Semidefinite Programming Based Algorithms for the Sparsest Cut Problem*

Luis A. A. Meira
Federal University of Sao Paulo, Brazil
augusto.meira@unifesp.br

Flávio K. Miyazawa
University of Campinas, Brazil
fkm@ic.unicamp.br

April 22, 2011

Abstract

In this paper we analyze a known relaxation for the Sparsest Cut Problem based on positive semidefinite constraints, and we present a branch and bound algorithm and heuristics based on this relaxation. The relaxed formulation and the algorithms were tested on small and moderate sized instances. It leads to values very close to the optimum solution values. The exact algorithm could obtain solutions for small and moderate sized instances, and the best heuristics obtained optimum or near optimum solutions for all tested instances. The semidefinite relaxation gives a lower bound $\frac{C}{W}$ and each heuristic produces a cut S with a ratio $\frac{c_S}{w_S}$, where either c_S is at most a factor of C or w_S is at least a factor of W. We solved the semidefinite relaxation using a semi-infinite cut generation with a commercial linear programming package adapted to the sparsest cut problem. We showed that the proposed strategy leads to a better performance compared to the use of a known semidefinite programming solver.

1 Introduction

Since the publication of the approximation algorithm for the Max-Cut Problem by Goemans and Williamson [8], the use of semidefinite programming has increased and it turns out to be an important technique to obtain good relaxations and algorithms for combinatorial optimization problems [8, 9, 21].

The Sparsest Cut Problem has applications in image segmentation [19], the metric labeling problem [3], and a natural application in graph conductance. It is a known NP-hard problem. We analyze the relaxation of the Sparsest Cut Problem formulation presented by Arora, Rao and Vazirani [2] and we propose four heuristics and an exact branch and bound algorithm for the Sparsest Cut Problem, based on semidefinite programming relaxation.

*This research was partially supported by CNPq and FAPESP.

These algorithms were tested on a set of random instances and, in all tests, optimum or near optimum solutions were produced.

Although we could not present approximation factors to the heuristics, we proved good characteristics for each one of them. We note that, assuming the Unique Games Conjecture of Khot [4], it is NP-hard to approximate the Sparsest Cut problem within any constant factor. We proved the existence of a bounded solution for the Sparsest Cut Problem when the expectation of both parts of the ratio are bounded. Considering that it is not always true that $E[X/Y] = E[X]/E[Y]$, this result has a general utility. See more details in Section 4.1. It is possible to apply this strategy to other problems with a ratio in the objective function.

The first approximation algorithm for the Sparsest Cut Problem has an $O(\log n)$ factor due to Leighton and Rao [15] in 1988. Recently, the interest in the Sparsest Cut Problem has increased. Arora et al. [2] present an $O(\sqrt{\log n})$-approximation algorithm based on a semidefinite program for the special case when all edge costs are in $\{0, 1\}$ and vertex weights are 1. For the general case, Chawla, Gupta and Räque [5] present an $O(\log^{3/4} n)$-approximation algorithm. The best known approximation algorithm is an $O(\sqrt{\log n} \log \log n)$-approximation algorithm due to Arora, Lee and Naor [1].

Devanur, Khot, Saket and Vishnoi [6] present a $\Theta(\log \log n)$ integrality gap instance for the considered semidefinite programming relaxation. In [18], Shmoys presents a survey on approximation algorithms for cut problems and their application to divide-and-conquer.

Semidefinite programming leads to good formulations for many optimization problems, but its use in practice is still limited due to the computational time consumed by the existing solvers. Moreover, there are no good warm start techniques for re-optimization. These drawbacks become clear after some computational tests performed with a semidefinite programming package based on the interior point method, see Section 6. To tackle these drawbacks, we used a cutting plane approach to solve semidefinite programs via a linear programming solver. In such approach, we take advantage of the mature state of the integer linear programming solvers and the use of fast warm start step after branching constraints or the addition of cutting planes.

The resolution of semidefinite programs by cutting plane methods has already been used by other authors. Krishnan and Mitchell [11, 12, 13] report the use of semi-infinite linear programming formulations to solve max-cut and min-bisection problems. Sherali and Fraticelli [17] explored connections between semidefinite programming and the Reformulation-Linearization technique in semi-infinite linear programming formulations.

To our knowledge, this is the first paper to present an exact algorithm for the Sparsest Cut Problem based on semidefinite programming.

2 Problem Definition

Given a graph $G = (V, E)$, a cost function $c : E \to \mathbb{R}^+$, a weight function $w : V \to \mathbb{R}^+$, and a set $S \subset V$, $S \neq \emptyset$, we denote by $\delta(S)$ the set of edges with exactly one extremity in S, $\mathcal{C}(S)$ the sum $\sum_{e \in \delta(S)} c_e$, $\mathcal{W}(S)$ the product $w(S)w(V \setminus S)$, where $w(C) := \sum_{v \in C} w_v$, and $\rho(S)$ the value $\mathcal{C}(S)/\mathcal{W}(S)$. The Sparsest Cut Problem can be defined as follows:

Sparsest Cut Problem (SC): Given a graph $G = (V, E)$, costs $c_e \in \mathbb{R}^+$ for each edge $e \in E$, and weights $w_v \in \mathbb{R}^+$ for each vertex $v \in V$, find a cut $S \subset V, S \neq \emptyset$, that minimizes $\rho(S)$. See Figure 1.

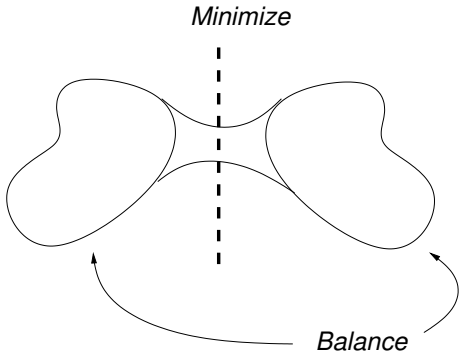

Figure 1: The objective of the Sparsest Cut Problem is to find a small balanced cut.

In the unweighted version, which we denote by USC Problem, we have $c_e = 1$ for each $e \in E$, and $w_v = 1$ for each $v \in V$.

The Sparsest Cut Problem is a natural formulation to discover bottlenecks in a road network. Suppose we have a city with two dense regions connected by few roads. In this case, it is possible that many cars go from one region to the other, and if the connecting roads do not have good flow capacity, or if there is a problem in one of the roads, they are prone to traffic jam. In the literature, there are many other problems that consider (vertex) partitions of a graph into two sets, as the Max Cut and Min Cut problems. The (unweighted) Max Cut problem looks for a partition of the graph into two sets, but connected with the maximum number of edges. The (unweighted) Min Cut problem looks for a partition of the graph into two sets connected by the minimum number of edges. In this problem, a minimum cut can divide the graph into two very unbalanced sets, and may not have any relation with bottlenecks. The weighted version of these problems consider weights in the edges and do not have any weight in the vertices. In Figure 2 we present a piece of map, where a solution for the Sparsest Cut problem is given by the closed curve. A straightforward application of the Sparsest Cut problem is to find regions where traffic jams may occur.

Another problem that is strongly related to the Sparsest Cut Problem is the Quotient Cut Problem, which can be defined as follows:

Min Quotient Cut Problem: Given a graph $G = (V, E)$, costs $c_e \in \mathbb{R}^+$ for each edge $e \in E$, and weights $w_v \in \mathbb{R}^+$ for each vertex $v \in V$, find a cut $S \subset V, S \neq \emptyset$, that minimizes $\mathcal{C}(S)/\min\{w(S), w(V \setminus S)\}$.

In [2], the Min Quotient Cut is also referred to as Sparsest Cut. In terms of approximability, an α-approximation algorithm for the Sparsest Cut (resp. Quotient Cut) Problem is a 2α-approximation algorithm for the Quotient Cut (resp. Sparsest Cut) Problem.

The following lemma can be deduced from the work of Leighton and Rao [15]. Because the proof is not straightforward, we present it here.

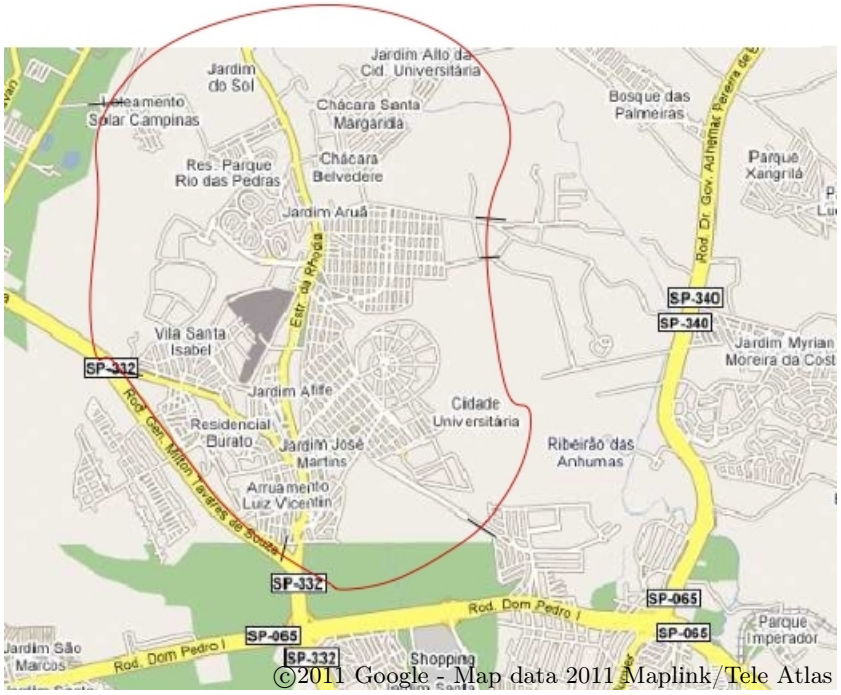

Figure 2: The Sparsest Cut Problem is the natural formulation to discover bottlenecks in a graph.

Lemma 2.1 *An α-approximation algorithm for the Sparsest Cut (resp. Quotient Cut) Problem is a 2α-approximation algorithm for the Quotient Cut (resp. Sparsest Cut) Problem.*

Proof. Let $\mathcal{A}_1$ be an α-approximation algorithm and I an instance for the Sparsest Cut Problem. Let S be the cut produced by the algorithm $\mathcal{A}_1(I)$. Without loss of generality, assume that $w(S) \leq w(V)/2$ and $w(V \setminus S) \geq w(V)/2$. Because $\mathcal{A}_1$ is an α-approximation algorithm, we have

$$\frac{\mathcal{C}(S)}{w(S)w(V \setminus S)} \leq \alpha \frac{\mathcal{C}(S')}{w(S')w(V \setminus S')}, \quad \forall S' \subset V, S' \neq \emptyset.$$

That is,

$$\begin{aligned}
\frac{\mathcal{C}(S)}{w(S)} &\leq \alpha \frac{\mathcal{C}(S')w(V \setminus S)}{w(S')w(V \setminus S')}, \quad \forall S' \subset V, S' \neq \emptyset \\
&\leq \alpha \frac{\mathcal{C}(S')}{\min\{w(S'), w(V \setminus S')\}} \frac{w(V)}{\max\{w(S'), w(V \setminus S')\}}, \quad \forall S' \subset V, S' \neq \emptyset \\
&\leq 2\alpha \frac{\mathcal{C}(S')}{\min\{w(S'), w(V \setminus S')\}}, \quad \forall S' \subset V, S' \neq \emptyset, \qquad (1)
\end{aligned}$$

where inequality (1) is valid because $w(V) \leq 2\max\{w(S'), w(V \setminus S')\}$.

Now, suppose that $\mathcal{A}_2$ is an α-approximation algorithm and I an instance for the Quotient Cut Problem. Let S be the cut produced by $\mathcal{A}_2(I)$. Without loss of generality, we suppose that $w(S) \leq w(V)/2$ and $w(V \setminus S) \geq w(V)/2$. Thus

$$\frac{\mathcal{C}(S)}{w(S)} = \frac{\mathcal{C}(S)}{\min\{w(S), w(V \setminus S)\}} \qquad (2)$$

$$\leq \alpha \frac{\mathcal{C}(S')}{\min\{w(S'), w(V \setminus S')\}}, \quad \forall S' \subset V, S' \neq \emptyset.$$

That is,

$$\begin{aligned}
\frac{\mathcal{C}(S)}{w(S)w(V \setminus S)} &\leq \alpha \frac{\mathcal{C}(S')}{\min\{w(S'), w(V \setminus S')\}\ w(V \setminus S)}, \quad \forall S' \subset V, S' \neq \emptyset \\
&= \alpha \frac{\mathcal{C}(S')}{w(S')w(V \setminus S')} \frac{\max\{w(S'), w(V \setminus S')\}}{w(V \setminus S)}, \quad \forall S' \subset V, S' \neq \emptyset \qquad (3) \\
&\leq 2\alpha \frac{\mathcal{C}(S')}{w(S')w(V \setminus S')}, \quad \forall S' \subset V, S' \neq \emptyset. \qquad (4)
\end{aligned}$$

Equality (3) is obtained multiplying the previous fraction by $\frac{\max\{w(S'),w(V\setminus S')\}}{\max\{w(S'),w(V\setminus S')\}}$. Inequality (4) is valid because $\max\{w(S'), w(V \setminus S')\} \leq w(V) \leq 2w(V \setminus S)$. □

3 Semidefinite Programming Formulation and Relaxation

We first present a formulation, using real variables, and then its relaxation, where each real variable is relaxed to an n-dimensional vector. Given an instance (G, c, w) for the Sparsest Cut Problem, the formulation uses a variable r_i, for each $i \in V_G$, that can assume value either r or $-r$, where $r \in \mathbb{R}^+$, indicating if the vertex is in one side or the other of the cut.

For each vertex $i \in V_G$, the fact that r_i has a value r or $-r$ allows us to scale r_i in a way that the denominator of the objective function ratio becomes 1 without changing the objective function. Note that $|r_i - r_j|^2$ is $4r^2$ if the edge (i, j) is chosen in the ratio, zero otherwise.

$$\begin{aligned}
&\rho_F = \min \sum_{i<j,(i,j)\in E} c_{ij}|r_i - r_j|^2 \\
&\text{s.t.} \\
&\sum_{i<j} w_i w_j |r_i - r_j|^2 = 1, \qquad (F) \\
&\qquad r_i \in \{r, -r\}, \quad \forall i \in V, \\
&\qquad r \geq 0, \\
&\qquad r_i, r \in \mathbb{R}, \quad \forall i \in V.
\end{aligned}$$

Lemma 3.1 *F is a* formulation *for the Sparsest Cut Problem.*

Proof. Let I be an instance for the Sparsest Cut Problem, $\mathcal{O}^* \subset V$ be an optimum solution to I and ρ_F be the value of the objective function of F solved with instance I. Now, we will prove that a solution for one problem leads to a solution for the other, with the same value.

Given a cut S, let $r = \frac{1}{\sqrt{4|S||V\setminus S|}}$, $r_i = r$ for each $i \in S$ and $r_i = -r$ for each $i \in V \setminus S$. The obtained attribution is feasible to F and has cost $\rho(S)$ for unweighted graphs. This is valid for any set S, including $\mathcal{O}^*$. In the weighted version, the same conclusion is valid for $r = \frac{1}{\sqrt{4\mathcal{W}(S)\mathcal{W}(V\setminus S)}}$.

To conclude that F is a formulation it is sufficient to show that any solution has an associated cut with the same cost. If $\mathcal{V} = (v, r)$ is a solution to F, we may obtain an associated cut $S_{\mathcal{V}} = \{i : r_i = -r\}$.

As $\mathcal{V}$ is feasible, $r = \frac{1}{\sqrt{4|S_{\mathcal{V}}||V \setminus S_{\mathcal{V}}|}}$ (in the weighted version $r = \frac{1}{\sqrt{4\mathcal{W}(S_{\mathcal{V}})\mathcal{W}(V \setminus S_{\mathcal{V}})}}$) and $\rho_F(\mathcal{V})$ values $\rho(S_{\mathcal{V}})$. □

For the rest of this paper, we consider $n = |V|$. If we relax r_i to an n-dimensional vector v_i in $\mathbb{R}^n$, we have a relaxation, which we denote by $\widehat{R}$. This is a relaxation, because, if all coordinates of v_i are equal to zero except the first, for each n-dimensional vector v_i, we have the formulation (F). To write the relaxation $\widehat{R}$ as a semidefinite program, we observe that, for $a, b \in \mathbb{R}^n$ with inner product $a \cdot b$, we have

$$\begin{aligned} \|a\| &= \sqrt{a_1^2 + a_2^2 + \ldots + a_n^2}, \\ a \cdot b &= \|a\| \, \|b\| \cos\theta = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n, \\ a - b &= (a_1 - b_1, a_2 - b_2, \ldots, a_n - b_n). \end{aligned}$$

Thus,

$$\|v_i - v_j\|^2 = (v_i - v_j) \cdot (v_i - v_j) = v_i \cdot v_i - 2 v_i \cdot v_j + v_j \cdot v_j. \tag{5}$$

We can strengthen the formulation F adding the following triangle inequalities:

$$\|v_i - v_j\|^2 \leq \|v_i - v_k\|^2 + \|v_k - v_j\|^2, \quad \forall i, j, k \in V.$$

Although there are $O(n^3)$ triangle inequalities, their insertions improved the convergency time of the exact Branch and Bound $(B\&B)$ and ensured good properties to the heuristics.

For instance, a graph with 60 vertices has more than 200,000 triangle inequalities.

For further information about semidefinite programming, see [9, 10, 14]. In the following we present the (weighted) relaxation $\widehat{R}$ and its corresponding semidefinite program R, where v_i is the n-dimensional vector associated with an arbitrary vertex $i \in V_G$ and $X \succeq 0$ means that X is positive semidefinite. A symmetric positive semidefinite matrix $X = (x_{ij})$ can be decomposed into a matrix V such that $X = \mathcal{V}^{\mathcal{T}} \cdot \mathcal{V}$, using a Cholesky decomposition algorithm. If we denote by v_i the i-th row of matrix V, we have $x_{ij} = v_i \cdot v_j$.

$(\widehat{R})$	(R)
minimize $\sum_{i<j} c_{ij} \|v_i - v_j\|^2$	minimize $\sum_{i<j} c_{ij}(x_{ii} + x_{jj} - 2x_{ij})$
such that	such that
$\|v_i - v_j\|^2 \leq \|v_i - v_k\|^2 + \|v_k - v_j\|^2 \quad \forall i, j, k \in V,$	$x_{ik} + x_{kj} - x_{ij} \leq x_{kk} \quad \forall i, j, k \in V,$
$\sum_{i<j} w_i w_j \|v_i - v_j\|^2 = 1,$	$\sum_{i<j} w_i w_j (x_{ii} + x_{jj} - 2x_{ij}) = 1,$
$\|v_i\| = \|v_0\| \quad \forall i \in V,$	$x_{ii} = x_{00} \quad \forall i \in V,$
$v_i \in \mathbb{R}^n.$	$X \succeq 0.$

We denote by $\widehat{R}_-$ (resp. R_-) the relaxation $\widehat{R}$ (resp. R) without the constraints $\|v_i\| = \|v_0\|$ (resp. $x_{ii} = x_{00}$). For unweighted graphs, Arora et al. [2] prove an integrality gap of $O(\sqrt{\log n})$ for

the relaxation $\widehat{R}_-$. Note that $\widehat{R}_-$ has only relative distances between vectors in the objective function and in all constraints. This implies that the objective function value does not change when all vectors v_j, for $j \in V$, are translated by the same displacement. An optimal solution to relaxation $\widehat{R}$ can also be translated. See Figure 3.

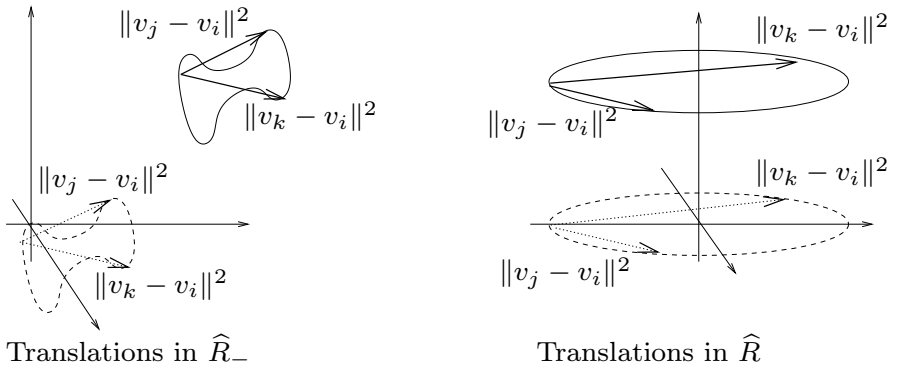

Figure 3: Translations in relaxations $\widehat{R}_-$ and $\widehat{R}$.

Definition 1 *An optimal solution X^* of R_- (resp. R) is said to be* small *if the corresponding solution v of $\widehat{R}_-$ (resp. $\widehat{R}$) minimizes $\sum_{i \in V} \|v_i\|$.*

To convert a non-small solution of R_- to a small one with the same objective value, it is sufficient to translate its barycenter to the origin. In the relaxation R it is sufficient to translate the hyperplane of small dimension that contains all vectors of the solution to the origin.

Given an instance I for the Sparsest Cut Problem, we denote by $R_-(I)$ and $R(I)$ the relaxations R_- and R defined with the corresponding values of I.

4 Heuristics for the Sparsest Cut Problem

In this section we propose some heuristics for the Sparsest Cut Problem. These heuristics receive as input parameter a feasible small solution $X^* = (x^*_{ij})$ to the relaxation $R_-(G, c, w)$ or $R(G, c, w)$ and then apply some rounding strategy to obtain a feasible solution.

In the following we present the heuristics *H1*, *H2*, *H3* and *H4*. The heuristics *H1* and *H2* use an input value $\xi(X^*, i, j)$ defined as $\xi(X^*, i, j) = \frac{x^*_{ij}}{\sqrt{x^*_{ii} x^*_{jj}}} = \cos(v_i, v_j)$. Note that the lower the value of $\xi(X^*, i, j)$, the greater is the separation of the vectors v_i and v_j. These algorithms choose a pair of vertices (s, t) such that $\xi(X^*, s, t)$ is minimum and randomly choose a real value in $[-1, 1]$ to say if a vertex is more close to s or not. Such pair (s, t) is chosen to minimize the probability of empty cuts. Although, all heuristics may generate empty cuts.

HEURISTIC *H1*$(G = (V, E), c, w, X^*)$

1. Let $s, t \in V$ such that $\xi(X^*, s, t)$ is minimum.
2. $S \leftarrow \emptyset$.
3. For each $u \in V$ do
4. select a random number α uniformly distributed in $[-1, 1]$
5. if $\xi(X^*, s, u) \geq \alpha$ then
6. $S \leftarrow S \cup \{u\}$.
7. Return(S).

HEURISTIC *H2*$(G = (V, E), c, w, X^*)$

1. Let $s, t \in V$ such that $\xi(X^*, s, t)$ is minimum.
2. $S \leftarrow \emptyset$
3. Select a random number α uniformly distributed in $[-1, 1]$.
4. For each $u \in V$ do
5. if $\xi(X^*, s, u) \geq \alpha$
6. $S \leftarrow S \cup \{u\}$.
7. Return(S).

HEURISTIC *H3*$(G = (V, E), c, w, X^*)$

1. Let $\mathcal{V} = (v_i)$ be the Cholesky decomposition of X^* ($X^* = \mathcal{V}^T\mathcal{V}$).
2. $S \leftarrow \emptyset$
3. Select a random vector $r \in \mathbb{R}^n$ uniformly distributed in the unit sphere.
4. For each $u \in V$ do
5. if $v_u \cdot r \geq 0$
6. $S \leftarrow S \cup \{u\}$.
7. Return(S).

HEURISTIC *H4*$(G = (V, E), c, w, X^*)$

1. Let $\mathcal{V} = (v_i)$ be the Cholesky decomposition of X^* ($X^* = \mathcal{V}^T\mathcal{V}$).
2. Let $A \leftarrow \emptyset$ and $B \leftarrow \emptyset$.
3. Select two vectors $r_1, r_2 \in \mathbb{R}^n$ uniformly distributed in the unit sphere.
4. For each $u \in V$ do
5. if $v_u \cdot r_1 >= 0$ and $v_u \cdot r_2 >= 0$ then $A \leftarrow A \cup \{u\}$
6. if $v_u \cdot r_1 < 0$ and $v_u \cdot r_2 < 0$ then $B \leftarrow B \cup \{u\}$.
7. Let G' be the graph obtained from G contracting A and B to vertices a and b, resp.
8. Let S' be a minimum cut separating a and b in G'.
9. Let S be the set of vertices in G corresponding to S'.
10. Return(S).

4.1 Some Properties of the Heuristics

Lemma 4.1 *Let $\mathcal{P}$ be a minimization problem with domain $\mathcal{D}$ and objective function $f(S) = \frac{g(S)}{h(S)}$, where $g(S) > 0$ and $h(S) > 0$ for each $S \in \mathcal{D}$. Let $\mathcal{A}$ be a randomized algorithm for $\mathcal{P}$. Given an instance I of $\mathcal{P}$ and a solution S produced by $\mathcal{A}(I)$, denote by G_S the random variable $g(S)$ and H_S*

the random variable $h(S)$ and let $\mathcal{E} = \frac{E[G_S]}{E[H_S]}$. Then, there exists a solution S' with $f(S') \leq \mathcal{E}$.

Proof. We have

$$\frac{E[G_S]}{E[H_S]} = \mathcal{E} \Rightarrow E[G_S] - \mathcal{E}E[H_S] = 0. \tag{6}$$

By the linearity of the expectation, we have

$$E[G_S - \mathcal{E}H_S] = 0.$$

This proves the existence of at least one solution S^{gap} such that

$$g(S^{gap}) - \mathcal{E}h(S^{gap}) \leq 0.$$

That is, $f(S^{gap}) \leq \mathcal{E}$. □

Fact 4.2 *Given a randomized algorithm $\mathcal{A}$ to problem USC, let C_S and W_S be the random variables $\mathcal{C}(S)$ and $\mathcal{W}(S)$, resp., where S is the cut produced by $\mathcal{A}$ over instance I. Let S^* be an optimal cut of I, $f^* = \rho(S^*)$ and $\beta \geq 1$ be such that $\frac{E[C_S]}{E[W_S]} = \beta f^*$. If Z is the random variable $C_S - \beta f^* W_S$ then $\Pr[Z \geq f^* \frac{n}{b}] \leq p$ for any $p \in (0,1)$ where $n = |V|$ and $b = \frac{p}{(\beta-1)(1-p)n}$.*

Proof. The expectation $E[Z]$ is equal to $E[C_S] - f^*\beta E[W_S]$. Substituting β we obtain that $E[Z] = 0$. We also have that $1 \cdot (n-1) \leq W_S \leq n^2$.

Since f^* is a lower bound for the value of any solution, we have $\frac{C_S}{W_S} \geq f^*$ and therefore,

$$\begin{aligned} Z &= C_S - \beta f^* W_S \\ &\geq -(\beta - 1) f^* W_S \\ &\geq -(\beta - 1) f^* n^2. \end{aligned}$$

For the sake of contradiction, suppose that $\Pr[Z \geq f^* \frac{n}{b}] > p$. In this case, we obtain that

$$\begin{aligned} E[Z] &\geq -(\beta - 1) f^* n^2 \Pr[Z < f^* \tfrac{n}{b}] + f^* \tfrac{n}{b} \Pr[Z \geq f^* \tfrac{n}{b}] \\ &> -(\beta - 1) f^* n^2 (1-p) + f^* \tfrac{n}{b} p \\ &= 0. \end{aligned}$$

This contradiction finishes the proof. □

Lemma 4.3 *Given a randomized polynomial time algorithm $\mathcal{A}$ to problem USC, let C_S and W_S be the random variables $\mathcal{C}(S)$ and $\mathcal{W}(S)$, resp., where S is the cut produced by $\mathcal{A}$ over instance I. Let S^* be an optimal cut of I and $\beta \geq 1$ be such that $\frac{E[C_S]}{E[W_S]} = \beta\rho(S^*)$. Then it is also possible to obtain a cut S' with $\rho(S') \leq (1+\epsilon)\beta\rho(S^*)$ in polynomial time with high probability, for any $\epsilon > 0$.*

Proof. Let $f^* = \rho(S^*)$. Consider a pair (p, b) that respects Fact 4.2 in such a way that $b = 2/\epsilon$ for any $\epsilon > 0$. With probability greater than $1 - p$, we have that

$$Z = C_S - \beta f^* W_S \leq f^* \frac{n}{b}.$$

Because $1 \cdot (n-1) \leq W_S$ and $n \geq 2$ we have $\frac{n}{b} \leq 2\frac{W_S}{b}$. That is, with probability greater than $1-p$, we have

$$\mathcal{C}_S - \beta f^* \mathcal{W}_S \leq f^* \frac{n}{b} \leq 2\frac{f^* \mathcal{W}_S}{b}.$$

Thus, with probability greater than $1-p$,

$$\begin{aligned} \frac{C_S}{W_S} &\leq \left(\beta + \frac{2}{b}\right) f^* = (\beta + \epsilon)\, f^* \\ &\leq (1+\epsilon)\, \beta \rho(S^*), \end{aligned} \tag{7}$$

where inequality (7) is valid because $\beta \geq 1$.

As $b = 2/\epsilon$, we have $p = \dfrac{1}{1 + \frac{\epsilon}{2(\beta-1)}}$. If we get a solution S' with minimum value from n executions, the probability that inequality (7) is not satisfied for S' is less than or equal to $p^n \leq \mathbf{e}^{-\frac{\epsilon}{2(\beta-1)}}$, where $\mathbf{e}$ is the base of the natural logarithm. Therefore, if β and ϵ are constants, the probability to obtain the desired cut is greater than a positive constant. We let to the interested reader to produce an algorithm that finds such a cut with high probability. □

Although we do not have the equality $E[A/B] = E[A]/E[B]$ in general, lemmas 4.1 and 4.3 describe a way to contour this difficulty for the Unweighted Sparsest Cut.

For the remaining of this section, we denote by $X^* = (x^*_{ij})$ a small optimum solution for $R(G, c, w)$ and by s and t two vertices where $\xi(X^*, s, t)$ is minimum. Given a solution X^* for relaxation R, we denote by Y^* the solution X^* scaled by a constant factor k such that $Y^* = kX^*$ and $Y^*_{ii} = 1$ for each vertex i.

Given a feasible solution X for R, and $\mathcal{V} = (v)$ its corresponding solution for $\widehat{R}$, denote by

$$\mathcal{C}(X) = \frac{1}{4}\sum_{i<j} c_{ij}(x_{ii} + x_{jj} - 2x_{ij}) = \frac{1}{4}\sum_{i<j} c_{ij}\|v_i - v_j\|^2,$$

$$\mathcal{W}(X) = \frac{1}{4}\sum_{i<j} w_i w_j(x_{ii} + x_{jj} - 2x_{ij}) = \frac{1}{4}\sum_{i<j} w_i w_j\|v_i - v_j\|^2,$$

and $\rho(X) = \mathcal{C}(X)/\mathcal{W}(X)$.

Observe that $\mathcal{C}(Y^*) = \sum_{i<j} c_{ij}\frac{1-y^*_{ij}}{2}$, $\mathcal{W}(Y^*) = \sum_{i<j} w_i w_j \frac{1-y^*_{ij}}{2}$ and $y^*_{ij} = \cos(v_i, v_j) = y^*_{ji}$.

Lemma 4.4 *The value $\rho(Y^*)$ is a lower bound for $\rho(S)$, for any set $S \subset V$, $S \neq \emptyset$.*

Proof. Let X^* be an optimum solution to R. As R is a relaxation of formulation F, $\rho(X^*)$ is a lower bound to the value of any optimum solution for the Sparsest Cut Problem. We have $\mathcal{C}(Y^*) = k\mathcal{C}(X^*)$, and $\mathcal{W}(Y^*) = k\mathcal{W}(X^*)$. Therefore, $\rho(Y^*) = \rho(X^*)$. □

To analyze the heuristics, we first consider that $\xi(X^*, s, t) = -1$. In this case, all heuristics generate non-empty cuts. We further present some ways to keep the same results when this condition is not true.

If each one of the four heuristics receives X^* as an input parameter, then they partially respect Lemma 4.1, but we could not obtain an approximation algorithm. In what follows, we show the strength and the weakness of each heuristic.

First consider heuristics *H1* and *H2*. These heuristics are, in some way, symmetric. While the heuristic *H1* guarantees that the expectation of $\mathcal{W}(S)$ is lower bounded by a constant factor of $\mathcal{W}(Y^*)$ (see Lemma 4.6), the heuristic *H2* guarantees that the expectation of $\mathcal{C}(S)$ is upper bounded by a constant factor of $\mathcal{C}(Y^*)$ (see Lemma 4.7).

It is also possible to obtain a feasible solution X^* for which Lemma 4.1 does not hold for heuristics *H1* and *H2*.

Fact 4.5 *If a and b are real numbers such that $|a| \leq 1$ and $|b| \leq 1$, then $\frac{1-ab}{2} \geq \frac{1}{2}\min\{\frac{1-a}{2}+\frac{1-b}{2}, \frac{1+a}{2}+\frac{1+b}{2}\}$.*

Proof. The proof is divided in two cases, depending on the signs of a and b.

Case 1: a and b have a same sign. Since $(a-b)^2 \geq 0$, we have that $-ab \geq -\frac{1}{2}(a^2+b^2)$ and therefore

$$\begin{aligned} 1-ab &\geq 1-\frac{a^2+b^2}{2} \\ &\geq 1-\frac{|a|+|b|}{2} \\ &= \min\{1-\frac{a+b}{2}, 1+\frac{a+b}{2}\} \\ &= \min\{\frac{1-a}{2}+\frac{1-b}{2}, \frac{1+a}{2}+\frac{1+b}{2}\}. \end{aligned}$$

Case 2: a and b have different signs: The proof of this case follows from the fact that $-ab \geq 0 \geq \min\{\frac{a+b}{2}, \frac{-(a+b)}{2}\}$, which gives us that

$$\frac{1-ab}{2} \geq \frac{1}{2}\min\{\frac{1-a}{2}+\frac{1-b}{2}, \frac{1+a}{2}+\frac{1+b}{2}\}.$$

□

The following lemma shows that heuristic *H1* produces a cut where $E[\mathcal{W}(S)]$ is at least a factor of $\mathcal{W}(Y^*)$.

Lemma 4.6 *If S is a solution produced by heuristic H1 and $\xi(X^*, s, t) = -1$, then $E[\mathcal{W}(S)] \geq \frac{\mathcal{W}(Y^*)}{2}$.*

Proof. For a vertex $u \neq s$, the probability that edge us is in $\delta(S)$ is $\frac{1-\xi(X^*,u,s)}{2} = \frac{1-\cos(v_u, v_s)}{2} = \frac{1-y^*_{us}}{2}$.

If u and v are vertices that are not equal to s then the probability that edge uv is in $\delta(S)$ is the probability that $(s,u) \in \delta(S)$ and $(s,v) \notin \delta(S)$ plus the probability that $(s,u) \notin \delta(S)$ and $(s,v) \in \delta(S)$. That is,

$$\begin{aligned} \Pr[(u,v) \in \delta(S)] &= \frac{1-y^*_{us}}{2}\left(1-\frac{1-y^*_{vs}}{2}\right) + \left(1-\frac{1-y^*_{us}}{2}\right)\frac{1-y^*_{vs}}{2} = \frac{1-y^*_{us}y^*_{vs}}{2} \\ &\geq \frac{1}{2}\min\{\frac{1-y^*_{us}}{2}+\frac{1-y^*_{vs}}{2}, \frac{1+y^*_{us}}{2}+\frac{1+y^*_{vs}}{2}\} \quad (8) \\ &\geq \frac{1}{2}\left(\frac{1-y^*_{uv}}{2}\right). \quad (9) \end{aligned}$$

The inequality (8) is valid from Fact 4.5, and inequality (9) is valid from the triangle inequality constraints of relaxation R. More precisely,

$$\frac{1-y^*_{uv}}{2} \leq \frac{1-y^*_{us}}{2} + \frac{1-y^*_{vs}}{2},$$

and

$$\begin{aligned} \frac{1-y^*_{uv}}{2} &\leq \frac{1-y^*_{ut}}{2} + \frac{1-y^*_{vt}}{2} \\ &= \frac{1+y^*_{us}}{2} + \frac{1+y^*_{vs}}{2}. \end{aligned} \tag{10}$$

In inequality (10) we use the fact that $\xi(X^*, s, t) = -1$. In this case, the angle between s and t is π, and we have $\cos(v_u, v_s) = -\cos(v_u, v_t)$.

The expectation of $\mathcal{W}(S)$ is

$$\begin{aligned} E[\mathcal{W}(S)] &= \sum_{u<v} w_u w_v \Pr[uv \in \delta(S)] \\ &= \sum_{u \in V\setminus\{s\}} w_u w_s \Pr[us \in \delta(S)] + \sum_{u \in V\setminus\{s\}} \sum_{v \in V\setminus\{s\} \mid v<u} w_u w_v \Pr[uv \in \delta(S)] \\ &\geq \sum_{u \in V\setminus\{s\}} w_u w_s \frac{1-y^*_{us}}{2} + \frac{1}{2} \sum_{u \in V\setminus\{s\}} \sum_{v \in V\setminus\{s\} \mid v<u} w_u w_v \frac{1-y^*_{uv}}{2} \\ &\geq \frac{1}{2} \sum_{u<v} w_u w_v \frac{1-y^*_{uv}}{2} \\ &= \frac{\mathcal{W}(Y^*)}{2}. \end{aligned}$$

□

In what follows, we present a feasible solution X^* where the expectation of $\mathcal{C}(S)$ for the cut produced by heuristic *H1* is unbounded. Consider the case where $(1-y^*_{uv})/2 = 0$, which means no contribution to $\mathcal{C}(Y^*)$ and $(1-y^*_{us})/2 = 1/2$. In this case the probability that (u, v) is in $\delta(S)$ is $1/2$, that implies a half contribution to the expectation of $\mathcal{C}(S)$. Adding *edge by edge*, each edge could have a similar behavior, and the expectation of $\mathcal{C}(S)$ will grow unbounded in relation to the value of $\mathcal{C}(Y^*)$. See Figure 4.

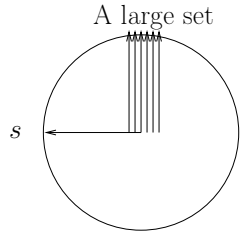

Figure 4: *H1* may produce a cut S with unbounded value of $\mathcal{C}(S)$.

Now, we consider the heuristic *H2*. In this case, we prove that $E[\mathcal{C}(S)]$ is at most $\mathcal{C}(Y^*)$, but we could not obtain a lower bound to $E[\mathcal{W}(S)]$.

Lemma 4.7 *If S is a solution produced by heuristic H2, then $E[\mathcal{C}(S)] \leq \mathcal{C}(Y^*)$.*

Proof. The probability that $(u, s) \in \delta(S)$ for $u \in V \setminus \{s\}$ is given by

$$\Pr[(u,s) \in \delta(S)] = \frac{1 - \xi(X^*, u, s)}{2} = \frac{1 - \cos(v_u, v_s)}{2} = \frac{1 - y^*_{us}}{2}.$$

If u and v are vertices that are not equal to s then

$$\Pr[(u,v) \in \delta(S)] = \left| \frac{y^*_{su} - y^*_{sv}}{2} \right| = \left| \frac{1 - y^*_{sv}}{2} - \frac{1 - y^*_{su}}{2} \right|,$$

that is exactly the probability to choose α between the values $\xi(X^*, s, v)$ and $\xi(X^*, s, u)$. The triangle inequality constraint implies that

$$\frac{1 - y^*_{sv}}{2} - \frac{1 - y^*_{su}}{2} \leq \frac{1 - y^*_{uv}}{2} \quad \text{and} \quad \frac{1 - y^*_{su}}{2} - \frac{1 - y^*_{sv}}{2} \leq \frac{1 - y^*_{uv}}{2}.$$

Therefore, we have

$$\Pr[(u,v) \in \delta(S)] = \left| \frac{1 - y^*_{su}}{2} - \frac{1 - y^*_{sv}}{2} \right| \leq \frac{1 - y^*_{uv}}{2}, \quad \text{for all } \ (u,v) \in V \times V,$$

and

$$E[\mathcal{C}(S)] = \sum_{u<v} c_{uv} \Pr[(u,v) \in \delta(S)] \leq \sum_{u<v} c_{uv} \frac{1 - y^*_{uv}}{2} = \mathcal{C}(Y^*).$$

□

To obtain a solution X^* where the expectation of $\mathcal{W}(S)$ is not lower bounded, consider an edge (u, v), where $y^*_{uv} = -1$ (full contribution to $\mathcal{W}(Y^*)$), and $y^*_{us} = y^*_{vs}$. In this case, we have $\Pr[(u,v) \in \delta(S)] = 0$. Symmetrically to *H1*, adding *edge by edge*, each edge may have a similar behavior, and the expectation of $\mathcal{W}(S)$ does not increase, becoming unbounded in relation to the value of $\mathcal{W}(Y^*)$. See Figure 5.

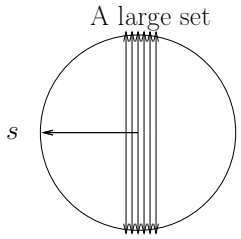

Figure 5: *H2* may produce a cut S with unbounded value of $\mathcal{W}(S)$.

The heuristic *H3* uses the hyperplane rounding strategy presented by Goemans and Williamson [8] for the Max-Cut problem. They proved that the probability that an edge (u, v) is cut by a random hyperplane is $\arccos(y^*_{uv})/\pi$. As can be seen in Figure 6, this value is always greater than $0.878(1 - y^*_{uv})/2$. Therefore, the following lemma is straightforward.

Lemma 4.8 *If S is a solution produced by heuristic H3, then $E[\mathcal{W}(S)] \geq 0.878\mathcal{W}(Y^*)$.*

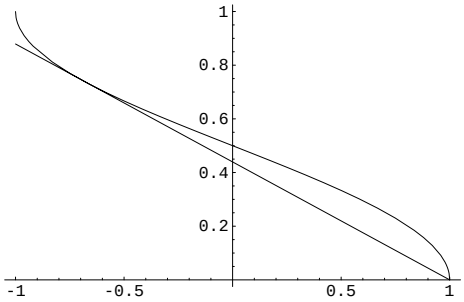

Figure 6: $\frac{\arccos(x)}{\pi}$ versus $0.878\left(\frac{1-x}{2}\right)$.

The difficulty to obtain an upper bound to $\Pr[(u,v) \in \delta(S)] = \frac{\arccos(y^*_{uv})}{\pi}$ comes from the fact that $\frac{\arccos(y^*_{uv})}{\pi}$ cannot be bounded by $k\frac{1-y^*_{uv}}{2}$ for any constant k (see Figure 6).

Analyzing the heuristic *H4*, we found out some interesting properties. The probability that (u,v) belongs to (A,B) is the probability that r_1 and r_2 separate the edge (u,v), that is equal to $(\arccos(x_{uv})/\pi)^2$ times the probability that $u \in A \cup B$, which is $1/2$. That is,

$$\Pr[(u,v) \in (A,B)] = \left(\frac{\arccos(x_{uv})}{\pi}\right)^2 \frac{1}{2}.$$

We can observe that this probability is upper and lower bounded by constant factors of the contribution in $\mathcal{C}(Y^*)$ and $\mathcal{W}(Y^*)$ (see Figure 7). More precisely, we have that $E[\mathcal{C}(A,B)] \leq \frac{1}{2}\mathcal{C}(Y^*)$ and $E[\mathcal{W}(A,B)] \geq 0.2023\mathcal{W}(Y^*)$. Therefore, the following lemma is valid.

Lemma 4.9 *If S is a solution produced by heuristic H4 then $E[\mathcal{W}(S)] \geq 0.2023\mathcal{W}(Y^*)$.*

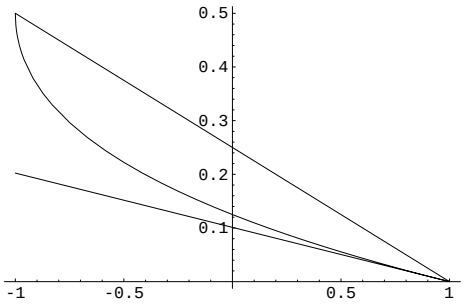

Figure 7: $0.2023\frac{1-x}{2}$ versus $\frac{1}{2}\frac{1-x}{2}$ versus $(\arccos(x)/\pi)^2\,\frac{1}{2}$.

Provided that the final cut S may have more edges than (A,B), we could not prove an upper bound for $E[\mathcal{C}(S)]$ by a factor of $\mathcal{C}(Y^*)$.

Now, consider the case when $\xi(X^*,s,t) > -1$. One way to deal with this situation is to assure that one edge ij is separated. In Section 5 we show in details how to do this separation. By solving the relaxation with this separation for an arbitrary vertex i and all $j \in V \setminus \{i\}$ and choosing the minimum, we can obtain a lower bound that contains a pair of vertices with $\xi(X^*,s,t) = -1$. Another way to deal with this situation is to discard empty cuts in the heuristics. In this case, $\mathcal{C}(S)$ and $\mathcal{W}(S)$ will be multiplied by the same factor and therefore we can maintain the same analysis, except for heuristic *H1* that uses the fact $\xi(X^*,s,t) = -1$.

5 A Branch and Bound Algorithm for the Sparsest Cut Problem

The idea used in the branch and bound algorithm, which we denote by $B\&B$, for the Sparsest Cut Problem is to use the relaxation R as a dual bound and the heuristics as primal bounds. Although the relaxation presents high time complexity, it leads to excellent lower bounds.

The first step consists of solving relaxation R, which leads to the first lower bound. The second step consists of applying the four heuristics based on this relaxation. Due to the fact that the heuristics are much faster than solving the relaxation, each heuristic can be repeated several times without substantially increasing overall time. During the execution, the $B\&B$ algorithm maintains the best solution obtained and its value as the current upper bound.

Initially, an arbitrary vertex is chosen – say, vertex 0. At each iteration, the algorithm chooses an active node from the $B\&B$ tree with the minimum value in the objective function and perform a branching step if a gap is found between the lower and upper bounds. The branching consists of setting the angle between v_i and v_0 to be 0 or π. When the angle between each pair of vertices is in $\{0, \pi\}$, then each vector v_i, for all $i \in V$, is in $\{-v_0, v_0\}$ and an integer solution is obtained.

Each node of the branch and bound tree is a semidefinite program Q obtained from the formulation R with some integrality constraints. Let X_Q be a solution of the program Q. The "most fractional variable" of X_Q, say $ifrac(X_Q)$, is a vertex f so that the angle v_f to v_0 is closest to $\pi/2$. Therefore, the most fractional variables in a node are the vertices f with $|\sin(v_f, v_0)|$ maximum. The angle between v_i and v_j can be computed from $\xi(X^*, i, j)$. The $B\&B$ algorithm maintains a max-heap using $|\sin(v_f, v_0)|$ as the key function for the heap.

In each step, the algorithm removes a node Q from the max-heap, and obtains its corresponding vertex f. If the value of a solution of the current node Q is smaller than the current upper bound, it produces two new nodes Q' and Q''. Node Q' is obtained from node Q by setting the angle between v_0 and v_f to π, which can be done by adding the constraint $x_{f0} = -x_{ff} = -x_{00}$. Node Q'' is obtained from node Q by setting the angle between v_0 and v_f to 0, which can be done by adding the constraint $x_{f0} = x_{ff} = x_{00}$.

The branch and bound algorithm is faster than the brute force algorithm (generates all possible cuts) when it comes to find an optimal solution and allows us to discover the integrality gap of R in graphs of the benchmark we have used. The branch and bound algorithm is presented in Figure 8.

A common strategy used in many branch and bound algorithms to solve NP-hard problems is to relax the integrality constraints in an ILP formulation and insert additional constraints to tighten its relaxation. As semidefinite programming is a generalization of linear programming, any linear program can also be solved via semidefinite programming. The difference is due to the fact that linear programming can be solved exactly and semidefinite programming can be solved within a small error ϵ, in polynomial time complexity in the input size and in $\log(1/\epsilon)$.

The use of SDP models in combinatorial optimization problems is relatively new if compared to LP. The decision to solve a problem via semidefinite programming or linear programming involves some aspects. The state of the art of LP solvers reached a point in which many robust, fast and stable implementations are encountered, such as Cplex and Xpress-MP. These solvers also allow integer programming formulations and features as pre-processing and branch-and-cut algorithms.

ALGORITHM $B\&B(P)$, where $P = R(G, c, w)$

1. Let $\mathcal{H}$ be a max-heap of semidefinite programs, where the key is given by the function $key(\cdot)$
2. $X_{UB} \leftarrow Heuristics(P)$
3. $\mathcal{H} \leftarrow \{P\}$.
4. While $\mathcal{H} \neq \emptyset$ do
5. $\quad Q \leftarrow RemoveMax(\mathcal{H})$ and $f \leftarrow ifrac(X_Q)$.
6. $\quad$ if $\rho(Q) < \rho(X_{UB})$ then
7. $\quad\quad Q' \leftarrow Q \cup \{x_{f0} = -x_{ff} = -x_{00}\}$
8. $\quad\quad Q'' \leftarrow Q \cup \{x_{f0} = x_{ff} = x_{00}\}$
9. $\quad\quad$ if $\rho(Q') < \rho(X_{UB})$ then
10. $\quad\quad\quad$ if $X_{Q'}$ is a feasible solution then $X_{UB} \leftarrow X_{Q'}$
11. $\quad\quad\quad$ else
12. $\quad\quad\quad\quad X' \leftarrow Heuristics(Q')$
13. $\quad\quad\quad\quad$ if $\rho(X') < \rho(X_{UB})$ then $X_{UB} \leftarrow X'$
14. $\quad\quad\quad\quad \mathcal{H} \leftarrow \mathcal{H} \cup \{Q'\}$.
15. $\quad\quad$ if $\rho(Q'') < \rho(X_{UB})$ perform the corresponding steps executed in lines 10–14 for Q''.
16. Return X_{UB}.

Figure 8: Semidefinite based branch and bound algorithm.

For the comparison between the time to solve SDP and LP relaxations in instances of equivalent size, see Table 1 (column *Av. CPU time*) and Table 2 in Section 6. The time to solve the associated linear program in these computational experiments was approximately 400 times faster than the one to solve the corresponding semidefinite program for instances with $n = 40$ vertices.

An important advantage of the use of linear programming is the use of *warm start* strategies to solve the relaxations associated with each node. Despite the existence of theoretical works describing the possibility of a good initial point for a family of problems with SDP programs [16], some solvers, such as SDPA package used in the experiments presented in this paper, have restrictions that do not allow this optimization (see additional information in Section 6). It is well known that ILP solvers may take advantage of the use of a previous basis to obtain a new solution after a branching step.

On the other hand, semidefinite programming has some advantages. The use of semidefinite constraints may generate better lower bounds that may reduce the number of visited nodes. Moreover, given an SDP solution X^*, we can take advantage of its geometric interpretation by obtaining a set $\mathcal{V}$ of vectors. This set of vectors was used in the rounding strategies of heuristics *H3* and *H4*. Given an SDP solution X^*, we can obtain a matrix $\mathcal{V}$ such as $X^* = \mathcal{V}\mathcal{V}^T$ by using the Cholesky Decomposition of X^*.

For many linear programming formulations that use modulus, the relaxation usually does not give useful information and the use of a branch and bound algorithm based on a semidefinite program may be competitive.

6 Computational Results

In the first attempt, we used the SDPA package [20] as an SDP solver. It is an implementation of a primal-dual interior-point method for semidefinite programming. Unfortunately, the *warm start* was not implemented in the SDPA package, and we had to recompute the semidefinite program in each node of the branch and bound tree. Our effort to compute a good initial point in the branch and bound procedure using the SDPA was not successful.

The experiments described in this section were performed by following the same approach used by Goemans and Williamson [8] for the Max-Cut Problem. We generated four types of instances: types A, B, C and D. In instances of type A, each vertex has weight 1, and each edge is selected with probability 0.5 to receive cost 1. If an edge is not selected, its cost is zero. In instances of type B, the vertex weight is a random rational number uniformly distributed in $[0, 50]$, and the edge cost is also a rational number uniformly distributed in $[0, 50]$. In instances of type C, each vertex has weight 1, and each edge is selected with probability $9/|V|$ to receive cost 1. The edge cost is zero if the edge is not selected. While type A instances correspond to dense graphs, type C instances correspond to sparse ones because the expected vertex degree is 9. In the instances of type D, we arbitrarily divided the set V in two disjoint sets, V_1 and V_2, where $|V_1| = |V_2|$. Each vertex has weight 1. The edges with both extremities in V_1 or with both extremities in V_2 are selected with probability 0.5 to receive cost 1. Edges linking vertices in V_1 with vertices in V_2 are selected with probability 0.25 to receive cost 1. All edges that were not selected have cost zero.

We produced a set of instances containing 20, 30 and 40 vertices. For each instance, we obtained an optimum relaxed solution using SDPA on a Xeon 2.4 GHz with 1024MB of RAM and Linux operational system. Table 1 presents the average computational time, in seconds, to solve each group of instances.

Type	Size	Number of Instances	Av. CPU Time (s)
A	20	400	155
	30	95	6668
	40	5	30564
B	20	400	108
	30	146	4305
	40	26	18718
C	20	400	137
	30	95	6568
	40	2	26660
D	20	400	166
	30	93	6695
	40	3	28980

Table 1: Time to solve R with the SDPA solver.

In order to make further investigation in formulation R, we removed the semidefinite constraint,

obtaining a new relaxation, named $R_X := R \setminus \{X \succeq 0\}$, which was solved with the XPRESS-MP linear programming solver. It should be noticed that program R_X is a relaxation of the Sparsest Cut formulation. Thus an optimum integer solution of R_X may not be a feasible solution. These computational results are presented in Table 2. The SDP gap (%) for an instance I is given by $(\frac{R(I)}{R_X(I)} - 1)100$. As we limited the computational time, the experiment with SDPA was not able to find a solution to all instances in Table 2. To compute the SDP gap we used the solver described in the following subsection.

Type	Size	Number of Instances	Average SDP Gap (%)	Max. SDP Gap (%)	Av. CPU Time (s)
A	20	400	0.3	6.0	0.55
	30	300	1.0	10.0	7.05
	40	20	3.0	13.0	71.6
B	20	400	0.0	0.0	0.41
	30	300	0.0	0.0	4.33
	40	20	0.0	0.0	28.62
C	20	400	0.1	5.0	0.50
	30	300	0.2	9.0	5.6
	40	20	0.0	1.0	39.35
D	20	400	0.3	6.0	0.52
	30	300	1.0	8.0	7.00
	40	20	3.0	10.0	63.42

Table 2: Results of relaxation R_X with no semidefinite constraint.

The XPRESS-MP solver obtained solutions to relaxation R_X more than 200 times faster than SDPA obtained solutions to relaxation R. The quality of the solution obtained for formulation R_X was at most 13% worst and, on average, less than 3% worst.

Based on these results we propose, in the next section, a new and faster way to solve the semidefinite programming formulation to the Sparsest Cut problem.

6.1 A cutting plane based SDP solver for the Sparsest Cut problem

The poor time performance of SDPA over Sparsest Cut Formulation led us to another approach to solve an SDP formulation. We present an algorithm that solves the relaxation R_X using a linear programming solver, adding violated SDP constraints until the problem becomes positive semidefinite, within a small error.

The strategy is based on the algorithm proposed by Fraticelli in subsection 3.3 of [7]. Given a symmetric matrix X, this algorithm returns $\mathbf{0}$ if $X \succeq 0$. Otherwise, the algorithm obtains a vector α, with $\|\alpha\| = 1$, so that $\alpha^T X \alpha < 0$. We denote by FRATICELLI(X) the Fraticelli algorithm applied on a square symmetric matrix X.

Given a linear program P, we denote by LP(P) an optimal solution of P. As in [7], we relax the

semidefinite constraint $X \succeq 0$ to the symmetric constraint $X = X^T$, where X^T is the transpose of X. The algorithm is described below:

Algorithm SDP Solver (max cx, $Ax \geq b$, $X \succeq 0$)

1. Relax the $X \succeq 0$ constraint to $X = X^T$
2. Let P be the linear program $P = (\max cx,\ Ax \geq b,\ X = X^T)$
3. $(x', X') \leftarrow \text{LP}(P)$
4. $\alpha \leftarrow$ Fraticelli(X')
5. While $(\alpha \neq \mathbf{0})$ do
6. add constraint $\alpha^T X' \alpha \geq 0$ to P
7. $(x', X') \leftarrow \text{LP}(P)$
8. $\alpha \leftarrow$ Fraticelli(X')
9. Return (x', X').

When this algorithm is applied to simple instances, we detected a very slow convergence, as the cut generated by the Fraticelli Algorithm was superficial. We improved the algorithm convergence time by using deeper cuts as stated in Lemma 6.1. The algorithm takes the result of the LP solver, say (x', X'), and increases the diagonal values of X'. When the constraint $X \succeq 0$ is relaxed, the semidefinite property is recovered faster when the algorithm uses deeper cuts.

Lemma 6.1 *Consider an arbitrary $\lambda \in \mathbb{R}^+$, a square symmetric matrix $X_{n\times n}$, and the matrix $\hat{X}$, where $\hat{x}_{ij} = x_{ij}$ if $i \neq j$, and $\hat{x}_{ij} = (1+\lambda)x_{ij}$ otherwise. If $\hat{X}$ is not positive semidefinite then there exists $\alpha \in \mathbb{R}^n$ such that $\alpha^T X \alpha < -\lambda \sum_i x_{ii}\alpha_i^2$.*

Proof. When $\hat{X}$ is not positive semidefinite, there exists a vector $\alpha \in \mathbb{R}^n$ such that $\alpha^T \hat{X} \alpha < 0$. As $\alpha^T \hat{X} \alpha = \alpha^T X \alpha + \lambda \sum_{\forall i} x_{ii}\alpha_i^2$, we have $\alpha^T X \alpha < -\lambda \sum_i x_{ii}\alpha_i^2$. ∎

From this lemma, one is able to infer that if $\hat{X} \not\succeq 0$, so $X \not\succeq 0$. When $\hat{X} \succeq 0$ and λ is close to zero, we can assume that $X \succeq 0$. Using this lemma, we choose λ as large as possible to obtain a vector α such that $\alpha^T \hat{X} \alpha < 0$ using the Fraticelli algorithm. The original violated constraint is $\alpha^T X \alpha < -\lambda \sum_i x_{ii}\alpha_i^2$ and the deeper cut added is $\alpha^T X \alpha \geq 0$. In the computational experiments, we consider X positive semidefinite when $\hat{X}$ is positive semidefinite and $\lambda < 10^{-8}$.

The constraint $-x_{00} \leq x_{ij} \leq x_{00}$ is an SDP valid inequality for the Sparsest Cut and we decided to insert this constraint for each pair $\{i, j\}$ in the beginning of the algorithm.

Although there exists more robust methods solving semidefinite programs, the proposed approach is very simple and allows to solve semidefinite programs with the use of traditional LP solvers. Moreover, the use of linear programming in a branch and bound method to solve semidefinite programs allows the use of a same linear program, with few different constraints in each node of the branch and bound tree. As many nodes have linear programs with few differences, it is possible to use only one linear program for the whole branch and bound tree and obtain solutions for one node modifying few constraints. We can also obtain a solution for one node faster, taking advantage of a previous LP basis.

We considered two experiments: in the first, the algorithm inserts all triangle inequalities in the beginning, whereas in the second only the violated triangle inequalities are iteratively inserted in

each node of the branch and bound tree. In this case, the algorithm searches for violated triangle constraints. This approach improves the time and use of memory, which allowed us to solve larger Sparsest Cut Instances.

As the heuristics are probabilistic, each one was executed 50 times on the same instance, returning the best obtained solution. Empty cuts were discarded.

For each instance and for each heuristic, except heuristic *H2*, we also saved a number k that is the iteration in which the respective heuristic found its best solution. As there are at most $|V| - 1$ solutions that can be obtained by heuristic *H2*, all these solutions are generated. For the heuristics *H1* and *H3* we also counted the empty cuts to compute the corresponding value of k. The time to solve the heuristics without the time to solve the relaxation R is negligible (less than 0.3% of the time to solve R).

In the *Visited Nodes* column in tables 4 and 6, only non-leaf nodes were considered in the branch and bound tree. The *integrality gap (%)* for an instance I is given by $(\frac{B\&B(I)}{R(I)} - 1)100$ and the *deviation factor (%)* of a heuristic H_i is given by $(\frac{H_i(I)}{B\&B(I)} - 1)100$.

The results obtained with all triangle inequalities inserted in the beginning are described in tables 3 and 4. The results obtained with triangle inequalities inserted only when necessary are described in tables 5 and 6.

Typ	Siz	Number of Instances	Integrality Gap(%)		Average Deviation Factor (%)				Maximum Deviation Factor (%)			
			Av.	Max	*H1*	*H2*	*H3*	*H4*	*H1*	*H2*	*H3*	*H4*
A	20	400	0.00	0.82	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.19
	30	300	0.00	0.26	0.15	0.00	0.00	0.00	24.55	1.15	0.00	0.00
	40	20	0.02	0.30	0.81	0.01	0.01	0.01	16.25	0.15	0.15	0.15
B	20	400	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	30	300	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	2.18	0.00
	40	20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
C	20	400	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	30	300	0.01	0.58	0.18	0.02	0.03	0.03	33.44	5.33	6.85	5.33
	40	20	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
D	20	400	0.00	0.20	0.04	0.00	0.00	0.00	7.69	0.35	0.00	0.00
	30	300	0.00	0.70	0.00	0.00	0.00	0.00	0.31	0.31	0.31	0.31
	40	20	0.01	0.03	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00

Table 3: Result using Xpress-MP and all triangle inequalities from the beginning.

Adding triangle inequalities on demand gave us the possibility to solve larger instances (sizes 50 and 60) because this approach is faster and presents a better use of memory. The instances were solved in half of the time if compared to the case when all triangle constraints are inserted in the beginning.

The new implementation using semi-infinite cuts and linear programming in problems with up to 200,000 constraints was at least 80 times faster and at most 1,500 times faster than using the Sdpa solver.

Typ	Siz	Number of Instances	Average k			Av. CPU Time (s)		Visited Nodes		SDP Cuts	
			H1	*H3*	*H4*	Heu	*B&B*	Av.	Max	Av.	Max
A	20	400	1.1	1.1	2.2	1.2	1.4	1.0	1	6.5	215
	30	300	1.5	1.2	2.3	27.2	36.8	1.0	2	17.5	665
	40	20	2.8	1.4	2.0	654.8	1070.5	1.0	1	85.9	976
B	20	400	1.0	1.1	2.2	0.4	1.1	1.0	1	2.7	90
	30	300	1.0	1.1	2.5	3.7	6.1	1.0	1	0.7	65
	40	20	1.0	1.0	2.5	19.9	22.2	1.0	1	0.2	4
C	20	400	1.0	1.1	2.0	1.0	1.2	1.0	1	4.0	105
	30	300	1.3	1.3	2.2	17.8	22.5	1.1	26	12.6	818
	40	20	1.0	1.1	2.2	113.9	120.9	1.0	1	5.0	84
D	20	400	1.3	1.2	2.3	1.3	1.5	1.0	1	7.4	119
	30	300	1.1	1.2	2.7	30.3	39.0	1.0	4	17.6	194
	40	20	3.4	2.0	6.2	576.5	826.8	1.0	1	75.2	514

Table 4: Result using XPRESS-MP and all triangle inequalities from the beginning.

Typ	Siz	Number of Instances	Integrality Gap(%)		Average Deviation Factor (%)				Maximum Deviation Factor (%)			
			Av.	Max	*H1*	*H2*	*H3*	*H4*	*H1*	*H2*	*H3*	*H4*
A	20	400	0.00	0.82	0.14	0.00	0.00	0.00	56.12	1.19	0.00	0.00
	30	300	0.00	0.29	0.09	0.00	0.00	0.00	22.22	1.15	0.00	0.00
	40	20	0.02	0.32	1.13	0.01	0.01	0.01	22.59	0.15	0.15	0.15
	50	20	0.02	0.25	0.00	0.00	0.00	0.00	0.05	0.05	0.05	0.00
	60	20	0.04	0.44	0.57	0.05	0.05	0.03	6.60	0.95	0.95	0.63
B	20	400	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	30	300	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	40	20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	50	20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	60	20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
C	20	400	0.00	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	30	300	0.01	0.58	0.16	0.00	0.01	0.01	30.72	1.42	1.42	1.42
	40	20	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	50	20	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	60	20	0.32	5.18	3.93	0.00	0.00	1.08	62.85	0.00	0.00	17.31
D	20	400	0.00	0.20	0.02	0.00	0.00	0.00	3.43	0.81	0.00	0.00
	30	300	0.00	0.70	0.00	0.00	0.00	0.00	0.31	0.31	0.31	1.02
	40	20	0.01	0.07	0.18	0.00	0.00	0.00	3.65	0.00	0.01	0.01
	50	20	0.01	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	60	20	0.01	0.03	0.00	0.00	0.00	0.05	0.00	0.00	0.00	1.05

Table 5: Result using XPRESS-MP and triangle inequalities on demand.

Typ	Siz	Num of Inst.	Average k			Av. CPU Time (s)		Visited Nodes		SDP Cuts		Trig Cuts (×1000)	
			H1	*H3*	*H4*	Heu	*B&B*	Av.	Max	Av.	Max	Av.	Max
A	20	400	1.2	1.1	2.1	0.6	0.7	1.0	1	5.6	240	2.2	6.0
	30	300	1.4	1.3	2.2	11.9	16.9	1.0	3	17.3	970	9.4	19.6
	40	20	2.5	1.4	2.4	351.6	515.7	1.0	1	85.2	927	27.5.3	48.5
	50	20	1.9	1.5	2.2	2560	6637	1.1	2	133.5	1076	61.9	103.0
	60	20	2.6	2.8	4.5	15812	29055	1.1	2	285.9	2162	112.6	189.9
B	20	400	1.0	1.0	1.9	0.3	0.9	1.0	1	0.0	0	1.6	2.4
	30	300	1.0	1.0	1.9	2.4	3.5	1.0	1	0.0	0	5.9	8.7
	40	20	1.0	1.0	1.9	11.8	14.2	1.0	1	0.0	0	14.4	16.7
	50	20	1.0	1.0	2.2	43.9	45.7	1.0	1	0.0	0	27.9	34.5
	60	20	1.0	1.0	1.9	129.2	132.2	1.0	1	0.0	0	50.1	56.7
C	20	400	1.1	1.1	1.9	0.5	0.7	1.0	1	3.6	124	2.3	6018
	30	300	1.3	1.2	2.5	11.3	14.1	1.1	21	7.9	449	12.4	23549
	40	20	1.0	1.1	1.9	75.7	79.2	1.0	1	2.9	57	34.1	48797
	50	20	1.0	1.0	1.6	315.2	334.5	1.0	1	0.0	0	74.6	92456
	60	20	3.7	2.4	4.1	3893	5913	1.1	3	26.1	417	122.5	172.3
D	20	400	1.3	1.1	2.2	0.5	0.7	1.0	17	8.2	777	1.8	6.8
	30	300	1.2	1.2	2.4	8.3	17.7	1.0	8	16.2	305	7.2	22.0
	40	20	2.3	3.0	2.9	244.1	393.1	1.1	2	78.5	657	23.4	54.3
	50	20	1.0	1.4	3.0	744.8	1038	1.0	1	43.7	247	50.9	108.7
	60	20	1.3	1.8	4.5	2548	3687	1.0	1	66.4	291	95.1	195.5

Table 6: Result using XPRESS-MP and triangle inequalities on demand.

The relaxation R has a surprisingly good integrality gap. For almost all tests the value obtained from the relaxation was equal to the value of an optimum solution. The quality of the relaxation leads to good heuristics and the branch and bound algorithm visited a small number of nodes in the branch and bound tree.

The performance of the branch and bound algorithm depends basically on the semidefinite programming solver. Naturally, the $B\&B$ algorithm with formulation R may also become faster by using better solvers.

7 Conclusion

In this paper we analyze a known relaxation for the Sparsest Cut Problem using semidefinite programming and present an exact algorithm and heuristics based on this formulation. The heuristics obtained solutions with values very close to the optimum, and the exact algorithm can be executed on small and medium sized instances. We present two lemmas, Lemma 4.1 and Lemma 4.3, that deal with the expectation of a ratio of random variables. Such expectation is not trivial and the proposed technique can be used in other problems.

We implemented a new semidefinite programming solver based on linear programming and cutting

plane approach. This solver has a good performance and it is very promising for SDP problems dominated by linear programming constraints as well as for general SDP problems.

The performed tests showed that the use of semidefinite programming as a generalization of linear programming is a promising modeling strategy to other combinatorial optimization problems.

8 Future Works

One part of this work describes a way to solve semidefinite programs with more than two hundred thousand constraints, applied to a specific problem, in less CPU time, when compared with a well known semidefinite programming solver. As the described method is general, it is also promising to use the presented approach to solve general semidefinite programs and to compare with other solvers with instances of public benchmarks.

9 Acknowledgements

We would like to thank Cristina G. Fernandes and Marcelo D. Passos for their helpful suggestions for this paper.

References

[1] Sanjeev Arora, James R. Lee, and Assaf Naor. Euclidean distortion and the sparsest cut. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 553–562, New York, NY, USA, 2005. ACM.

[2] Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 222–231, New York, NY, USA, 2004. ACM.

[3] Evandro C. Bracht, Luis, A. A. Meira, and F. K. Miyazawa. A greedy approximation algorithm for the uniform metric labeling problem analyzed by a primal-dual technique. *ACM Journal of Experimental Algorithmics*, 10:2.11, 2005.

[4] S. Chawla, R. Krauthgamer, R. Kumar, Y. Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. In *20th Annual IEEE Conference on Computational Complexity*, pages 144–153, June 2005.

[5] Shuchi Chawla, Anupam Gupta, and Harald Räcke. Embeddings of negative-type metrics and an improved approximation to generalized sparsest cut. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 102–111, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[6] Nikhil R. Devanur, Subhash A. Khot, Rishi Saket, and Nisheeth K. Vishnoi. Integrality gaps for sparsest cut and minimum linear arrangement problems. In *STOC '06: Proceedings of the*

thirty-eighth annual ACM symposium on Theory of computing, pages 537–546, New York, NY, USA, 2006. ACM Press.

[7] Barbara M. P. Fraticelli. *Semidefinite Cuts and Partial Convexification Techniques with Applications to Continuous Nonconvex Optimization, Stochastic Integer Programming, and Facility Layout Problems*. PhD thesis, Virginia Polytechnic Institute, 2001.

[8] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the Association for Computing Machinery*, 42:1115–1145, 1995.

[9] C. Helmberg. Semidefinite programming for combinatorial optimization. *Habilitationsschrift*, ZIB-Report ZR-00-34, Konrad-Zuse-Zentrum, October 2000.

[10] Thomas Hofmeister and Martin Hühne. Semidefinite programming and its applications to approximation algorithms. In *Lectures on Proof Verification and Approximation Algorithms. (the book grow out of a Dagstuhl Seminar, April 21-25, 1997)*, pages 263–298, London, UK, 1998. Springer-Verlag.

[11] K. Krishnan and J. E. Mitchell. Semi-infinite linear programming approaches to semidefinite programming (SDP) problems. In P. M. Pardalos and H. Wolkowicz, editors, *Novel approaches to hard discrete optimization problems*, volume 37 of *Fields Institute Communications Series*, pages 123–142. AMS, 2003.

[12] K. Krishnan and J. E. Mitchell. A unifying framework for several cutting plane methods for semidefinite programming. 21(1):57–74, 2006.

[13] K. Krishnan and J.E. Mitchell. A semidefinite programming based polyhedral cut-and-price approach for the maxcut problem. 33(1):51–71, 2006.

[14] M. Laurent and R. Rendl. Semidefinite programming and integer programming. In K. Aardal, G. Nemhauser, and R. Weismantel, editors, *Handbook on Discrete Optimization*, pages 393–514. Elsevier B.V., December 2005.

[15] T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. *Proc. 29th Ann. IEEE Symp. on Foundations of Comput. Sci., IEEE Computer Society, 422-431.*, 1988.

[16] J. E. Mitchell. Restarting after branching in the SDP approach to MAX-CUT and similar combinatorial optimization problems. *Journal of Combinatorial Optimization*, 5(2):151–166, 2001.

[17] Hanif D. Sherali and Barbara M. P. Fraticelli. Enhancing rlt relaxations via a new class of semidefinite cuts. *J. of Global Optimization*, 22(1-4):233–261, 2002.

[18] David B. Shmoys. Cut problems and their application to divide-and-conquer, 1996. In D. Hochbaum, editor, Approximation Algorithms for NP-Hard Problems, pages 192–235. PWS Publishing.

[19] S. Wang and J. Siskind. Image segmentation with ratio cut. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(6):675–690, 2003.

[20] Makoto Yamashita. SDPA - Semidefinite programming solver, 2002. available at: http://grid.r.dendai.ac.jp/sdpa/.

[21] Heng Yang, Yinyu Ye, and Jiawei Zhang. An approximation algorithm for scheduling two parallel machines with capacity constraints. *Discrete Appl. Math.*, 130(3):449–467, 2003.