Multicommodity Flow Approximation used for Exact Graph Partitioning ⋆

Meinolf Sellmann†, Norbert Sensen‡, Larissa Timajev‡

†Cornell University
Department of Computer Science
4130 Upson Hall
Ithaca, NY 14853, U.S.A.
sello@cs.cornell.edu

‡University of Paderborn
Faculty of Computer Science, Electrical Engineering, and Mathematics
Fürstenallee 11
33102 Paderborn, Germany
{sensen,timajev}@upb.de

Abstract. We present a fully polynomial-time approximation scheme for a multicommodity flow problem that yields lower bounds of the graph bisection problem. We compare the approximation algorithm with Lagrangian relaxation based cost-decomposition approaches and linear programming software when embedded in an exact branch&bound approach for graph bisection. It is shown that the approximation algorithm is clearly superior in this context. Furthermore, we present a new practical addition to the approximation algorithm which improves its performance distinctly. Finally, we prove the performance of the graph bisection algorithm using multicommodity flow approximation by computing formerly unknown bisection widths of some DeBruijn- and Shuffle-Exchange-Graphs.

1 Introduction

We consider the graph bisection problem that consists in partitioning the nodes of an arbitrary undirected graph into two sets of equal size such that the (possibly weighted) sum of edges that cross the cut is minimized. The problem is known to be NP-hard. Based on our previous work presented in [31], we develop an algorithm that, given a graph, determines its exact bisection width which is defined as the value of the smallest balanced cut. Because of its inherent hardness, we cannot hope for an efficient algorithm to tackle the problem (at least in terms of worst case running times and under the common assumption that $\mathrm{NP} \neq \mathrm{P}$), but we aim at increasing the size of instances that can still be solved in a reasonable amount of time.

As it is the case for any combinatorial optimization problem, the task of computing the bisection width of a graph is twofold. First, an optimal solution to the problem must be computed, and second, its optimality must be proved. Regarding the first task, efficient heuristics have been developed in the literature for computing high quality and, for small graphs, often even optimal solutions very quickly [14, 19, 24, 26]. To prove the optimality of a given solution, we use a total enumeration branch&bound-approach. The key issue for the development of a good tree search algorithm is to compute tight lower bounds on the bisection width.

⋆ This work was partly supported by the German Science Foundation (DFG) project SFB-376, by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT), and by the Intelligent Information Systems Institute, Cornell University (AFOSR grant F49620-01-1-0076).

In the last years, a number of exact approaches for solving the graph partitioning problem have been presented. The most recent approach is presented in [18] by S.E. Karisch, F. Rendl, and J. Clausen who use semidefinite programming relaxations for computing bounds on the objective. In [6], Ferreira et al. present a branch-and-cut algorithm for the problem. The same approach is followed by L. Brunetta, M. Conforti, and G. Rinaldi in [2]. In [15], E. Johnson, A. Mehrotra, and G. Nemhauser elaborate on a column generation approach for the graph partitioning problem.

In [31], we presented a new lower bound for the bisection width of a graph. It can be obtained by solving a multicommodity flow problem with variable sender volumes. The problem may be viewed as a generalized maximum multicommodity flow problem: we need to compute a maximum multicommodity flow on an undirected network where the commodities have exactly one source, but may have many sinks. Building up on existing techniques for the solution of multicommodity flow problems, we develop a fully polynomial time approximation scheme (FPTAS). The routine is embedded in a branch&bound-framework for exact graph bisection and experimentally compared with a Lagrangian relaxation based cost-decomposition approach and a barrier LP-solver on various test instances. On top of that, we compare the multicommodity bound and the different algorithms for its computation/approximation with the semidefinite bounding routine developed in [18]. Finally, we prove the performance of our exact graph bisection algorithm by computing the previously unknown bisection width of some DeBruijn- and Shuffle-Exchange-Graphs.

The paper is structured as follows: In Section 2, we review the lower bounds on the bisection width that have been proposed in [31], especially the so called *VarMC*-bound. In Section 3, we develop an algorithm for the approximation of that VarMC-bound. A complete branch&bound-approach is sketched in Section 4. Finally, in Section 5, we present our empirical evaluation.

2 Bounds on Graph Bisection

Our work bases on the lower bounds on the graph partitioning problem introduced in [31]. In the following, we give a small survey on the main ideas of these bounds:

A well known lower bound (see [21]) on the bisection width can be achieved by embedding a clique with the same number of nodes n into the given graph G. When embedding the clique, an edge between two nodes translates into a path in G. The number of paths that use the same edge e is called the *congestion* of e. The edge with the maximum congestion then determines the congestion of the embedding. If the clique can be embedded with congestion C, we know that G has a bisection width of at least $\frac{n^2}{4C}$. Note that the clique embedding can also be viewed as an integer multicommodity flow problem where every vertex sends a commodity of size one to every other vertex.

First, we note that actually we do not need to enforce the integrality constraints on the flows and thereby can strengthen the bound computed [1]. We can further improve on this bound by taking two steps of generalization: First, it can be observed that every single-source multicommodity flow instance (with arbitrary demands and destinations)

[1] For flows, the congestion of an edge is the amount of flow that is routed through it.

can be used to compute a lower bound on the bisection width. The critical point is the $CutFlow$, i.e. the amount of flow which can be ensured to cross every possible bisection of the graph. It is easy to show that $\frac{CutFlow}{C}$ is always a valid lower bound on the bisection problem.

Second, we do not have to select an appropriate multicommodity flow instance by ourselves: Typically, linear programming techniques are used to solve multicommodity flow problems. In [31], we introduced the idea to leave the selection of an appropriate multicommodity flow instance to the linear program by adding some variables and constraints. Two different possibilities with a different degree of freedom for the selection of the multicommodity flow instance have been introduced: the *VarMC* and the *MVarMC* formulations. In the MVarMC formulation, every node has the freedom to send a commodity of arbitrary size to each other node. Whereas in the VarMC formulation, every node has to send a commodity of arbitrary size to every other node, whereby each destination gets the same share. Experiments have shown that, for the graph bisection problem, the VarMC formulation gives equally good bounds when compared with the MVarMC formulation. Therefore, in the paper at hand, we compare different solution techniques for the computation of the VarMC-bound.

3 Approximation of the VarMC-bound

3.1 The LP-formulation

Even though the continuous multicommodity flow problem (MCF) is solvable in polynomial time, for more than a decade now researchers have tried to develop approximation schemes for the problem. The reason for this at first surprising fact is that large multicommodity flow instances have to be solved in many application areas and in combination with a whole variety of discrete optimization problems such as network design or graph bisection. And standard simplex or interior point LP-solvers, even specialized software based on primal basis partitioning [4, 5] or Lagrangian relaxation based resource- or cost-decompositions [3, 9] are simply not fast enough to tackle real-size instances of these problems very efficiently. Therefore, there is a big interest in the development of algorithms that provide near optimal solutions more quickly.

In this section, we develop an ϵ-approximation scheme for the VarMC-bound. The space restrictions prevent us from explaining the VarMC-bound in more detail. We refer the interested reader to [31]. However, in the following we state a general linear program that generalizes the VarMC-bound in the sense that for a given graph the VarMC-bound is a special instance of that LP with specific edge-capacities, cost coefficients, and demand values.

Let $G = (V, E)$ denote an undirected network with $|V| = n$, $|E| = m$, with associated node-arc incidence matrix $N \in \{-1, 0, 1\}^{n \times 2m}$, and capacities $u \in \mathbb{R}^m_{\geq 0}$. Further, let $K \in \mathbb{N}$ denote the number of commodities with source nodes $s^k \in V$ and demands $d^k \in \mathbb{R}^n$, $1 \leq k \leq K$, whereby $d^k_i \geq 0$ for all $i \neq s^k$, and $\sum_i d^k_i = 0$. Finally, denote with p^k the cost coefficient of commodity k. Then, the problem is to

$$
\begin{array}{lrll}
\text{Maximize} & \sum_k p^k \lambda^k & & \\
\text{subject to} & N x^k = \lambda^k d^k & \forall\ 1 \le k \le K & (1) \\
& \sum_k x^k_{ij} + x^k_{ji} \le u_{ij} & \forall\ \{i,j\} \in E & (2) \\
& \lambda, x \ge 0. & &
\end{array}
$$

This way to state the problem allows us directly to compute a lower bound on the bisection width of a given graph (compare with Section 2): the objective maximizes the CutFlow, whereby the variables λ^k determine the sender volume of commodity k, and the x^k give the corresponding flow in the network. The constraints (1) ensure that x^k is a feasible flow of commodity k with volume λ^k, and restrictions (2) enforce that the capacity of the undirected edges $\{i,j\}$ is not violated (i.e., the congestion is at most 1).

Note, that to solve the problem it is sufficient to look at the special case with $p^k = 1$ for all $1 \le k \le K$, because for $p^k \le 0$ we set $\lambda^k = 0$ and $x^k = 0$, and otherwise we can scale the demand d^k by $1/p^k$.

3.2 The FPTAS

The first FPTAS's for maximum multicommodity flow were based on Lagrangian relaxations and linear programming. They have been improved and adapted to different models in a series of papers [12, 13, 17, 20, 22, 25, 27, 32]. While all this research was built on the idea of rerouting existing flows, the idea of augmenting flow has led to a couple of new algorithms with improved running times [7, 8, 10, 16, 34]. Several publications also report about the usability of approximation algorithms for multicommodity flow problems in practice [1, 11, 13, 28, 29].

We try to keep the paper self-contained. Note, however, that the following description is analogue to the "maximum multicommodity flow"-section in [8], which itself builds directly on the analysis given in [10]. Our modest contribution here consists in the generalization of the existing algorithms for the maximum multicommodity flow problem to an FPTAS that can handle commodities with more than one sink. This, of course, is essential for the computation of a lower bound for the bisection width of a graph (see Section 2). At the same time, because we focus on graph bisection here, we enable the FPTAS to handle undirected edges. However, the changes that are necessary for that purpose are not of fundamental nature, so that the theory we present can also be used for the development of an approximation scheme for generalized maximum multicommodity flow problems on directed graphs with multi-sink commodities.

For all $1 \le k \le K$, denote with $\pi^k(i)$ the set of all paths from the source s^k to the sink $i \in V \setminus \{s^k\}$. Further, set $\pi^k = \cup_{i \ne s^k} \pi^k(i)$. Using variables y^k_P representing the flow of commodity k along some path $P \in \pi^k$, we achieve a *path-based* formulation of the problem:

$$
\begin{array}{lrl}
\text{Maximize} & \sum_k \lambda^k & \\
\text{subject to} & \sum_{P \in \pi^k(i)} y^k_P = \lambda^k d^k_i & \forall\ 1 \le k \le K,\ i \in V \setminus \{s^k\} \\
& \sum_k \sum_{P \in \pi^k : \{i,j\} \in P} y^k_P \le u_{ij} & \forall\ \{i,j\} \in E \\
& \lambda, y \ge 0 &
\end{array}
$$

The dual of the above can be written as

$$\begin{array}{lll} \text{Minimize} & \sum\limits_{\{i,j\}\in E} u_{ij} l_{ij} & \\ \text{subject to} & \sum\limits_{\{i,j\}\in P} l_{ij} \geq z_h^k & \forall\, 1 \leq k \leq K,\ h \in V \setminus \{s^k\},\ P \in \pi^k(h) \\ & \sum\limits_{i \neq s^k} d_i^k z_i^k \geq 1 & \forall\, 1 \leq k \leq K \\ & l \geq 0. & \end{array}$$

That is, we have to assign a length $l_{ij} \geq 0$ to each edge $\{i,j\} \in E$, such that $\sum_{i \neq s^k} d_i^k \, dist_i^k(l) \geq 1$ for all $1 \leq k \leq K$, and $\sum_{\{i,j\}\in E} u_{ij} l_{ij}$ is minimized, whereby $dist_i^k(l)$ denotes the shortest path distance from s^k to the sink $i \in V$ in G under length function l.

Consider the approximation scheme sketched in Figure 1. We start with length function $l \equiv \delta$ for an appropriately defined $\delta = \delta(\epsilon, G, d)$, and the primal solution $x \equiv 0$. The length function l is defined on the undirected edge set E, whereas we maintain flow values x_{ij}^k and x_{ji}^k for each edge $\{i,j\} \in E$ and all $1 \leq k \leq K$. While there is still a tree T along which we can route the demand of a commodity k with costs less than 1, the algorithm selects such a tree and augments flow along this tree. More precisely, the algorithm selects a tree with approximately minimal costs up to an approximation factor of $1+\epsilon$. This property is achieved by maintaining a lower bound $\hat{\alpha}$ on the current minimal routing costs of any commodity.

```
 1: x :≡ 0, l :≡ δ
 2: α̂ ← min_k Σ_i d_i^k δ, oldSink ← −1
 3: repeat
 4:    for all 1 ≤ k ≤ K do
 5:       if oldSink ≠ s^k then
 6:          T ← Shortest_Path_Tree(s^k, l)
 7:          oldSink ← s^k
 8:       while Σ_i d_i^k dist_i^k(l) < min{1, (1+ε)α̂} do
 9:          for all (i,j) ∈ T do
10:             c_ij^k ← Σ_{h∈T_j} d_h^k
11:          φ ← min_{(i,j)∈T} (2x_ji^k + u_ij)/c_ij^k, Δ :≡ 0
12:          for all (i,j) ∈ T do
13:             Δ_ji ← − min{x_ji^k, φ c_ij^k}
14:             Δ_ij ← φ c_ij^k + Δ_ji
15:             if Δ_ij + Δ_ji > 0 then
16:                l_ij ← l_ij (1 + ε(Δ_ij + Δ_ji)/u_ij)
17:          x ← x + Δ
18:          T ← Shortest_Path_Tree(s^k, l)
19:    α̂ ← α̂(1+ε)
20: until α̂ ≥ 1
```

Fig. 1. The FPTAS.

The amount of flow sent along tree T is determined in the following way: Denote with T_j all nodes in the subtree routed at node $j \in V$ (including j). For each edge $\{i,j\} \in T$ we compute the congestion c_{ij}^k when routing the demand of commodity k along that tree, i.e., we set $c_{ij}^k = \sum_{h \in T_j} d_h^k$. Basically, we achieve a feasible routing by scaling the flow by $\min_{\{i,j\}\in T} u_{ij}/c_{ij}^k$. However, because we are working on an undirected network here, we would like to consider only flows with $\min\{x_{ij}^k, x_{ji}^k\} = 0$

for all $1 \leq k \leq K$ and $\{i,j\} \in E$. When we also incorporate and change the current flow x^k_{ji} of commodity k in the opposite direction, we achieve an even bigger scaling factor of $\min_{(i,j)\in T}(2x^k_{ji} + u_{ij})/c^k_{ij}$. Formally, we can prove Lemma 1 regarding the change $\Delta_{ij} + \Delta_{ji}$ of the current flow on edge $\{i,j\} \in E$ of commodity k.

In case of a positive flow change on an edge $\{i,j\} \in E$ ($\Delta_{ij} + \Delta_{ji} > 0$), we update the dual variables by setting $l_{ij} \leftarrow (1 + \epsilon(\Delta_{ij} + \Delta_{ji})/u_{ij})l_{ij}$, i.e., we increase the lengths of an edge exponentially with respect to the congestion of that edge. Finally, the primal solution is updated by $x \leftarrow x + \Delta$. This setting may yield an infeasible solution, since it may violate some capacity constraint. However, the mass balance constraints are still valid. This allows us, at the end of the algorithm, to scale the final flows x^k so that they build a feasible solution to the problem.

With the FPTAS in Figure 1, we are able to prove

Theorem 1. *Let $T_{SP} = m + n \log n$. An ϵ-approximate VarMC-bound on the bisection width of a graph can be computed in time $O^*(mT_{SP}/\epsilon^2)$*[2].

Following the analysis in [8], we prove the previous theorem with the help of a sequence of Lemmas[3]. We set $\rho = \min_{k,i}\{d^k_i \mid d^k_i > 0\}$, and $\sigma = \log_{1+\epsilon}((1+\epsilon)/(\delta\rho))$. Then,

Lemma 1. *In every iteration, in which the current flow is changed, it holds:*

a) $\Delta_{ij} + \Delta_{ji} \leq u_{ij}$ *for all* $\{i,j\} \in E$, *and*
b) there exists an edge $\{i,j\} \in E$ *such that* $\Delta_{ij} + \Delta_{ji} = u_{ij}$.

Lemma 2. *The flow obtained by scaling the final flow by $1/\sigma$ is primal feasible.*

Lemma 3. *Let $\tau = (1+\epsilon)/\rho$, and denote with L the maximum number of edges in a simple path from a source s^k to one of its sinks $i \in V$. When setting $\delta = \tau(\tau L \max_k \sum_{i \neq s^k} d^k_i)^{-1/\epsilon}$, the final flow scaled by $1/\sigma$ is optimal with a relative error of at most 3ϵ.*

3.3 Implementation Details

Three main practical improvements have been suggested for this kind of approximation algorithms for multicommodity flow problems (see e.g. [11, 28, 8]): First, it is suggested to choose ϵ more aggressively and to restart the algorithm with a smaller guaranteed error only if the final performance achieved is not good enough. Of course, this procedure makes sense when we want to compute a solution with a given maximal error as fast as possible. But in our context, we have to select an ϵ that works best for the branch& bound algorithm. We come back to this point in the experiments described in Section 5.

Second, in [8, 28] it is suggested to use another length function: Instead of setting $l_{ij} \leftarrow l_{ij}(1 + \epsilon\frac{\Delta}{u_{ij}})$ one can also modify l by $l_{ij} \leftarrow l_{ij}e^{\epsilon\frac{\Delta}{u_{ij}}}$. This modified length function does not change the analysis but it is reported that the algorithm behaves better

[2] We write O^* to denote the "smooth" O-calculus hiding logarithmic factors.
[3] A full version of this paper including all proofs can be found in [30].

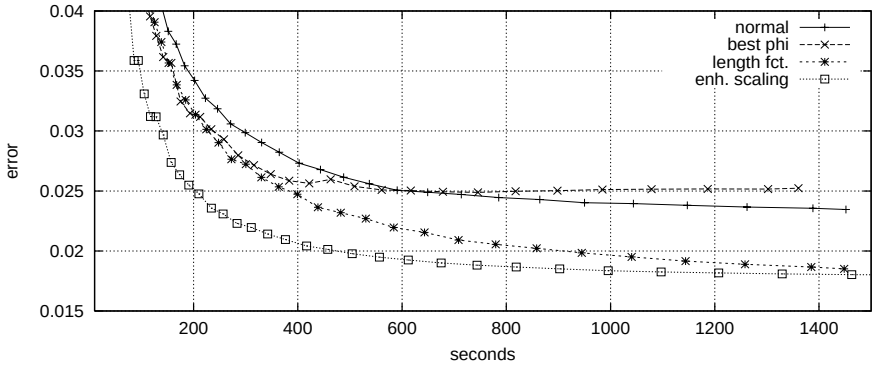

Fig. 2. Comparison of different practical improvements of the FPTAS.

in practice. Third, 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 (in Line 11). This suggestion originates from approximation algorithms prior to [10] that work slightly differently. The idea is to choose ϕ such that a specific potential function is minimized that corresponds to the dual value of the problem. In our algorithm, though, it is not possible to compute ϕ efficiently so that the dual value is minimized. However, it is possible to compute ϕ so that the primal value is maximized, so we experimented with this variant.

Besides these suggestions we present another variation of the approximation algorithm that we call *enhanced scaling*: During the algorithm, for each commodity k, we obtain a flow x^k and possibly also a scalar $\lambda^k = -|x^k|/d^k_{s^k} \geq 0$. In order to construct a feasible flow, instead of scaling all x^k equally, we could also set up another optimization problem to find scalars ξ^k that solve the following LP:

$$\begin{array}{lll} \text{Maximize} & \sum_k \lambda^k \xi^k & \\ \text{subject to} & \sum_k \xi^k (x^k_{ij} + x^k_{ji}) \leq u_{ij} & \forall\, \{i,j\} \in E \\ & \xi \geq 0 & \end{array}$$

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 each 100th iteration is a good choice for the instance sizes that we tackle here. Notice that we use this scaling only in order to get a feasible solution value; the primal solution which is used while the algorithm proceeds is not changed! Indeed, we have also tried to continue with the scaled solution in the algorithm, but this variant performs quite badly.

Figure 2 shows the effect of the three different improvements. This example was computed with a DeBruijn graph of dimension 8, and $\epsilon = 0.1$. We have performed many tests with a couple of different settings, so we can say that Figure 2 shows a typical case. It shows the error that results from the difference of the current primal and dual solution depending on 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.

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 also carried out some experiments with all different combinations of the three improvements, but no combination performs as well as the variant with enhanced scaling alone. So in all following experiments we use this variant.

4 A Branch&Bound Algorithm

Our main goal is the computation of exact solutions for graph bisection problems. Thus, we construct a branch&bound algorithm using the described VarMC-bound as lower bound for the problems. A detailed description of the implementation can be found in [31]. In the following, we give a brief survey on the main ideas:

First, we heuristically compute a graph bisection using *PARTY* [26]. Since this start-solution is optimal in most cases, we only have to prove optimality. We use a pure depth first search tree traversal for this purpose. The branching is done on the decision whether two specific vertices $\{v, w\}$ stay in the same partition (join) or if they are separated (split). A join is performed by merging these two vertices into one vertex. A split is performed by introducing an additional commodity from vertex v to vertex w whose entire amount is known to cross the cut. Thus, it can be added to the $CutFlow$ completely. The selection of the pair $\{v, w\}$ for the next branching is done with the help of an upper bound on the lower bound. Additionally to this idea described in [31], the selection is restricted to pairs $\{v, w\}$ (if any) where one node (say, v) has been split with some other node $u \neq w$ before. Then, split of $\{v, w\}$ implies a join of $\{u, w\}$, of course.

The "Forcing Moves" strategy described in [31], Lemma 2, is strengthened. Forcing moves are joins of vertices that can be deduced in an inference process. In the original version, only adjacent vertices $\{v, w\}$ were considered. Now, we look at a residual graph with edge-capacities corresponding to the amount of capacity which is not used by a given VarMC solution. Two vertices v and w can be joined if the maximal flow from v to w in the residual graph exceeds a specific value.

5 Numerical Results

We now present the results of our computational experiments. All of them were executed on systems with Intel Pentium-III, 850 MHz, and 512 MByte memory. To show the behavior on different kinds of graphs, we use four different sets of 20 randomly generated graphs: The set *RandPlan* contains random maximal planar graphs with 100 vertices; the graphs are generated using the same principle as it is used in LEDA [23]. Benchmark set *RandReg* consists of random regular graphs with 100 vertices and degree four; for its generation, the algorithm from Steger and Wormald [33] is used. The set *Random* contains graphs with 44 vertices where every pair $\{v, w\}$ is adjacent with probability 0.2. The set *RandW* consists of complete graphs with 24 vertices where every edge has a random weight in the interval $\{0, \ldots, 99\}$.

In most works on exact graph partitioning (see e.g. [2, 18]), sets like *Random* and *RandW* are used for the experiments. We added the sets of random regular and random

	graph	$\epsilon = 0.025$ time	subp.	$\epsilon = 0.05$ time	subp.	$\epsilon = 0.1$ time	subp.	$\epsilon = 0.25$ time	subp.	$\epsilon = 0.5$ time	subp.	$\epsilon = 0.75$ 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

Fig. 3. Times and sizes of the search-trees of the branch&bound algorithm using the approximation algorithm with different ϵ's. (1): without forcing moves, (2): with forcing moves.

planar graphs here, because we believe that more structured graph classes should also be considered with respect to their bigger relevance for practical applications.

5.1 Lower Bounds using the FPTAS

The best choice of ϵ for the use in a branch&bound environment is a trade-off. If ϵ is too big, the bound approximation computed is too bad, and the number of subproblems in the search-tree explodes. On the other hand, if ϵ is chosen too small, the bound approximation for each subproblem is too time consuming.

Figure 3 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 forcing moves 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 forcing moves: looking at the random planar graphs, we see that less good solutions of the VarMC bound can result in stronger forcing moves so that the number of subproblems may even decrease. The figure also shows that the effects of forcing moves 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, which is a surprisingly large value.

5.2 Comparison of Lower Bound Algorithms

graph	CPLEX time	subp.	Approx. time	subp.	Cost-Dec. 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

Fig. 4. Average running times (seconds) and sizes of the search-trees using the different methods for computing the VarMC-bound.

We give a comparison of the results of the branch&bound algorithm with forcing moves using the following three different methods for computing the VarMC-bound: 1. standard barrier LP-solver (CPLEX, Version 7.0, primal and dual simplex algorithms cannot compete with the three approaches presented here), 2. the approximation algorithm with $\epsilon = 0.5$ and enhanced scaling, and 3. a Lagrangian relaxation based cost-decomposition routine using a max-cutflow formulation

Graph	$\|V\|$	$\|E\|$	bw	CPLEX		Decomp.		Approx.		CUTSDP	
				Bound	Time	Bound	Time	Bound	Time	Bound	Time
Grid 10x11	110	199	11	11.00	16	11.00	54	10.59	2	8.48	30
Torus 10x11	110	220	22	20.17	15	20.17	48	19.66	2	17.63	30
SE 8	256	381	28	26.15	484	25.69	244	24.94	16	18.14	453
DB 8	256	509	54	49.54	652	49.05	322	46.95	21	35.08	443
BCR m8	148	265	7	7.00	22	7.00	64	6.96	5	5.98	80
ex36a	36	297	117	104.17	9	104.16	26	97.57	<1	116.41	1

Fig. 5. Comparison of the different VarMC bounds with the semi-definite bound. The bound quality at the root node as well as the time for its computation are given.

and the Crowder rule in the subgradient update (see [30] for details). Figure 4 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.

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 or 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 widths, whereas both cost-decomposition and approximation allowed us to prove a bisection width of 92 and 48, respectively. Apart from being more memory efficient, we observe that cost-decomposition does not allow us to solve our generalized maximum multicommodity flow problem faster than standard LP methods.

5.3 Comparison with Semi-Definite Lower Bounds

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 presented in [18]. The work in [18] is the most recent and successful on the graph bisection problem apart from the VarMC-bound. So we compare our lower bounds with these ones. In order to compute the semi-definite bounds we have used the CUTSDP-package, which is publicly available. The program uses parameters *maxlarge* and *maxsmall*. According to the setting in [18], we set $\mathit{maxlarge} := 1$, and $\mathit{maxsmall} := 10$.

The results are presented in Figure 5. Here we have not used randomly generated graphs but more structured graphs: a grid, a torus, the shuffle-exchange graph of dimension 8, and the DeBruijn graph of dimension 8. Additionally we report the results on the "BCR m8" graph which stems from a finite elements application, and the "ex36a" graph which is a small random dense graph introduced in [18].

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 than the semi-definite bounds. And the running times are clearly the smallest ones out of the four compared methods. The results also show that the semi-definite bound is better for the very dense random graph while the VarMC bound is better for more structured or sparse graphs.

6 Conclusions

We developed an algorithm for the approximation of the VarMC-bound on the bisection width of a graph. It is based on an approximation scheme for maximum multicommodity flows and yields an ϵ-approximation in time $O^*(m^2/\epsilon^2)$. We experimented with different practical improvements that have been suggested for this kind of multicommodity flow approximation scheme and were able to improve the practical behavior by using the new enhanced scaling technique.

When comparing the approximation algorithms with a barrier LP-solver and a Lagrangian relaxation based cost-decomposition algorithm, we found that it is favorable to use the approximation scheme, both with respect to the memory consumption and running time. The VarMC-approximation scheme allowed us to compute the bisection width of large graphs, such as DeBruijn 9 (the bisection width is 92), Shuffle-Exchange 9 (48), and Shuffle-Exchange 10 (82), which were unknown and out of the reach of exact graph bisection algorithms before.

References

1. C. Albrecht. Provably good global routing by a new approximation algorithm for multicommodity flow. In *Proc. of the Int. Conference on Physical Design*, pages 19–25, 2000.
2. L. Brunetta, M. Conforti, and G. Rinaldi. A branch-and-cut algorithm for the equicut problem. *Mathematical Programming*, 78:243–263, 1997.
3. T.G. Crainic, A. Frangioni, and B. Gendron. Bundle-based relaxation methods for multicommodity capacitated fixed charge network design. *Discrete Applied Mathematics*, 112:73–99, 2001.
4. J. Farvolden, K. Jones, I. Lustig, and W. Powell. Multicommodity Network Flows — The Impact Of Formulation On Decomposition. *Mathematical Programming*, 62:95–117, 1993.
5. J. Farvolden, W. Powell, and I. Lustig. A primal partitioning solution for the arc-chain formulation of a multicommodity network flow problem. *Operations Research*, 41(4):669–693, 1993.
6. 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.
7. L. Fleischer and K.D. Wayne. Fast and simple approximation schemes for generalized flow. In *Proceedings 10th Symposium on Discrete Algorithms*, 1999.
8. L. K. Fleischer. Approximating Fractional Multicommodity Flow Independent of the Number of Commodities. *SIAM Journal on Discrete Mathematics*, 13(4):505–520, 2000.
9. A Frangioni. A Bundle type Dual-ascent Aproach to Linear Multi-Commodity Min Cost Flow Problems. Technical Report TR-96-01, Dipartimento di Informatica, Universita di Pisa, 1996.
10. N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *39th Annual IEEE Symposium on Foundations of Computer Science*, pages 300–309, 1998.
11. A.V. Goldberg, J.D. Oldham, S.A. Plotkin, and C. Stein. An Implementation of a Combinatorial Approximation Algorithm for Minimum-Cost Multicommodity Flow. In *Integer Programming and Combinatorial Optimization*, volume 1412, pages 338–352, 1998.
12. M.D. Grigoriadis and L.G. Khachiyan. Fast approximation schemes for convex programs with many blocks and coupling constraints. *SIAM Journal on Optimization*, 4:86–107, 1994.

13. M.D. Grigoriadis and L.G. Khachiyan. Approximate minimum-cost multicommodity flows. *Math. Programming*, 75:477–482, 1996.
14. B. Hendrickson and B. Leland. The chaco user's guide: Version 2.0. Technical Report SAND94-2692, Sandia National Laboratories, Albuquerque, 1994.
15. E. Johnson, A. Mehrotra, and G. Nemhauser. Min-cut clustering. *Mathematical Programming*, 62:133–151, 1993.
16. G. Karakostas. Fast Approximation Schemes for Fractional Multicommodity Flow Problems. In *SIAM Symposium on Discrete Algorithms*, 2002.
17. D. Karger and S. Plotkin. Adding multiple cost constraints to combinatorial optimization problems, with applications to multicommodity flows. In *Proc. of the 27th Symp. on Theory of Computing*, pages 18–25, 1995.
18. S. E. Karisch, F. Rendl, and J. Clausen. Solving graph bisection problems with semidefinite programming. *INFORMS Journal on Computing*, 12(3):177–191, 2000.
19. 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.
20. P. Klein, S. Plotkin, C. Stein, and E. Tardos. Faster approximation algorithms for the unit capacity concurrent flow problem with applications to routing and finding sparse cuts. *SIAM Journal on Computing*, 23:466–487, 1994.
21. F. T. Leighton. *Introduction to Parallel Algorithms and Architectures*. Morgan Kaufman, 1992.
22. T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos, and S. Tragoudas. Fast Approximation Algorithms for Multicommodity Flow Problems. *Journal of Computer and System Sciences*, 50(2):228–243, 1995.
23. K Mehlhorn and S. Nähler. LEDA: A Platform for Combinatorial and Geometric Computing. *Communications of the ACM*, 38(1):96–102, 1995.
24. 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.
25. S.A. Plotkin, D. Shmoys, and E. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Math. of Operations Research*, 20:257–301, 1995.
26. 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.
27. T. Radzik. Fast deterministic approximation for the multicommodity flow problem. In *Proc. of the 6th Symp. on Discrete Algorithms*, pages 486–492, 1995.
28. T. Radzik. Experimental study of a solution method for multicommodity flow problems. In *Proc. of the 2. Workshop on Algorithm Engineering and Experiments*, pages 79–102, 2000.
29. M. Sato. Efficient implementation of an approximation algorithm for multicommodity flows. Master's thesis, Graduate School of Engineering Science, Osaka University, 2000.
30. M. Sellmann. *Reduction Techniques in Constraint Programming and Combinatorial Optimization*. PhD thesis, University of Paderborn, Germany, http://www.upb.de/cs/sello/diss.ps, 2002.
31. N. Sensen. Lower Bounds and Exact Algorithms for the Graph Partitioning Problem. In F. Meyer auf der Heide, editor, *Algorithms - ESA 2001*, volume LNCS 2161, pages 391–403, 2001.
32. F. Shahrokhi and D.W. Matula. The maximum concurrent flow problem. *Journal of the ACM*, 37:318–334, 1990.
33. A. Steger and N.C. Wormald. Generating random regular graphs quickly. *Combinatorics, Probab. and Comput.*, 8:377–396, 1999.
34. N. Young. Randomized rounding without solving the linear program. In *Proc. of the 6th Symp. on Discrete Algorithms*, pages 170–178, 1995.