	2	1	2	1	22	3	11	1	41	11
4	15	5	27	11	76	28	67	15	64	13	54	10
5	39	52	2	1	6	1	7	1	3	1	4	1
6	0	0	198	662	248	795	307	1018	928	3243	17	6
7	12	3	37	15	29	11	44	12	124	40	107	20
8	17	9	28	9	188	191	147	152	263	142	536	280
9	13	3	1	1	2	1	2	1	3	1	4	1
Median	13	3	27	8	29	11	47	11	41	9	40	6
Maximum	40	52	198	662	248	795	307	1018	928	3243	536	280
Minimum	0	0	1	1	2	1	2	1	3	1	4	1
Avg.	**17.5**	**10.7**	**39.5**	**73.6**	**68.2**	**107.9**	**77.5**	**124.3**	**167.9**	**351.6**	**114.2**	**40.7**

Table A.49: Solving bisection problems on maximal RandPlan-graphs, $|E| = 3n - 6$ (Section 5.5)

n	80		90		100		110	
graph	time	subp.	time	subp.	time	subp.	time	subp.
0	0	1	327	107	505	131	24	5
1	1	1	250	85	73	19	1134	229
2	16	7	46	13	510	133	539	105
3	1	1	0	1	33	7	266	55
4	0	1	41	13	9	3	84	17
5	0	1	279	89	35	9	517	105
6	0	1	523	179	73	19	99	19
7	48	21	186	61	481	125	627	125
8	0	1	56	19	396	101	177	35
9	0	1	308	107	735	185	465	97
Median	0	1	186	61	73	19	266	55
Maximum	48	21	523	179	735	185	1134	229
Minimum	0	1	0	1	9	3	24	5
Avg.	**6.6**	**3.6**	**201.6**	**67.4**	**285.0**	**73.2**	**393.2**	**79.2**

Table A.50: Solving bisection problems on RandRegular-graphs of degree 4, $|E| = 2n$ (Section 5.5)

n	60		70		80		90	
graph	time	subp.	time	subp.	time	subp.	time	subp.
0	0	1	0	1	0	1	588	189
1	1	3	1	1	94	65	35	11
2	0	1	0	1	6	3	354	109
3	3	5	2	3	97	89	111	37
4	1	3	0	1	0	1	1268	419
5	4	5	0	1	76	35	26	9
6	0	1	2	3	99	43	50004	25781
7	4	16	17	11	5	3	32	11
8	0	1	0	1	28	11	155	53
9	10	18	8	7	16	7	246	83
Median	1	3	0	1	16	7	155	53
Maximum	10	18	17	11	99	89	50004	25781
Minimum	0	1	0	1	0	1	26	9
Avg.	**2.3**	**5.4**	**3.0**	**3.0**	**42.1**	**25.8**	**5281.9**	**2670.2**

Table A.51: Solving bisection problems on Random-graphs with edge-probability 0.05, $|E| \approx \frac{n^2}{40}$ (Section 5.5)