# A Heuristic for Fair Correlation-Aware Resource Placement*

**Technical Report CS-2009-04**

**January 20, 2009**

Raouf Boutaba, Martin Karsten, and Maxwell Young

David R. Cheriton School of Computer Science, University of Waterloo, ON, Canada; email: rboutaba@uwaterloo.ca, mkarsten@uwaterloo.ca, m22young@uwaterloo.ca

**Abstract.** The configuration of network resources greatly impacts the communication overhead for data intensive tasks and constitutes a critical problem in the design and maintenance of networks. To address the issue of resource placement, we analyze and implement a heuristic for solving a known NP-complete graph optimization problem called MAXIMUM SIZE BOUNDED CAPACITY CUT. Experimental results for our heuristic demonstrate promising performance on both synthetic and real world data. Next our heuristic is used as a sub-routine to solve another known NP-complete problem called MIN-MAX MULTIWAY CUT whose traits we adapt to yield a resource placement scheme that exploits correlations between network resources. Our experimental results show that the resulting placement scheme achieves a significant savings in communication overhead.

## 1 Introduction

While a measure of cost in a network is often dependent upon the given scenario, cost is generally coupled with the volume of communication between network entities. In turn, the communication between network entities is dependent upon the allocation of resources in the network. There are several scenarios characterized by frequent access to limited and multiple resources such as complex queries in distributed databases, which often require the aggregation of several objects, or the use of keyword indices by search engines in order to efficiently resolve user queries, or peer-to-peer (P2P) file sharing, where latency and bandwidth usage are highly dependent upon where files are stored.

In such environments, the location of resources has an impact on the efficiency of the system. For example, if resources $A$ and $B$ are often requested together, but stored at different locations, the communication overhead incurred by queries for these resources can be significantly greater than if $A$ and $B$ are colocated. In this work, we consider the communication costs between network entities and model the task of resource placement using a known graph optimization problem. Under this model, the goal is to distribute resources to locations in a network such that the maximum cost between any given pair of partitions is minimized; this aspect lends an important notion of *"fairness"* to our resource placement scheme.

We also demonstrate that correlation is crucial for motivating a notion of cost in our placement scheme. Correlation values are calculated using data from prior network

* Research partially supported by NSERC.

transactions. Therefore, it is assumed that knowledge of communication trends is available either as traffic engineering matrices, snapshots of past communications, or design considerations. For instance, Internet Service Providers routinely trace traffic in P2P networks and this information can be used to improve network performance [12]. Naturally, this approach raises the question of whether such correlations are stable enough over a sufficient period of time to warrant the additional computational costs of a correlation-aware placement scheme. Indeed, it has been demonstrated that resource correlations *do* remain stable over at least *month-long periods* [23].

From a practical perspective, the resource placement scheme we propose can be managed by an authority in a distributed setting, such as an ISP; there exist proposals for cooperation between ISPs and P2P networks [5, 20]. Alternatively, in a centralized scenario, such a scheme would be useful within single-administrative domains, such as allocation in data-centers. We treat the details of such a setup as outside the scope of this paper, and instead focus on the analysis of our scheme.

### 1.1 Our Contributions

We propose a resource placement scheme that exploits correlation information between network resources; we call this a *correlation-aware resource placement scheme*. In formulating our scheme, two known NP-complete problems are dealt with: MAXIMUM SIZE BOUNDED CAPACITY CUT (MAXSBCC) and MIN-MAX MULTIWAY CUT (MMMC). We mathematically analyze and implement a heuristic for MAXSBCC based on the technique of semidefinite programming (SDP). Our heuristic is then employed as a subroutine for solving MMMC. To the best of our knowledge, our work provides the first empirical evaluation of these two problems.

We then identify two challenges to employing MMMC as a model of resource placement. First, naively employing a cost metric can lead to insensible solutions. Second, privacy issues are absent from the model. We address both challenges by showing how correlation adequately motivates important cost metrics, and we mathematically extend the model to account for privacy constraints. Finally, an experimental evaluation of our correlation-aware resource placement scheme is conducted with real-world data.

### 1.2 Related Work

Our work differs from a number of previous treatments on data placement where cost metrics are not motivated by correlation and, moreover, cost is measured as an aggregate [14] or average [6] across the entire network. The most relevant related work is [23] where the authors address the use of correlation in placing data items in a network. However, again, their work aims to minimize an *aggregate* notion of cost in contrast to the substantially different min-max approach used here.

Our correlation-aware resource placement scheme is based on the MIN-MAX MULTIWAY CUT problem which was introduced by Tardos and Svitkina [18]. The MIN-MAX MULTIWAY CUT problem is NP-complete and the best known approximation algorithm relies on obtaining an efficient solution to a sub-problem called the MAXIMUM SIZE BOUNDED CAPACITY CUT problem. In [18], the authors cite an algorithm developed in [8] to achieve a polylogarithmic approximation. However, the algorithm

of [8] is extremely intricate and, several years after the theoretical result, no implementation exists.[1] Moreover, due to the large number of constraints, this algorithm is likely to be extremely computationally intensive, even for very small problem instances.

In contrast, our work incorporates an efficient heuristic based on the technique of semidefinite programming (SDP). SDP has figured prominently in the development of heuristics for problems in the areas of phylogenic reconstruction [16], machine learning [22], sensor network layout [7], bioinformatics [13] and graph partitioning [9]; these results demonstrate that heuristics can benefit greatly from this optimization technique.

## 2 Our Heuristic and Analysis

To solve the MIN-MAX MULTIWAY CUT problem, we solve a subproblem known as the MAXIMUM-SIZE BOUNDED-CAPACITY CUT (MAXSBCC) problem introduced in [18]. The input to MAXSBCC is an undirected graph $G = (V, E)$ with weights on the vertices $w(v)$, capacities on the edges $c(e)$, source and sink vertices $v_s, v_t \in V$, and an integer $B$. Given a partition of $V$ into $S$ and $T$, denote by $\delta(S)$ the total weight of the cut edges and denote by $w(S)$ the total weight of the vertices in $S$. The MAXSBCC problem is to find an $s$-$t$ cut $(S, T)$ such that $\delta(S) \leq B$, and $w(S) = \sum_{v \in S} w(v)$ is maximized. In [18], the authors are concerned with a $(\alpha, \beta)$-bicriteria approximation algorithm for MAXSBCC. That is, given an instance of MAXSBCC with an optimal solution $(S^*, T^*)$, returns in polynomial time a solution $(S', T')$ such that $\delta(S') \leq \alpha B$ and $w(S') \geq \beta w(S^*)$ where $\alpha \geq 1$ and $0 < \beta \leq 1$. Therefore, solutions *may exceed the budget* and this also turns out to be true for our heuristic.

Consider the quadratic program specified by Equations (1)-(4). Variable $x_i$ corresponds to $v_i \in V$, $w_i$ is the weight of vertex $v_i$, and $w_{ij}$ is the edge weight of $(v_i, v_j)$ which is zero if no such edge exists.

$$\max \quad \sum_{i=1}^{n} \frac{1 + x_s x_i}{2} w_i \qquad (1)$$

$$s.t. \quad \sum_{i<j} x_i x_j \, w_{ij} \geq M \qquad (2)$$

$$x_s x_t = -1 \qquad (3)$$

$$x_i \in \{-1, 1\} \qquad (4)$$

$$\max \quad \sum_{i=1}^{n} \frac{1 + y_{si}}{2} w_i \qquad (5)$$

$$s.t. \quad \mathbf{Y} = (y_{ij}) \succeq 0 \qquad (6)$$

$$\sum_{i<j} y_{ij} \, w_{ij} \geq M \qquad (7)$$

$$y_{ii} = 1 \qquad (8)$$

$$y_{st} = -1 \qquad (9)$$

Equation (1) counts the cumulative weight of the vertices in $S$. Let $\tau$ denote the cumulative weight of the edges internal to the set $T$ and let $\sigma$ denote the cumulative weight of the edges internal to the set $S$. Equation (2) counts $\sigma + \tau - \delta(S)$. Equation (3) states that the source and sink nodes must be in separate partitions. Equation (4) guarantees that each vertex belongs to one and only one partition. Treating each vertex variable $x_i$ as a vector $\mathbf{v_i}$, and letting $y_{ij} = \mathbf{v_i} \cdot \mathbf{v_j}$, the SDP specified by Equations (5)-(9) is obtained where '$\succeq$' denotes that $Y$ is positive semidefinite. We now analyze the semidefinite program to motivate its suitability for MAXSBCC. Due to space constraints, proofs of the following results are given in the appendix. Let $W^* \geq M$ denote

[1] To the best of our knowledge, no such implementation exists. Furthermore, via email correspondence, Professor R. Krauthgamer (one of the authors of [8]) stated he was unaware of any such implementation.

the value of Equation (7) given by solving the semidefinite program and let $W \leq W^*$ denote its value after applying the standard technique of hyperplane rounding.

**Lemma 1.** *Hyperplane rounding of the SDP provides a* $W$ *such that* $E[W] \geq 0.945W^*$.

Let $S^*$ denote the value of Equation (5) given by the solution to the semidefinite program and let $S$ denote the value of this quantity after hyperplane rounding. We establish the following critical result:

**Lemma 2.** *Let* $\epsilon$ *be a small positive constant. With probability at least* $1 - \frac{1}{n}$*, hyperplane rounding need only be applied* $\lceil 2 \ln n/\epsilon \rceil$ *times before a rounded solution to the SDP is obtained such that* $W \geq (1.823 - \lambda)W^*$ *and* $w(S) \geq \beta' w(S^*)$ *where* $1 \geq \lambda \geq 0.823$.

Finally, we address the quality of our solution:

**Theorem 1.** *With probability at least* $1 - \frac{1}{n}$*, for* $\lambda \in [0.823, 1]$*, the above algorithm achieves a cut* $(S, T)$ *such that:*

$$\delta(S) \leq \lambda + \left(\frac{1-\lambda}{2}\right)\frac{w(E)}{B}$$
$$w(S) \geq (1.823 - \lambda) \cdot w(S^*)$$

As $\lambda$ approaches 1, the quality of our approximation increases correspondingly. Therefore, the quality of our solution depends heavily on $\lambda$ which, in turn, depends on the value $M$ that we set in our program. This relationship suggests a heuristic approach whereby we attempt to improve our solution quality by modifying $M$. Throughout, we refer to the above algorithm by ALG($M$) to reflect that the performance depends on $M$.

### 2.1 Our Full Heuristic: MaxSBCC Solver

We seek a solution to MAXSBCC by performing multiple iterations of ALG(M) and modifying the value of $M$ in our semidefinite program formulation at each iteration. Denote the output of ALG by the tuple $(v, S, T, B_{actual})$ where:

- v: is a boolean variable with value `true` if the solution returned is an $s$-$t$ cut; `false` otherwise.
- $S$: is the set of nodes on the source side of the cut.
- $T$: is the set of nodes on the sink side of the cut.
- $B_{actual}$: is the bound resulting from our setting of $M$.

Due to space constraints, we outline our heuristic which we call MAXSBCC SOLVER; the full pseudocode is given in the appendix. The input into MAXSBCC SOLVER is the graph $G = (V, E)$, the desired input bound $B_0$ and the number of iterations $r$ of ALG(M). The core idea is to modify $M$ at each iteration of ALG until we achieve a $B$ value close, or equal, to our original desired bound $B_0$. $M$ is modified by essentially performing a binary search through the possible values of the input bound $B$.

MAXSBCC SOLVER begins by storing our original input bound $B = B_0$ and $M = W(E) - 2B$. Once executed, the solution is checked for validity by inspecting the boolean $v$ variable. If MAXSBCC SOLVER failed to find an $s$-$t$ cut, the next iteration proceeds with the input bound $B$ doubled. Once a valid solution is returned, $B_{actual}$ is examined; if it is larger than our desired $B_0$, $B$ decreases by $\lceil B/2^j \rceil$ where $j$ denotes the

number of iterations where a valid solution has been achieved. Conversely, if $B_{actual}$ is smaller or equal to $B_0$, $B$ increases by $\lceil B/2^j \rceil$. Note that such changes in the value of $B$ are equivalent to modifying $M$, since $M = W(E) - 2B$ in our SDP.

At the end of $r$ iterations of this process, $S$ is returned. If a valid solution was found, the solution that gave a bound closest to $B_0$ (ie. the iteration where $|B_0 - B_{actual}|$ was smallest) is returned. Otherwise, if no valid solution was found, $S$ will be empty; however, if there exists a cut of size $B_0$ or less, then our algorithm will return a valid cut with high probability and with the attributes demonstrated by our analysis above.

### 2.2 MaxSBCC Solver: Experimental Results

Our experiments are performed on systems with up to $1300$ nodes; there are two reasons for this system size. First, although this situation is improving, solving semidefinite programs is still computationally expensive. The area of semidefinite programming is relatively new and available software for solving such programs is consequently limited. Many solvers scale as $O(n^3)$ where $n$ is the dimension of the semidefinite matrix. However, despite its current computational costs, semidefinite programming is a standard technique for solving many challenging problems. Moreover, a number of recent results address the issue of scalability such as parallelized implementations [17] as well as devising problem formulations that can be computed more efficiently [15]. We believe such techniques can allow our algorithm to scale to much larger system sizes but are outside the scope of this paper.[2] The second reason is that, for our experiments with MAXSBCC, we wish to compare against the optimum solution. This allows a particularly unforgiving comparison in judging the performance of our algorithm. We formulate an integer linear program (ILP) for each of the experimental problem instances. Using the ILP solver CPLEX [1], an optimal solution can be achieved for the purposes of comparison. For our experiments we use the SDP solver 'SemiDefinite Programming Algorithm in Matlab [2].

Our first data set consists of three unweighted (Table 1) and three weighted (Table 2) Barabási-Albert scale free graphs created using the BRITE topology generator [3]. Each graph is connected[3], consists of $300$ vertices and, for the weighted case, edge capacities are exponentially distributed in the range $\{1, ..., 1024\}$ while node weights are chosen uniformly at random in the range $\{1, 1024\}$. Our second data set consists of three unweighted (Table 3) and three weighted (Table 4) connected Waxman graphs [19] of $300$ nodes, with the same capacity and node weight distributions. Finally, we use a real-world data set collected in [11] consisting of a trace of peer-to-peer (P2P) traffic and containing information on data objects both advertised and queried over the course of two months; we restrict our use of this data to mp3 files. Correlation values are derived for each object by examining how often file $x$ and file $y$ were colocated at a peer. Edges with correlation values less than $0.25$, using a $p$-value of $0.1$, are discarded yielding a connected component of $358$ nodes which we used. Each correlation value is multiplied by a factor of $100$ and rounded to the nearest integer in order to provide integer input values for the SDP program. We then select a node of minimum degree, $d_{\min}$, to be the source and a node of maximum degree, $d_{\max}$, to be the sink. Budget values are selected

[2] In terms of our later application to resource placement, previous results of [23] have shown that only a small fraction of the total system need be optimized to achieve substantial savings.

[3] If the graph is disconnected, our algorithm can be used on each component individually. For simplicity, we used only connected graphs in our experiments.

| Trial | B | $S$ | $\delta$ | $S^*$ | $\delta^*$ | $S/S^*$, $\delta/\delta^*$ |
|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 2 | 1 | 2 | 1.000, 1.000 |
| | 10 | 5 | 10 | 9 | 10 | 0.556, 1.000 |
| | 18 | 9 | 18 | 20 | 18 | 0.450, 1.000 |
| | 26 | 15 | 27 | 29 | 24 | 0.517, 1.125 |
| | 34 | 16 | 30 | 38 | 34 | 0.421, 0.882 |
| | 42 | 25 | 43 | 49 | 42 | 0.510, 1.024 |
| | 46 | 299 | 46 | 299 | 46 | 1.000, 1.000 |
| 2 | 2 | 1 | 2 | 1 | 2 | 1.000, 1.000 |
| | 7 | 4 | 8 | 5 | 7 | 0.800, 1.143 |
| | 12 | 6 | 12 | 10 | 12 | 0.600, 1.000 |
| | 17 | 10 | 22 | 15 | 17 | 0.667, 1.294 |
| | 22 | 12 | 23 | 21 | 22 | 0.571, 1.045 |
| | 27 | 299 | 31 | 27 | 27 | 11.074, 1.148 |
| | 31 | 299 | 31 | 299 | 31 | 1.000, 1.000 |
| 3 | 2 | 1 | 2 | 1 | 2 | 1.000, 1.000 |
| | 9 | 4 | 8 | 7 | 9 | 0.571, 0.889 |
| | 16 | 9 | 19 | 15 | 16 | 0.600, 1.188 |
| | 23 | 12 | 23 | 23 | 23 | 0.522, 1.000 |
| | 30 | 15 | 31 | 30 | 27 | 0.500, 1.148 |
| | 37 | 17 | 38 | 38 | 37 | 0.447, 1.027 |
| | 41 | 299 | 41 | 299 | 41 | 1.000, 1.000 |

| Trial | B | $S$ | $\delta$ | $S^*$ | $\delta^*$ | $S/S^*$, $\delta/\delta^*$ |
|---|---|---|---|---|---|---|
| 1 | 6 | 973 | 6 | 973 | 6 | 1.000, 1.000 |
| | 1096 | 13409 | 1360 | 20362 | 1096 | 0.659, 1.241 |
| | 2186 | 20951 | 2590 | 31666 | 2186 | 0.662, 1.185 |
| | 3276 | 34038 | 3943 | 41133 | 3271 | 0.828, 1.205 |
| | 4366 | 151828 | 4367 | 152244 | 4362 | 0.997, 1.001 |
| | 5456 | 156249 | 5460 | 156243 | 5423 | 1.000, 1.007 |
| | 5460 | 156249 | 5460 | 156249 | 5460 | 1.000, 1.000 |
| 2 | 14 | 369 | 14 | 369 | 14 | 1.000, 1.000 |
| | 751 | 10592 | 959 | 13222 | 743 | 0.801, 1.291 |
| | 1488 | 13000 | 1496 | 23478 | 1487 | 0.554, 1.006 |
| | 2225 | 22865 | 2332 | 31874 | 2223 | 0.717, 1.049 |
| | 2962 | 151079 | 2976 | 151003 | 2940 | 1.001, 1.012 |
| | 3699 | 155300 | 3702 | 155284 | 3656 | 1.000, 1.013 |
| | 3702 | 155300 | 3702 | 155300 | 3702 | 1.000, 1.000 |
| 3 | 3 | 917 | 3 | 917 | 3 | 1.000, 1.000 |
| | 770 | 14476 | 1073 | 14958 | 770 | 0.968, 1.393 |
| | 1537 | 17647 | 1572 | 24342 | 1532 | 0.725, 1.026 |
| | 2304 | 22645 | 2317 | 33561 | 2304 | 0.675, 1.006 |
| | 3071 | 147415 | 3065 | 147992 | 3064 | 0.996, 1.000 |
| | 3838 | 151589 | 3838 | 151589 | 3838 | 1.000, 1.000 |

**Tables 1 & 2:** Results of the MAXSBCC heuristic on unweighted and weighted Barabási-Albert graphs, respectively.

as even increments in the range $[d_{\min}, d_{\max}]$. We set $r = 5$ in MAXSBCC SOLVER which we found yielded good solutions.

Tables 1-5 provide the results of MAXSBCC SOLVER in the $S$ and $\delta$ columns, while the optimum solution is given in the $S^*$ and $\delta^*$ columns. For both the weighted and unweighted synthetic data sets, the worst source side approximation is $0.421$ and the worst cut approximation is $1.393$ and, generally, the approximations are even significantly better than these worst cases. Moreover, for our experiment using the real-world data, we observe in Table V that our heuristic yields *very high quality solutions*. We also note that MAXSBCC SOLVER performs relatively quickly, completing within no more than $4$ hours on a machine utilizing a single $1.3$ GHz Intel Itanium2 CPU running SuSE Linux. In comparison, the trials with CPLEX frequently required up to $48$ hours on the same system. Overall, the performance of MAXSBCC SOLVER is promising and, with our heuristic in hand, we now move onto MMMC and resource placement.

## 3 Towards Resource Placement: Min-Max Multiway Cut

As a basic abstract model of resource placement, we use the MIN-MAX MULTIWAY CUT (MMMC) problem as defined in [18]. Given an undirected graph G = (V,E) with weighted edges and a subset of the vertices $T = \{t_1, ..., t_k\}$ which are called terminals, a multiway cut is a partition of $V$ into disjoint sets $S_1, ..., S_k$, such that $S_i$ contains $t_i$ for $i = 1, ..., k$. The goal in MMMC is to partition $V$ such that the maximum cut between any two partitions is minimized. If cut size is related to the cost of communication between two partitions, then MMMC seeks to *fairly* distribute this cost over all partitions. Therefore, we deal with a setting where local communication is relatively cheap, while communication between individual machines or domains is costly.

| Trial | B | $S$ | $\delta$ | $S^*$ | $\delta^*$ | $S/S^*$, $\delta/\delta^*$ |
|---|---|---|---|---|---|---|
| 1 | 2 | 1 | 2 | 1 | 2 | 1.000, 1.000 |
| | 5 | 2 | 6 | 3 | 5 | 0.667, 1.200 |
| | 8 | 3 | 11 | 7 | 8 | 0.428, 1.375 |
| | 11 | 5 | 12 | 10 | 11 | 0.500, 1.091 |
| | 14 | 5 | 12 | 14 | 14 | 0.357, 0.857 |
| | 17 | 299 | 17 | 299 | 17 | 1.000, 1.000 |
| 2 | 2 | 1 | 2 | 1 | 2 | 1.000, 1.000 |
| | 4 | 3 | 5 | 2 | 4 | 1.500, 1.250 |
| | 6 | 5 | 8 | 5 | 6 | 1.000, 1.333 |
| | 8 | 5 | 8 | 7 | 8 | 0.714, 1.000 |
| | 10 | 7 | 11 | 9 | 10 | 0.778, 1.100 |
| | 12 | 7 | 11 | 11 | 12 | 0.636, 0.917 |
| | 14 | 299 | 16 | 13 | 14 | 23.000, 1.143 |
| | 16 | 299 | 16 | 299 | 16 | 1.000, 1.000 |
| 3 | 2 | 1 | 2 | 1 | 2 | 1.000, 1.000 |
| | 5 | 4 | 6 | 3 | 5 | 1.333, 1.200 |
| | 8 | 5 | 12 | 6 | 7 | 0.833, 1.714 |
| | 11 | 5 | 12 | 10 | 11 | 0.500, 1.091 |
| | 14 | 9 | 16 | 12 | 14 | 0.750, 1.143 |
| | 17 | 299 | 17 | 299 | 17 | 1.000, 1.000 |

| Trial | B | $S$ | $\delta$ | $S^*$ | $\delta^*$ | $S/S^*$, $\delta/\delta^*$ |
|---|---|---|---|---|---|---|
| 1 | 13 | 916 | 13 | 916 | 13 | 1.000, 1.000 |
| | 344 | 6047 | 289 | 9049 | 342 | 0.668, 0.845 |
| | 675 | 8987 | 642 | 15041 | 675 | 0.598, 0.951 |
| | 1006 | 149667 | 1125 | 149377 | 991 | 1.002, 1.135 |
| | 1337 | 150765 | 1366 | 150501 | 1256 | 1.001, 1.088 |
| | 1668 | 151290 | 1668 | 151290 | 1668 | 1.000, 1.000 |
| 2 | 11 | 78 | 11 | 78 | 11 | 1.000, 1.000 |
| | 343 | 5844 | 459 | 6705 | 336 | 0.872, 1.366 |
| | 675 | 157520 | 606 | 156514 | 606 | 1.006, 1.000 |
| | 1007 | 157520 | 1051 | 157146 | 987 | 1.002, 1.065 |
| | 1339 | 158254 | 1672 | 157520 | 1051 | 1.005, 1.591 |
| | 1672 | 158254 | 1672 | 158254 | 1672 | 1.000, 1.000 |
| 3 | 15 | 631 | 15 | 631 | 15 | 1.000, 1.000 |
| | 387 | 8372 | 367 | 9895 | 385 | 0.846, 0.953 |
| | 759 | 12554 | 734 | 15957 | 756 | 0.787, 0.971 |
| | 1131 | 159297 | 1173 | 157260 | 1106 | 1.013, 1.061 |
| | 1503 | 160081 | 1474 | 160115 | 1471 | 0.999, 1.002 |
| | 1876 | 161490 | 1876 | 161490 | 1876 | 1.000, 1.000 |

**Tables 3 & 4:** Results of the MAXSBCC heuristic on unweighted and weighted Waxman graphs, respectively.

The 'fairness' aspect of MMMC distinguishes this work from a number of other resource placement models. In contrast, by minimizing the aggregate or average notion of cost, it is possible for some network participants to suffer a disproportionate amount of traffic. Here, we are not concerned with the number of resources allocated to a network entity, but with the cost of inter-partition traffic incurred by holding such items. With abundant availability of disk-space, we treat bandwidth and latency as principal aspects in our model.

### 3.1 Experimental Results

In [18], the authors show how a $(\alpha, \beta)$-approximation algorithm for MAXSBCC can be used to achieve an $(\alpha \log_\beta n)$-approximation for MIN-MAX MULTIWAY CUT. Here, we employ their result using our heuristic for MAXSBCC; we call this algorithm MMMC SOLVER. Due to space constraints, we refer the reader [18] for further details.

Using the data set from [11], the most widely held $2000$ mp3 files are extracted. Pair-wise correlations are computed between all data objects and we discard edges with a correlation value below a cutoff point $c$ which differs per trial. Edge capacities were then multiplied by a factor of $100$ and rounded to the nearest integer. For each trial, $\tau$ terminal nodes are chosen uniformly at random from the total number of nodes $N$ and the input bound was chosen to be an arbitrary value of $1000$. MMMC SOLVER is then run on the graph problem. Since no ILP formulation for MMMC is known, we compare against two other algorithms in order to evaluate the quality of our solution. The first is a simple random placement of nodes to partitions which models the behavior we would expect from employing a secure hash function; we denote this algorithm by RANDOM. The second is a greedy algorithm, denoted by GREEDY, that begins with a random assignment to partitions and then attempts to reduce the size of the maximum

| B | $S$ | $\delta$ | $S^*$ | $\delta^*$ | $S/S^*$, $\delta/\delta^*$ |
|---|---|---|---|---|---|
| 25 | 159975 | 25 | 159975 | 25 | 1.000, 1.000 |
| 115 | 160957 | 127 | 163526 | 113 | 0.984, 1.124 |
| 205 | 166445 | 228 | 165908 | 202 | 1.003, 1.129 |
| 295 | 167438 | 278 | 167108 | 278 | 1.002, 1.000 |
| 385 | 167825 | 354 | 168476 | 381 | 0.996, 0.929 |
| 476 | 176567 | 476 | 176567 | 476 | 1.000, 1.000 |

**Table 5:** Results of the MAXSBCC heuristic on real world data.

| Trial | c | $N$ | $\tau$ | RANDOM | GREEDY | MMMC SOLVER |
|---|---|---|---|---|---|---|
| 1 | 0.30 | 111 | 4 | 1290 | 831 | 34 |
| | | | | 1475 | 888 | 66 |
| | | | | 1222 | 875 | 39 |
| 2 | 0.28 | 379 | 5 | 2708 | 2273 | 163 |
| | | | | 2495 | 2190 | 300 |
| | | | | 2480 | 1962 | 136 |
| 3 | 0.26 | 813 | 6 | 3756 | 2990 | 152 |
| | | | | 3567 | 3179 | 238 |
| | | | | 3514 | 3114 | 219 |
| 4 | 0.24 | 1256 | 7 | 5246 | 4145 | 288 |
| | | | | 5208 | 4060 | 256 |
| | | | | 5297 | 4556 | 538 |

**Table 6:** results of testing RANDOM, GREEDY, MMMC SOLVER on MMMC test cases.

cut by greedily reassigning vertices. In particular, each non-terminal node involved in a maximum cut is tested in all other partitions. If such a relocation reduces the maximum cut, the new assignment is immediately kept; otherwise, the node remains at its original location. A solution is returned when no further reduction can be obtained.

Table 6 provides the results of our experiments. The longest running experiment consisted of $1256$ nodes with $7$ terminals and the running time of MMMC SOLVER was under $8$ hours. Over all four trials, MMMC SOLVER demonstrates superior performance in the size of the maximum cut. The discrepancy between RANDOM and MMMC SOLVER is significant although expected. More strikingly, the difference between GREEDY and MMMC SOLVER is substantial suggesting that the greedy approach becomes trapped in local optima which MMMC SOLVER is able to avoid. In particular, the maximum cut between any two partitions yielded by MMMC SOLVER is *never more than* $14\%$ of that yielded by GREEDY.

## 4 Correlation-Aware Resource Placement: Extending MMMC

Initially, the MIN-MAX MULTIWAY CUT problem appears to model almost any resource placement problem. However, this is not the case for two reasons:

1. *Cost Dependencies:* consider modelling a P2P network as discussed in the original work [18] where the terminals are peers and the remaining non-terminal vertices are data items. A data item belonging to $S_i$ is stored at peer $t_i$ and edge capacities reflect expected communication patterns. The goal is to allocate data items among the peers so as to minimize the expected communication cost. *However, costs will likely depend on where items are placed in the network.* Here, MMMC requires input for the edges and yet this input will be defined by the very solution we seek. Consequently, cost metrics need to be carefully motivated.
2. *Privacy:* there are often constraints on where resources can be placed in the network and, by itself, the MIN-MAX MULTIWAY CUT problem does not address this issue. This may be due to privacy issues where sensitive data can only be allocated to secure locations. Conversely, a server may refuse to maintain a particular re-

source given legal concerns or quality of service constraints. Resources may even be physically restricted to certain locations.

We now show how to solve these two problems and arrive at our correlation-aware resource placement scheme.

### 4.1 Cost Metrics Motivated by Correlation

In this section, we demonstrate how the use of correlation in our model avoids the problem of cost dependencies and motivates two important cost metrics. Overall, the main benefit of correlation information is that it is independent of location. Throughout, assume we are provided with positive correlation values between nodes in the network.[4]

**Latency as a Cost Metric:** consider resources $d_1$ and $d_2$, which are strongly correlated. They may be colocated in order to reduce the communication overhead involved in obtaining them both. For instance, in response to a query involving $d_1$, both resources $d_1$ and $d_2$ may be fetched in anticipation of a follow-up request for $d_2$; alternatively, less inter-machine communication may be required if both resources are located on one, or even a small number of machines, if substantial inter-machine communication is required for a query. Therefore, under scenarios where the size of resources is relatively small, the correlation values on our input graph to the MIN-MAX MULTIWAY CUT give rise to latency as a plausible cost metric.

This problem domain is suited to a number of applications. For instance, text search engines typically utilize inverted indices in order to be efficient. Primarily, an inverted index stores information matching a keyword to documents that contain it. A query with $K$ terms often requires that the inverted indices of all $K$ terms be accessed. For distributed search systems, these indices are placed on many different machines. Consequently, the communication overhead between machines storing the indices required for resolving the same query presents a critical factor in supporting fast search [23].

**Bandwidth as a Cost Metric:** for sizable data items, bandwidth becomes the dominating cost, not latency. Consider two large files $d_1$ and $d_2$ that are highly correlated in the sense that if a user obtains one, he is likely to obtain the other. As a simple example, $d_1$ and $d_2$ might be two jpeg files by a user's favorite artist. In this case, it does not matter whether $d_1$ and $d_2$ are colocated since our cost metric is dominated by bandwidth consumption which does not necessarily bear any relationship to the correlation value on the edge $(d_1, d_2)$ in our input graph.

The situation, however, is quite different for queries involving the collection of more than one data source. For instance, a user may wish to compute a function over the aggregation of $d_1$ and $d_2$. There are a number of settings where such complex queries are useful for allowing richer search capabilities. A range query might require a join operation on $d_1$ and $d_2$. Here, it makes sense to have $d_1$ and $d_2$ colocated; the query can be resolved without downloading of *at least one of* $d_1$ *or* $d_2$. Such complex queries motivate a meaningful relationship between correlation and bandwidth consumption.

[4] Our approach can incorporate negative correlations; however, for simplicity we restrict our attention to positive correlation in the context of our work.

### 4.2 Adding Privacy Constraints

We enforce privacy constraints by embedding them into the MAXSBCC sub-problem. The following primal semidefinite form for our SDP of Section 2 can be obtained from Equations (5)-(9) by standard methods:

$$\begin{aligned} \max\ C \cdot X\ s.t.\ & (1/2)A \bullet X = W \\ & E_{ii} \bullet X = 1, \text{for } 1 \leq i \leq n \\ & (1/2)E_{st} \bullet X = -1 \\ & X \succeq 0 \end{aligned}$$

where $P \bullet Q$ denotes the standard $\sum_i \sum_j P_{ij}Q_{ij}$. $B'$ is such that entry $b'_{11} = 1$ and all other entries in the first row and first column have value $1/2$; the rest of the entries are $0$. Then $C = (1/2)I + (1/2)B'$, $X_{ij} = \mathbf{v_i v_j}$, and $A$ is the capacity matrix for $G$. $E_{ij}$ is an $n \times n$ matrix with a 1 in the $ij^{th}$ and $ji^{th}$ entries and zeros everywhere else.

Using the primal form, privacy constraints are added in the following fashion. Assuming feasibility, if we wish to constrain the location of a particular resource $b$ to a terminal $v$, we include $(1/2)E_{v,b} \bullet X = 1$; alternatively, $(1/2)E_{v,b} \bullet X = -1$ ensures that $b$ will not be stored at $v$. We can also force resources $a$ and $b$ to be colocated or separted by setting $(1/2)E_{a,b} \bullet X = 1$ or $(1/2)E_{a,b} \bullet X = -1$, respectively. The mathematical analysis of Section 2 changes little and the results remain unchanged.

### 4.3 Experimental Results

Our experimental work is in the context of Section 4.1. Assume a homogenous multi-administrative network where users in the network are issuing text queries and the cost of resolving a query *within the domain of the particular issuer of the query* is inexpensive, while communication between administrative domains is costly. We consider the problem of placing inverted indices such that (1) the communication overhead between domains during query resolutions is reduced and (2) no domain is involved in an excessive number of transactions involving multiple domains.

We utilized the query data[5] of [11] which totals $5462$ queries, each consisting of several terms, by users in the network. Using the SMART 'stopword' list [4], queries were pruned to remove trivial terms.[6] From this data set, the $K$ most prevalent terms were extracted and correlation values between each pair were calculated. The most prevalent terms are not necessarily correlated with one another; therefore, we extracted the largest group of terms that did share positive correlations. Represented as a connected component where nodes are terms and edges are weighted by correlation values, this graph was used as our input. $10$ randomly chosen terminals were chosen to correspond to domains. There were $12$ trials in total, consisting of $53, 144, 238, 336, 423, 523, 630, 734, 847, 956, 1064$ and $1157$ nodes, respectively. In each trial, all $5462$ pruned queries were executed. For each term in a particular query that matched a top key word in our trial, data was collected; otherwise, the term was ignored.

**Distinct Domains Accessed per Query:** if terms within a single query require accessing multiple domains, then the number of unique domains accessed provides a measure

[5] All query data was used, not just queries related to mp3 files.
[6] Pruning did not remove queries, but trivial terms were removed from a large number of queries.

| Trial | GREEDY | MMMC SOLVER |
|---|---|---|
| 1 | 1218 | 1111 |
| 2 | 2247 | 1996 |
| 3 | 2942 | 2409 |
| 4 | 3439 | 2333 |
| 5 | 3808 | 2493 |
| 6 | 4191 | 2693 |
| 7 | 4561 | 2909 |
| 8 | 4886 | 3024 |
| 9 | 5285 | 3159 |
| 10 | 5555 | 3291 |
| 11 | 5858 | 3471 |
| 12 | 5997 | 3485 |

| Trial | GREEDY | MMMC SOLVER |
|---|---|---|
| 1 | 83.0% | 87.7% |
| 2 | 73.6% | 81.4% |
| 3 | 68.3% | 83.9.% |
| 4 | 64.8% | 96.8% |
| 5 | 62.4% | 94.7% |
| 6 | 59.4% | 97.6% |
| 7 | 58.3% | 96.8% |
| 8 | 56.8% | 97.8% |
| 9 | 55.2% | 98.8% |
| 10 | 54.4% | 98.8% |
| 11 | 53.6% | 97.8% |
| 12 | 53.1% | 99.4% |

| Trial | GREEDY | MMMC SOLVER |
|---|---|---|
| 1 | 7.0% | 10.0% |
| 2 | 9.7% | 12.1% |
| 3 | 11.5% | 13.7% |
| 4 | 11.7% | 3.2% |
| 5 | 14.5% | 5.3% |
| 6 | 13.7% | 2.4% |
| 7 | 14.6% | 3.2% |
| 8 | 14.7% | 2.2% |
| 9 | 16.1% | 1.3% |
| 10 | 15.7% | 1.2% |
| 11 | 15.9% | 2.1% |
| 12 | 16.2% | 5.8% |

**Tables 7:** Number of unique domain accesses per query aggregated over all queries. **Table 8:** Total percentage of queries that can be resolved within a single domain. **Table 9:** Of the remaining queries that require accessing two or more domains to resolve, the percentage attributed to the domain involved in the most number of such transactions.

of communication overhead per query. Table 7 illustrates the sum of such access data over all queries. In comparison with the placement scheme given by GREEDY, MMMC SOLVER achieves substantially fewer unique domain accesses over the course of executing all $5462$ queries. In particular, for Trial $4$ and above, MMMC SOLVER incurs *only* $68$*% down to* $58$*% of the unique domain accesses* performed by GREEDY.

**Queries Involving Multiple Domains:** the number of queries requiring communication between multiple domains concerns both the amount of communication overhead and also the aspect of fairness. Table 8 depicts data for both MMMC SOLVER and GREEDY on the number of queries that were resolved through a single domain only and the number of queries that required two or more domain accesses. For MMMC SOLVER, *at least* $81$*% of all queries could be resolved at a single domain*. Moreover, for Trial $4$ and above, this value grew to be $95\%$ or more. In contrast, a significantly smaller percentage of queries were resolved at a single domain using GREEDY. In terms of fairness, with MMMC SOLVER, no domain participated in transactions with other machines for more than $19\%$ of the queries. This value can be dissected further by examining how much of this $19\%$ is attributed to each domain. Table 9 gives this information; no domain is ever forced to participate in more than $14\%$ of these transactions involving more than one domain. Moreover, this value decreases substantially as the number of key words is increased, *dropping below* $6$*% after Trial* $4$.

## 5 Conclusion

In this work, we proposed a novel correlation-aware resource placement scheme. A heuristic was developed for solving the MAXSBCC problem. This heuristic is used as a critical sub-routine for solving MMMC which, after extensions to address cost metrics and privacy constraints, yields our correlation-aware resource placement scheme. The results of our experiments were encouraging and demonstrated that our scheme can yield substantial savings in communication overhead. Interesting future work includes

analyzing the performance benefits of using negative correlation information and parallelized implementations of our algorithms.

**Acknowledgements:** we gratefully thank Jared Saia for his helpful discussions.

## References

1. ILOG CPLEX. http://www.ilog.com/products/cplex/.
2. SDPA. http://homepage.mac.com/klabtitech/sdpa-homepage/.
3. BRITE. http://www.cs.bu.edu/brite/.
4. SMART. ftp://ftp.cs.cornell.edu/pub/smart/.
5. Vinay Aggarwal, Anja Feldmann, and Christian Scheideler. Can ISPs and P2P users cooperate for improved performance. *Computer Communication Review*, 37:3:29–40, 2007.
6. Ivan D. Baev and Rajmohan Rajaraman. Approximation algorithms for data placement in arbitrary networks. In $12^{th}$ *Annual ACM-SIAM Symposium on Discrete algorithms*, 2001.
7. Michael Carter, Holly Jin, Michael Saunders, and Yinyu Ye. Spaseloc: An adaptive subproblem algorithm for scalable wireless sensor network localization. *SIAM Journal on Optimization*, 17:4:1102–1128, 2006.
8. Uriel Feige and Robert Krauthgamer. A polylogarithmic approximation of the minimum bisection. In *IEEE Symposium on Foundations of Computer Science*, pages 105–115, 2000.
9. Bissan Ghaddar, Miguel Anjos, and Frauke Liers. A branch-and-cut algorithm based on semidefinite programming for the minimum k-partition problem. Unpub. Manuscript, 2007.
10. Michel Goemans and David Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42:6:1115–1145, 1995.
11. Shen-Tat Goh, Panos Kalnis, Spiridon Bakiras, and Kian-Lee Tan. Real datasets for file-sharing peer-to-peer systems. In *10th International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 201–213, 2005.
12. Peter Heywood. Caching in on p2p. www.lightreading.com document.asp? doc_id=34399.
13. Konstantinos Kalpakis and Parag Namjoshi. Haplotype phasing using semidefinite programming. In *Proc.* $5th$ *IEEE Symposium on Bioinformatics and Bioengineering*, 2005.
14. Christof Krick, Harald Racke, and Matthias Westermann. Approximation algorithms for data management in networks. In *SPAA'01*, pages 237–246, 2001.
15. Brian Kulis, Arun C. Surendran, and John C. Platt. Fast low-rank semidefinite programming for embedding and clustering. In *11th Int. Conf. on A.I. and Statistics*, pages 512–521, 2007.
16. Shlomo Moran, Satish Rao, and Sagi Snir. Using semi-definite programming to enhance supertree resolvability. In $5^{th}$ *Workshop on Algorithms in Bioinformatics (WABI)*, 2005.
17. Kazuhide Nakata, Makoto Yamashita, Katsuki Fujisawa, and Masakazu Kojima. A parallel primal-dual interior-point method for semidefinite programs using positive definite matrix completion. *Parallel Computing*, 32:1:24–43, 2006.
18. Zoya Svitkina and Éva Tardos. Min-max multiway cut. In *RANDOM-APPROX*, 2004.
19. Bernard M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6:9:1617–1622, 1988.
20. Haiyong Xie, Y. Richard Yang, Arvind Krishnamurthy, Yanbin Liu, and Avi Silberschatz. P4P: Provider portal for applications. In *SIGCOMM*, 2008.
21. Yinyu Ye and Jiawei Zhang. Approximation of dense-n/2-subgraph and the complement of min-bisection. *Journal of Global Optimization*, 25:1:55–73, 2003.
22. Yi Zhang, Samuel Burer, and W. Nick Street. Ensemble pruning via semi-definite programming. *Journal of Machine Learning Research*, 7:1315–1338, 2006.
23. Ming Zhong, Kai Shen, and Joel Seiferas. Correlation-aware object placement for multi-object operations. In *28th International Conference on Distributed Computing Systems (ICDCS)*, pages 512–521, 2008.

## Appendix

In this appendix, we include (1) the additional lemmas and the proofs that were omitted from Section 2 and (2) the full pseudocode that was omitted from Section 2.1.

### Analysis and Proofs for Section 2

Additional lemmas have been included which did not fit into the main paper; consequently, the lemmas are renumbered. Throughout, let $W^* \geq M$ denote the value of Equation (7) given by solving the semidefinite program and let $W \leq W^*$ denote its value after applying hyperplane rounding.

**Lemma 1.** *For $y \in [-1, 1]$, $(1 - \arccos(y)/\pi) > 0.945y$.*

*Proof.* We can rewrite the above inequality as $(1-\theta/\pi) \geq \alpha \cos\theta$ for $\theta \in [0, \pi]$. Define $f(\theta) = (1 - (\theta/\pi))/\cos\theta$ and note that $f(\theta)$ is convex and minimized at a unique value $x_{min} \in [0, \pi/2]$ such that $\tan x_{min} = 1/(\pi - x_{min})$. The root of $\pi \tan(x_{min}) - (x_{min}) \tan(x_{min}) - 1 = 0$ is $x_{min} > 0.3432$. Choose $x = 0.3432 < x_{min}$. Now, $f(0.3432) < 0.9459$ and because $f$ is concave, $f(0.3432) - \alpha = \epsilon > 0$ where we consider $\epsilon$ to be an error term. By obtaining a tight bound $\epsilon \leq \beta$, we can get an accurate estimate of $\alpha$ since $\alpha = f(0.3432) - \epsilon \geq f(0.3432) - \beta$. Let $f'$ be the derivative of $f$. Evaluating $f'(0.3432) < -6.2 \times 10^{-6} = m$ gives the slope of the tangent line tangent at $x = 0.3432$. We also have that $(0.3432 - x_{min}) < -6.6 \times 10^{-6}$. Finally, let $y_{min}$ be the actual value of $f$ at its minimum. Therefore $\epsilon = f(0.3432) - y_{min} = (0.3432 - x_{min})m < 10^{-10}$ and we have $\alpha \geq 0.9459 - 1.0 \times 10^{-10} > 0.945$.

**Lemma 2.** *Hyperplane rounding of the SDP provides a $W$ such that $E[W] \geq 0.945W^*$.*

*Proof.* Note that $\mathbf{v_i v_j} = \mathbf{1}$ iff $\mathbf{v_i}$ and $\mathbf{v_j}$ are on the same side of the hyperplane. Therefore, by Lemma 5, we have:

$$E[W] = \sum_{i<j} (1 - \theta_{ij}/\pi) w_{ij} \geq 0.945 \sum_{i<j} \mathbf{v_i v_j}\, w_{ij} = 0.945 W^*$$

noindentLet $S^*$ denote the value of Equation (5) given by the solution to the semidefinite program and let $S$ denote the value of this quantity after hyperplane rounding. The next lemma follows from previous work in [10]:

**Lemma 3.** *Hyperplane rounding of the SDP provides an $S$ such that $E[w(S)] \geq 0.878\, w(S^*)$.*

We now establish the following critical result:
**Lemma 4.** *Let $\epsilon$ be a small positive constant. With probability at least $1 - \frac{1}{n}$, hyperplane rounding need only be applied $\lceil 2 \ln n/\epsilon \rceil$ times before a rounded solution to the SDP is obtained such that $W \geq (1.823 - \lambda)W^*$ and $w(S) \geq \beta' w(S^*)$ where $1 \geq \lambda \geq 0.823$.*

*Proof.* We argue along similar lines as [21]. By Lemma 5 and Lemma 5, hyperplane rounding of the SDP program on lines 5-9 gives the following:

$$E[W] \geq \alpha W^* \text{ and } E[w(S)] \geq \beta w(S^*)$$

Define a random variable $Z$ such that $Z = \frac{W}{W^*} + \frac{w(S)}{w(S^*)}$. Note that:

$$E[Z] = \frac{E[W]}{W^*} + \frac{E[w(S)]}{w(S^*)} \geq \alpha + \beta > 1.823$$

There must exist a rounding trial $i$ where $Z$ achieves at least its expected value, that is $Z_i \geq \alpha + \beta$. On this trial, the rounded solutions to the contraints 5 and 7 of the SDP, $W_i$ and $S_i$ respectively, will satisfy $W_i \geq \alpha_i(W^*)$ and $w(S_i) \geq \beta_i w(S^*)$ where $\alpha_i + \beta_i = \alpha + \beta > 1.823$. Therefore, we are guaranteed that $\min\{\alpha_i, \beta_i\} > 0.823$ and, more precisely, we have:

$$W_i \geq (1.823 - \beta')W^* \text{ and } w(S_i) \geq \beta' w(S^*)$$

where $1 \geq \beta' \geq 0.823$. We now show that, with high probability, $\lceil 2 \ln n/\epsilon \rceil$ rounding attempts are sufficient to obtain a $Z$ that is arbitrarily close to its expectation. Let $p = Pr[Z \geq (1-\epsilon)E(Z)]$ where $\epsilon > 0$ is some small constant. Then, $E(Z) \leq (1-\epsilon)E(Z) + 2p$ which implies that $p \geq \frac{\epsilon E(Z)}{2} = \frac{\epsilon(\alpha+\beta)}{2}$. We then have:

$$(1-p)^{c \ln n} \leq (1 - \epsilon(\alpha+\beta)/2)^{2 \ln n/\epsilon} \leq e^{-(\alpha+\beta)\ln n} < n^{-1}$$

The next lemma concerns the cut found by our heuristic:

**Lemma 5.** *Let $\lambda \in (0,1]$ and $w(E) = \sum_{e_i \in E} c(e_i) = \sigma + \tau + \delta(S)$. If $W \geq \lambda W^*$, then $\delta(S) \leq \frac{w(E) - \lambda M}{2}$.*

*Proof.* Note that $W = \sigma + \tau - \delta(S)$. Then, by assumption $W = \sigma + \tau - \delta(S) \geq \lambda W^*$ and we have:

$$\begin{aligned} \delta(S) &\leq \sigma + \tau - \lambda W^* \\ &= \sigma + \tau + \delta(S) - \delta(S) - \lambda W^* \\ &\leq (w(E) - \lambda M)/2 \text{ since } W^* \geq M \end{aligned}$$

Finally, we address the quality of our solution:

**Theorem 1.** *With probability at least $1 - \frac{1}{n}$, for $\lambda \in [0.823, 1]$, the above algorithm achieves a cut $(S,T)$ such that:*

$$\begin{aligned} \delta(S) &\leq \lambda + \left(\frac{1-\lambda}{2}\right)\frac{w(E)}{B} \\ w(S) &\geq (1.823 - \lambda) \cdot w(S^*) \end{aligned}$$

*Proof.* Given a budget $B$, we want to set up our SDP by having $M = (w(E) - 2B)/\lambda_{actual}$. We know that $z$ will meet its expectation before too long and when that happens $\lambda_{actual} \in [0.823, 1]$. However, we do not know the exact value of $\lambda_{actual}$ prior to solving the SDP. Therefore, we can only guarantee that $\delta(S) \leq (w(E) - \lambda_{predict} M)/2 = B_{actual}$ where $\lambda_{predict}$ is our guess at the actual value $\lambda_{actual}$. As a result, $B_{actual}$ may not be the initially desired $B$ we were handed. One way of addressing this issue is to set $\lambda_{predict} = 1$ and then set $M = (w(E) - 2B)/\lambda_{predict}$ accordingly. For example, if $w(E) = 400$ and the desired bound is 100, one could set up the SDP with $M = 200$. Then, in the worst case, all executions of our algorithm

might set the actual value to $\lambda_{actual} = 0.823$. In this case, the SDP solved the problem using a bound $B_{actual} = (400 - 0.823 \cdot 200)/2 \approx 117$. Therefore, the approximation ratio of the cut we obtain is $\delta(S)/B$ which is at most $B_{actual}/B$. In this concrete example, we have $B_{actual}/B = 117/100$ and so we have set our input bound too high. We adopt this procedure and for $\lambda_{actual} = \lambda \in [0.823, 1]$ the ratio by which our cut can exceed the bound is:

$$\frac{\delta(S)}{B} \leq \frac{w(E) - \lambda M}{w(E) - M} = \lambda + \left(\frac{1-\lambda}{2}\right)\frac{w(E)}{B}$$

### MAXSBCC SOLVER Pseudocode

The pseudocode for MAXSBCC SOLVER is given below:

MAXSBCC SOLVER$(G, B_0, r)$

```
1: B ← B_0, j ← 0, S ← {}
2: for (i = 1 to r) do
3:    M ← W(E) − 2B
4:    (v, S, T, B_actual) ← ALG(M)
5:    if (v == false) then
6:       B ← 2 · B
7:    if ( v == true ∧ B_actual > B_0) then
8:       B ← B − ⌈B/2^j⌉
9:       j ← j + 1
10:   else if (v == true ∧ B_actual < B_0 ) then
11:      B ← B + ⌈B/2^j⌉
12:      j ← j + 1
13:   else
14:      Return S
15: Return S
```