An Experimental Evaluation of Semidefinite Programming and Spectral Algorithms for Max Cut

RENEE MIRKA and DAVID P. WILLIAMSON, Cornell University, USA

We experimentally evaluate the performance of several Max Cut approximation algorithms. In particular, we compare the results of the Goemans and Williamson algorithm using semidefinite programming with Trevisan's algorithm using spectral partitioning. The former algorithm has a known .878 approximation guarantee whereas the latter has a .614 approximation guarantee. We investigate whether this gap in approximation guarantees is evident in practice or whether the spectral algorithm performs as well as the SDP. We also compare the performances to the standard greedy Max Cut algorithm which has a .5 approximation guarantee, two additional spectral algorithms, and a heuristic from Burer, Monteiro, and Zhang (BMZ). The algorithms are tested on Erdős–Renyi random graphs, complete graphs from TSPLIB, and real-world graphs from the Network Repository. We find, unsurprisingly, that the spectral algorithms provide a significant speed advantage over the SDP. In our experiments, the spectral algorithms and BMZ heuristic return cuts with values which are competitive with those of the SDP.

CCS Concepts: • **Theory of computation** → **Graph algorithms analysis**; *Approximation algorithms analysis;*

Additional Key Words and Phrases: Max cut, spectral algorithms

ACM Reference format:
Renee Mirka and David P. Williamson. 2023. An Experimental Evaluation of Semidefinite Programming and Spectral Algorithms for Max Cut. *ACM J. Exp. Algor.* 28, 1, Article 2.1 (August 2023), 18 pages.
https://doi.org/10.1145/3609426

1 INTRODUCTION

Given as input a graph $G = (V, E)$ and weights $w_e \in \mathbb{R}^+$ for all $e \in E$, the Max Cut problem asks to partition V into two sets such that the sum of the weights of the edges crossing the partition is maximized. In particular, a cut is given by a pair of sets (S, T) such that $V = S \cup T$ and $S \cap T = \emptyset$. The value of this cut is

$$\sum_{(s,t)\in E: s\in S, t\in T} w_{(s,t)},$$

and Max Cut seeks to find a cut maximizing this quantity.

Max Cut is a problem of vast theoretical and practical significance. It is polynomial solvable for certain classes of graphs, e.g., planar graphs [11, 18], and is well-known to be NP-hard in

An extended abstract of this work appeared in the 20th Symposium on Experimental Algorithms.
The authors were partially supported by the National Science Foundation under grant number CCF-2007009.
Authors' address: R. Mirka and D. P. Williamson, Cornell University, Ithaca, NY 14850; emails: {rem379, davidpwilliamson}@cornell.edu.

1084-6654/2023/08-ART2.1 $15.00
https://doi.org/10.1145/3609426

general; it appears on Karp's original list of NP-complete problems [15]. Additionally, Max Cut has applications in fields such as data clustering [19], circuit design, and statistical physics [2]; see Poljak and Tuza for a comprehensive survey [20].

Many researchers have made improvements towards exact solvers for Max Cut. For general graphs of unbounded average degree, Williams presented a Max Cut algorithm using exponential space to exactly solve (and count the number of optimal solutions) in $O(m^3 2^{\omega n/3})$ time where $\omega < 2.376$ [27]. Croce, Kaminski, and Paschos introduced an algorithm to find a Max Cut in graphs with bounded maximum degree, Δ, running in $O^*(2^{(1-2/\Delta)n})$ time where $O^*()$ suppresses polynomial factors [6]. Golovnev improved this to $O^*(2^{(1-3/(\Delta+1))n})$ [10]. Results from Hrga et al., Hrga and Povh, Krislock, Malick, and Roupin, and Rendl, Rinaldi, and Wiegele utilize branch and bound techniques to produce other exact solvers [13, 14, 17, 22]. However, due to the lack of an efficient (polynomial-time) algorithm, researchers have also considered finding good approximation algorithms. An α-approximation algorithm is a polynomial-time algorithm which guarantees a solution with a value of at least an α fraction of the optimal solution. As one of the most well-studied problems in theoretical computer science, there is a breadth of known approximation algorithms for Max Cut varying in runtime and approximation guarantee quality.

The simplest randomized approximation algorithm assigns a vertex $v \in V$ to either S or T with equal probability. In expectation, this is a .5-approximation algorithm. Another .5-approximation can be achieved through a simple greedy algorithm presented by Sahni and Gonzalez [24]. In this algorithm, start with $S, T = \emptyset$. While there are still unassigned vertices, any unassigned vertex v is chosen and the quantities $c_S(v) = \sum_{u \in S:(u,v) \in E} w_{(u,v)}$ and $c_T(v) = \sum_{u \in T:(u,v) \in E} w_{(u,v)}$ are computed. If $c_S(v) > c_T(v)$, v is assigned to T and otherwise to S.

The .5-approximation guarantee was the best known until Goemans and Williamson [9] presented a .878-approximation algorithm, which is the best possible guarantee assuming the Unique Games Conjecture [16]. Their algorithm relies on a semidefinite programming (SDP) relaxation of the Max Cut problem to find a high-value cut. While the approximation guarantee likely cannot be surpassed by another polynomial-time algorithm, solving the SDP can be quite costly in practice.

More recently, Trevisan [26] introduced a simple .531-approximation for Max Cut based on spectral partitioning. Soto [25] improved this guarantee to .614. Though the approximation guarantees are weaker than the SDP algorithm, the spectral techniques are much cheaper to implement. In theory, there is a tradeoff between the computational speed and solution quality of Goemans and Williamson's SDP algorithm versus Trevisan's spectral algorithm. This article seeks to determine whether this tradeoff exists in practice or if Trevisan's algorithm returns solutions competitive with those of the SDP.

It is common practice to apply a local search procedure at the end of a Max Cut algorithm or heuristic to locally optimize the found cut. One such local search procedure repeatedly moves a single vertex at a time from one side of the cut to the other, as long as the move increases the value of the cut. It terminates when there are no more vertices that can be moved to increase the value of the cut. In this work, we implement this local search method and apply it to all procedures in our experiments.

Several previous articles have experimentally compared Max Cut algorithms and heuristics. Bertoni, Campadelli, and Grossi compare cuts computed by their .39-approximation Lorena algorithm, inspired by Goemans–Williamson SDP, to the SDP and a neural .5-approximation algorithm [4]. They found, on average, Lorena provided larger cuts on random graphs in significantly less time than the SDP and comparable time to the neural algorithm. Dolezal, Hofmeister, and Lefmann compare cuts from six algorithms, including the SDP, on random graphs concluding that the computationally-cheap random .5-approximation algorithm provides the best tradeoff between runtime and cut quality [7]. Goemans and Williamson also included computational results in their

initial article demonstrating the SDP often outperforms its .878 approximation guarantee. Berry and Goldberg tested several graph partitioning heuristics against each other and against the SDP, finding the heuristics consistently produce larger cuts than the SDP [3]. Dunning, Gupta, and Silberholz performed a systematic review of Max Cut heuristics and computationally tested 19 of them [8]. Their results demonstrate the strength of a heuristic developed by Burer, Monteiro, and Zhang (BMZ) based on a rank-two relaxation of the Goemans–Williamson SDP [5]. In the initial presentation by BMZ, they also show their heuristic to outperform the SDP; we include a comparison of the BMZ heuristic in this work. Hassin and Leshenko also used the library of instances developed by Dunning et al. to compare their greedy heuristic to several others [12].

As far as we are aware there are no previously published results comparing Trevisan's spectral algorithm. We seek to fill this gap due to the importance of Trevisan's algorithm; it is the only known algorithm for Max Cut other than the SDP which guarantees an approximation better than .5, which can be achieved by simple algorithms. Additionally, the algorithm does not rely on the strength of any semidefinite or linear programming solvers or relaxations, and there is no proof that the analysis of the approximation guarantee is tight. Theoretically, the computational speed of Trevisan's algorithm seems to be far faster than the SDP, and if the quality of the cuts is comparable to the SDP, there is justification for using spectral methods over the SDP in heuristics.

In this article, we evaluate the performances of the SDP, spectral, and greedy algorithms on a variety of graphs. We also compare these performances against the BMZ heuristic and optimal values or upper bounds computed by the Biq Mac solver developed by Rendl, Rinaldi, and Wiegele [22]. Section 2 provides more complete descriptions of the five algorithms and BMZ heuristic considered. Section 3 describes the experiments and presents the results of the algorithms on different classes of graphs. Finally, Section 4 concludes with a summary of the performances and introduces a few possible directions for future theoretical study.

2 ALGORITHMS

This section describes the local search procedure, five algorithms, and BMZ heuristic that we implemented for Max Cut. We first describe the local search procedure utilized in all experiments in Section 2.1. Section 2.2 describes the benchmark greedy .5-approximation algorithm for Max Cut. Section 2.3 describes Trevisan's spectral algorithm for Max Cut, while Section 2.4 describes two simplifications of this algorithm. Finally, Section 2.5 describes the SDP algorithm and Section 2.6 describes the BMZ heuristic based on a rank-two relaxation of the SDP algorithm.

2.1 Repeated 1-Flip Local Search

It is common practice in the heuristics community to use a local search procedure to supplement algorithm implementations. We implemented a repeated 1-flip local search which was applied after each procedure before returning a cut; the local search works as follows. Given a cut (S, T), the algorithm repeatedly checks to see whether there is a vertex v such that the value of the cut can be improved by moving v from S to T or vice versa. If more than one such vertex exists, the algorithm selects to move the one that maximizes the improvement to the value of the cut and repeats the process. When no such vertices remain, the procedure terminates and the cut is returned. See Algorithm 1 for pseudocode.

2.2 Greedy Algorithm

The first Max Cut algorithm we consider is the standard greedy algorithm. This fast and simple algorithm provides a benchmark for the speed and cut value of the others. For a graph $G = (V, E)$, we return a cut (L, R) by greedily assigning vertices to either L or R one at a time. We start with $L, R = \emptyset$. In each step of the algorithm, we choose a vertex $v \in V$ yet to be assigned to L or R. Then

ALGORITHM 1: Local Search

```
input  : G = (V, E), A = (a_ij) where a_ij = w_(i,j), cut (L, R)
output : Cut (L, R)
updates ← n-dimensional array of 0s;
for i = 1, ..., n do
    if i ∈ L then
        updates[i] = Σ_{j∈L:(i,j)∈E} w_(i,j) − Σ_{j∈R:(i,j)∈E} w_(i,j);
    else
        updates[i] = Σ_{j∈R:(i,j)∈E} w_(i,j) − Σ_{j∈L:(i,j)∈E} w_(i,j);
    end
end
while max(updates) > 0 do
    v' = argmax(updates);
    if v' ∈ L then
        for j such that (v', j) ∈ E do
            if j ∈ L then
                updates[j] = updates[j] − 2w_(v',j);
            else
                updates[j] = updates[j] + 2w_(v',j);
            end
        end
        move v' to R;
    else
        for j such that (v', j) ∈ E do
            if j ∈ R then
                updates[j] = updates[j] − 2w_(v',j);
            else
                updates[j] = updates[j] + 2w_(v',j);
            end
        end
        move v' to L;
    end
    updates[v'] = −updates[v'];
end
return (L, R)
```

we add v to L or R by choosing the larger of the two cuts $(L \cup \{v\}, R)$ and $(L, R \cup \{v\})$ at this step. See Algorithm 2 for pseudocode.

2.3 Trevisan's Algorithm

The next algorithm for finding a large cut in a graph is Trevisan's spectral algorithm. Trevisan proved a .531 approximation ratio for the algorithm, while Soto improved the analysis to .614. Here, we describe Soto's presentation of the algorithm. Given a graph $G = (V, E)$ with $|V| = n$, the adjacency matrix $A = (a_{ij})$ is given by $a_{ij} = w_{ij}$ if $(i, j) \in E$ and 0, otherwise. Then the normalized adjacency matrix $\mathcal{A}$ is given by $\mathcal{A} = D^{-1/2}AD^{-1/2}$ where $D = diag(d)$ for $d(i) = \sum_{j:(i,j)\in E} w_{(i,j)}$ the weighted degree of vertex i. In our implementation of the algorithm, we compute the eigenvector, x, corresponding to the minimum eigenvalue of $I + \mathcal{A}$. After normalizing x so that $\max_i |x_i| = 1$,

ALGORITHM 2: Greedy

input : $G = (V, E), A = (a_{ij})$ where $a_{ij} = w_{(i,j)}$
output: Cut (L, R)
$L \leftarrow \emptyset$;
$R \leftarrow \emptyset$;
while $V \setminus (L \cup R) \neq \emptyset$ **do**
  select $v \in V \setminus (L \cup R)$;
  $L\_weight = \sum_{(v,j) \in E : j \in L} w_{(v,j)}$;
  $R\_weight = \sum_{(v,j) \in E : j \in R} w_{(v,j)}$;
  if $L\_weight > R\_weight$ **then**
    Add v to R;
  else
    Add v to L;
  end
end
return (L, R)

a number t^2 is drawn uniformly at random from [0,1]. We let

$$L = \{v : x_v \leq -t\},$$
$$R = \{v : x_v \geq t\}, \text{ and}$$
$$V' = V \setminus (L \cup R).$$

Now (L, R) represents a partial cut of the vertices with V' being the vertices yet to be partitioned.

Given L, R, and V', we compute

$$C = \text{total weight of the edges between } L \text{ and } R,$$
$$X = \text{total weight of the edges between } L \cup R \text{ and } V', \text{ and}$$
$$M = \text{total weight of all edges} - \text{total weight of the edges between vertices of } V'.$$

If $C + X/2 - M/2 < 0$, we use the greedy algorithm to partition the vertices instead of t as the expected value of the cut is worse than that of greedy. If $C + X/2 - M/2 > 0$, we keep the partial cut and recurse to find a cut of the vertices in V' given by (L', R'). Finally, we return the larger of the cuts $(L \cup L', R \cup R')$ and $(L \cup R', R \cup L')$. See Algorithm 3 for pseudocode.

Since the results of Trevisan's algorithm are highly reliant on the t^2 value chosen, one could ask if there are ways to modify the algorithm to increase the likelihood of choosing a good t^2 value. We tested two methods. The first chooses 5 t^2 values at each stage of the algorithm. The idea here was that choosing many random numbers should increase the probability of choosing a "good" random number. The major roadblock is deciding which partial cut corresponds to one of the t^2 values the algorithm should recurse on. We tested the greedy choice. More specifically, C, X, and M were computed for each drawn t^2 value as above, and we kept the partial cut maximizing $C + X/2 - M/2$. We made this selection because it represents the partial cut that is currently performing better than the greedy algorithm by the largest margin. In particular, given a partial cut due to a partial assignment of vertices, we can consider three types of edges: edges with both endpoints assigned, edges with exactly one endpoint assigned, and edges with neither endpoint assigned. The third type is not affected by the partial cut and is not considered in this iteration. However, C computes the value of the cut in Trevisan's algorithm due to the first type of edge. $X/2$ is the expected value added to this cut from the edges of the second type if the remaining vertices are greedily assigned, and $M/2$ is the expected value of the greedy cut due to edges of both of the

ALGORITHM 3: Trevisan

input : $G = (V, E)$ with $|V| = n, A = (a_{ij})$ where $a_{ij} = w_{(i,j)}$
output: Cut (L, R)
$D = diag(d)$ where $d(i) = \sum_{j:(i,j)\in E} w_{(i,j)}$;
$\mathcal{A} = D^{-1/2}AD^{-1/2}$;
$x \leftarrow$ minimum eigenvector of $I + \mathcal{A}$;
$x_i \leftarrow \frac{x_i}{\max_{j=1,\dots n} |x_j|}$ for $i = 1, \dots n$;
$t^2 \sim U(0, 1)$;
$L \leftarrow \{v \in V : x_v \leq -t\}$;
$R \leftarrow \{v \in V : x_v \geq t\}$;
$V' = V \setminus (L \cup R)$;
$C \leftarrow \sum_{(i,j)\in(L,R)} w_{(i,j)}$;
$X \leftarrow \sum_{(i,j)\in(L\cup R,V')} w_{(i,j)}$;
$M \leftarrow \sum_{(i,j)\in E} w_{(i,j)} - \sum_{(i,j)\in E:i,j\in V'} w_{(i,j)}$;
if $C + X/2 + M/2 < 0$ **then**
  return Greedy(V,E,A)
else
  $(L', R') \leftarrow Trevisan(V', E, A)$;
  return the better cut between $(L \cup L', R \cup R')$ and $(L \cup R', R \cup L')$;
end

first two types. Therefore, if $C + X/2 > M/2$, the partial cut being considered is performing better than the greedy algorithm would be in expectation. The greater the difference between $C + X/2$ and $M/2$, theoretically the better Trevisan's algorithm is performing compared to greedy. It is not obvious that this is the best heuristic, but it does allow the algorithm to test several random values quickly.

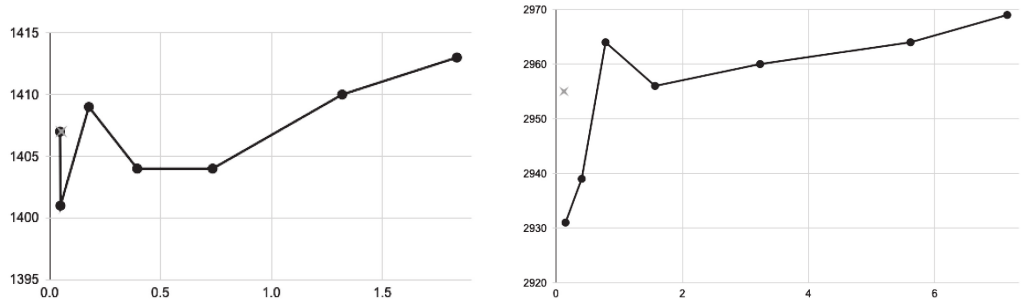
Alternatively, we also experimented with running Trevisan's algorithm for several iterations and maintaining the best cut that was found. The advantage here as opposed to the previous modification is we do not have to determine which t value to keep. However, the runtime is slower because the entire algorithm is run several times instead of adding additional quick random draws.

The results of these modifications for two of the tested graphs are provided in Figure 1. In each figure, the line represents the results from trials of running the algorithm multiple times (1, 2, 5, 10, 20, 35, and 50 times). Note that the line is not monotonically increasing. This is because each group of runs was unique and not a cumulative total. For example, when considering how well Trevisan's algorithm performs when running 10 iterations, we run 10 new iterations and do not build off of the 5 from the previous data point. The single dot represents the average runtime and cut value of the best result when implementing the first modification of multiple t^2 values.

The experiments were run on a variety of graphs and Figure 1 is a representative sample. Selecting multiple t^2 values seemed to perform well for the sparser Network Repository graphs, but overall, it seems running the algorithm multiple times is more effective in increasing cut quality than choosing multiple t^2 values. However, the number of iterations needed is not obvious, though it appears at least five are beneficial. Due to this observation, we use this method of running Trevisan's algorithm five times and keeping the best cut for the experiments presented in this article.

2.4 Simple Spectral and Sweep Cuts Algorithms

The simple spectral algorithm is a modification of Trevisan's algorithm described in the previous section. Instead of drawing a random number t in $[0, 1]$, we return the cut corresponding to $t = 0$.



(a) G(100,.5) (b) G(200,.25)

Fig. 1. Plots depicting the effects on runtime and returned cut quality of running Trevisan's algorithm multiple times on the G(100,.5) and G(200,.25). The X and Y axes are the time in seconds and the cut value, respectively. The light gray "x" is presented for comparison and is the result of running Trevisan's algorithm once, but testing many random values.

ALGORITHM 4: Simple Spectral

input : $G = (V, E)$ with $|V| = n, A = (a_{ij})$ where $a_{ij} = w_{(i,j)}$
output: Cut (L, R)
$D = diag(d)$ where $d(i) = \sum_{j:(i,j)\in E} w_{(i,j)}$;
$\mathcal{A} = D^{-1/2}AD^{-1/2}$;
$x \leftarrow$ minimum eigenvector of $I + \mathcal{A}$;
$L = \{v \in V : x_v < 0\}$;
$R = V \setminus L$;
return (L, R);

In particular, let x be the eigenvector corresponding to the smallest eigenvalue as before. Since scaling numbers by a positive factor does not change their sign, we may skip the normalizing step for x. We let $L = \{v : x_v < 0\}$ and $R = V \setminus L$ and return the cut (L, R). This modified simple spectral algorithm has no known approximation guarantee. See Algorithm 4 for pseudocode.

The sweep cuts algorithm works in a similar fashion. Here, we consider $n - 1$ different cuts and return the best. Given the smallest eigenvector x, we sort the entries so that $x_{i_1} \le x_{i_2} \le \cdots \le x_{i_n}$. Then we calculate the sweep cut value for $L_j = \{i_1, \ldots, i_j\}$ and $R_j = V \setminus L_j$ for $j = 1, \ldots, n-1$. The sweep cuts algorithm returns the cut (L_j, R_j) of maximal value. See Algorithm 5 for pseudocode.

It is worth noting that the sweep cuts algorithm will always perform at least as well as the simple spectral algorithm in terms of cut value since one of the sweep cuts will be the same as the $t = 0$ cut. However, it is interesting to see how much better the sweep cuts algorithm performs since it is also guaranteed to have a slower runtime for the same reasons.

2.5 SDP Algorithm

Goemans and Williamson introduced a .878 approximation algorithm for Max Cut based off of the following model:

$$MaxCut(G) = \max_{x_i \in \{1,-1\}} \frac{1}{2} \sum_{i<j} w_{ij}(1 - x_i x_j),$$

where again w_{ij} is the weight of edge (i, j). Note that this models the problem of finding a Max Cut. Each cut (S, T) corresponds to an assignment of 1 or -1 for each x_i. In particular, let $x_i = 1$

ALGORITHM 5: Sweep Cuts

input : $G = (V, E)$ with $|V| = n, A = (a_{ij})$ where $a_{ij} = w_{(i,j)}$
output: Cut (L, R)
$D = diag(d)$ where $d(i) = \sum_{j:(i,j)\in E} w_{(i,j)}$;
$\mathcal{A} = D^{-1/2}AD^{-1/2}$;
$x \leftarrow$ minimum eigenvector of $I + \mathcal{A}$;
sort x so that $x_{i_1} \leq x_{i_2} \leq \cdots \leq x_{i_n}$;
$bestL \leftarrow \emptyset$;
$bestR \leftarrow \emptyset$;
for $j = 1, \ldots n$ **do**
  $L = \{i_1, \ldots, i_j\}$;
  $R = V \setminus L$;
  if *the value of* (L, R) > *the value of* $(bestL, bestR)$ **then**
    $bestL \leftarrow L$;
    $bestR \leftarrow R$;
  end
end
return $(bestL, bestR)$;

ALGORITHM 6: SDP

input : $G = (V, E)$ with $|V| = n, A = (a_{ij})$ where $a_{ij} = w_{(i,j)}$
output: Cut (L, R)
$X = (x_1 \ldots x_n) \leftarrow \text{argmax}_{||x_i||=1} \frac{1}{2} \sum_{i<j} a_{ij}(1 - x_i \cdot x_j)$;
$r \leftarrow$ a vector uniformly distributed on the unit sphere in $\mathbb{R}^n$;
$L = \{i : \langle r, x_i \rangle \leq 0\}$;
$R = V \setminus L$;
return (L, R);

if $i \in S$ and $x_i = -1$ if $i \in T$, then $\frac{1}{2} \sum_{i<j} w_{ij}(1 - x_i x_j)$ exactly computes the value of the cut (S, T). Thus, $MaxCut(G)$ is solving for the cut with maximum value. Since solving $MaxCut(G)$ directly is NP-complete, Goemans and Williamson instead relax $MaxCut(G)$ to a model solvable by a semidefinite program. In particular, instead of requiring $x_i \in \{1, -1\}$, they require $v_i \in \mathbb{R}^n$ to be unit vectors and replace $x_i x_j$ with $\langle v_i, v_j \rangle$. Given a solution to this SDP relaxation, they draw a random vector $r \in \mathbb{R}^n$ uniformly from the unit sphere and partition the vertices according to

$$L = \{i : \langle r, v_i \rangle \leq 0\} \text{ and}$$
$$R = \{i : \langle r, v_i \rangle > 0\}.$$

See Algorithm 6 for the pseudocode.

This gives the .878 approximation in expectation. Since drawing random vectors is computationally cheap, we opted to draw multiple random vectors instead of just 1. We ran a few tests to determine how many we should select for each instance and whether local search should be run for each cut corresponding to a random vector or just the best. First, we compared selecting 100 random vectors and running local search on the single best to selecting 100 random vectors and running local search on each one. The results indicated that better cuts were almost always achieved by running local search on each random vector and returning the best of those cuts, but it increased the overall runtime of tested instances by about 10%. Since the SDP is already the slowest method, we wanted to limit the runtime increase. Instead, we ran the SDP and selected 100 random

vectors with local search on the single best and recorded the overall runtime. We then used this runtime as a budget to run the SDP and draw as many random vectors with local search after each one as possible. Overall, this kept the runtime consistent but also demonstrated some interesting behavior. In particular, in some cases, due to the randomness of the SDP solve time, the budget was not enough to even draw one random vector, and, in other cases, a pattern emerged showing that not very many random vectors were needed to beat the value of the current benchmark solution, if local search was run after each one. As a result, for our final implementation providing the results presented in this work, we drew 10 random vectors and ran local search after each one. In nearly all cases, this resulted in a cut with value at least as large as and a runtime at worst as slow as those produced by drawing 100 random vectors and only locally searching the best one.

2.6 BMZ Heuristic

BMZ observe that adding an additional constraint, that the solution lies in $\mathbb{R}^2$, to the Goemans–Williamson SDP allows the optimization to be represented as a nonlinear optimization problem with n variables and a nonconvex objective function. They utilize this formulation to develop a powerful Max Cut heuristic.

In particular, BMZ begin with the Goemans–Williamson SDP, but require that $v_i \in \mathbb{R}^2$ instead of $\mathbb{R}^n$. These vectors in $\mathbb{R}^2$ can then be represented as a single vector $\theta \in \mathbb{R}^n$ where θ_i is the angle such that $v_i = (\cos\theta_i, \sin\theta_i)$ for $i = 1, \ldots n$. With this translation, $\langle v_i, v_j \rangle$ can be replaced with $\cos(\theta_i - \theta_j)$, and the relaxed optimization objective function becomes

$$\min_{\theta \in \mathbb{R}^n} f(\theta)$$

where

$$f(\theta) = \frac{1}{2} \sum_{i<j} w_{ij} \cos(\theta_i - \theta_j).$$

This relaxation is an unconstrained optimization problem with a nonconvex objective function and creates the baseline of the BMZ heuristic. Because $f(\theta)$ is a nonconvex function, we cannot optimally minimize it in practice and, therefore, cannot expect a guarantee on the quality of the solution. However, this heuristic has performed well in previous experiments, and the low number of variables indicates a potential for scalability of the technique, an area where the Goemans–Williamson SDP struggles in particular.

Before presenting the heuristic, note that for a given θ, we may assume $\theta_i \in [0, 2\pi)$ for $i = 1, \ldots, n$, and any angle $\alpha \in [0, \pi)$ induces a cut $(S_\alpha, V \setminus S_\alpha)$ where $S_\alpha = \{i : \theta_i \in [\alpha, \alpha + \pi)\}$. Furthermore, it is easy to compute all cuts produced in this manner: try $\alpha = \theta_i$ for every $\theta_i < \pi$ and $\alpha = \theta_i - \pi$ for every $\theta_i > \pi$.

The heuristic runs as follows. Fix a constant N, and begin with $k = 0$ and $\theta_0 \in \mathbb{R}^n$ randomly selected. Obtain a local minimum θ' to $f(\theta)$ beginning the search at θ_0 and using a gradient algorithm with a back-tracking Armijo line-search. Find the best possible associated cut $(S_{\alpha'}, V \setminus S_{\alpha'})$ through the procedure previously described. If this is the best cut seen so far, reset $k = 0$. Otherwise, increase k by 1 and set θ_0 to be the vector with $\theta_0(i) = \pi$ if $i \in S_{\alpha'}$ and 0, otherwise. Finally, apply a slight random perturbation to θ_0 and repeat the optimization until $k > N$, then return the best cut found. See Algorithm 7 for pseudocode.

There are a few implementation choices to consider in regards to the time versus cut quality tradeoff of this heuristic. First is the choice of N, the number of consecutive non-improving perturbations allowed. A larger N allows the heuristic more opportunities to improve the cut, but requires more compute time. Additionally, this heuristic is faster than the SDP (especially if $N = 0, 1$), so similarly to Trevisan's algorithm, one might decide to run the entire heuristic $M > 0$ times and

ALGORITHM 7: BMZ

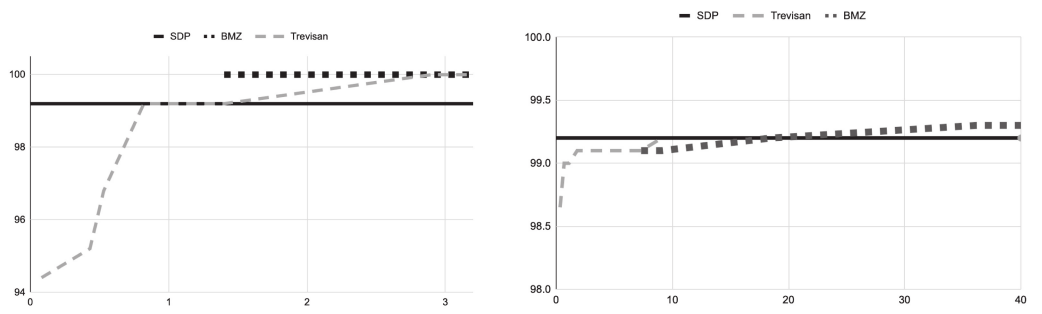
input : $G = (V, E)$ with $|V| = n$, $A = (a_{ij})$ where $a_{ij} = w_{(i,j)}$
output: Cut (L, R)
$k \leftarrow 0$;
$\theta \leftarrow$ random vector in $\mathbb{R}^n$;
$bestL, bestR \leftarrow \emptyset$;
while $k \leq N$ **do**
  $\theta' \leftarrow$ local minimum of $\frac{1}{2}\sum_{i<j} w_{(i,j)} \cos(\theta_i - \theta_j)$ found via a back-tracking Armijo line-search beginning at θ;
  $L', R' \leftarrow \emptyset$;
  for $i = 1, \ldots, n$ **do**
    if $\theta'[i] \leq \pi$ **then**
      $L = \{j : \theta'[j] \in [\theta'[i], \theta'[i] + \pi)\}$;
      $R = V \setminus L$;
    else
      $L = \{j : \theta'[j] \in [\theta'[i] - \pi, \theta'[i])\}$;
      $R = V \setminus L$;
    end
    if *the value of* (L, R) $>$ *the value of* (L', R') **then**
      $L' \leftarrow L$ $R' \leftarrow R$;
    end
  end
  if *the value of* (L', R') $>$ *the value of* $(best\_L, best\_R)$ **then**
    $bestL \leftarrow L$;
    $bestR \leftarrow R$;
    $k = 0$;
  else
    $k \leftarrow k + 1$;
  end
  $\theta \leftarrow$ a random perturbation of the cut vector for (L, R);
end
return $(bestL, bestR)$

select the best cut found in all runs. This can be beneficial since the θ_0 initialization is random. After testing several combinations of N and M, we ran our experiments of the BMZ heuristic with $M = 1$ and $N = 10$.

We also implemented and tested windmill cuts—an alternative way to produce a cut from a rank-two solution to the Goemans–Williamson SDP, or in this case a local minimum from the BMZ heuristic. For an integer $k > 0$, $Windmill_k$ is the cut given by $(S, V \setminus S)$ where $S = \{v \in V : \theta_v \in [2\ell\frac{\pi}{k}, (2\ell + 1)\frac{\pi}{k}), 0 \leq \ell < k\}$. If an optimal solution to the Goemans–Williamson SDP lies in two dimensions, then Avidor and Zwick [1] have shown that $Windmill_k$ improves the .878 approximation ratio. We tested the cut quality of both $Windmill_4$ and the exhaustive procedure previously described for the BMZ heuristic. There did not appear to be a significant difference, and in fact, the exhaustive procedure more often returned better cuts on our tested graphs. For this reason, we ran our experiments with the original cut procedure.

3 EXPERIMENTS

All algorithms were implemented in Julia and can be found at https://github.com/rmirka/max-cut-experiments. They were run on a machine with an Apple M2 chip and 16 GB of memory. The SDP



(a) ENZYMES8 (b) G(350,.75)

Fig. 2. Plots depicting the effect of a longer budgeted runtime on the found cut quality of Trevisan's algorithm and the BMZ heuristic on the ENZYMES8 and G(350,.75) random graphs. The X and Y axes are the percentage of the time budget used and the percentage of the known optimal or upper bound cut value achieved, respectively.

algorithm was computed with the JuMP modeling language for Julia and the SCS optimizer as the SDP solver. The LinearAlgebra package was used for the eigenvector computations of the spectral algorithms, and line search was implemented based on the MQLib C++ version (https://github.com/MQLib/MQLib/blob/master/src/heuristics/maxcut/burer2002.cpp) for the unconstrained nonlinear optimization of the BMZ heuristic. All graphs were also submitted to the Biq Mac solver (https://biqmac.aau.at) to compute optimal values. This solver has a three-hour time limit; for graphs where the optimal cut was not found in this time, the best cut found and the upper bound are returned instead. In these cases, the upper bound values are reported in the tables, instead of the optimal, and denoted with a *. The tables also indicate whether any of the six tested algorithms returned an optimal cut. Additionally, the fastest compute times and largest cut values found for each graph are indicated with bold values in the tables.

We measured the algorithms' performance with three types of test data. We used 20 Erdős-Renyi random graphs with 50–500 vertices, 16 complete graphs from TSPLIB [21] with 29–280 vertices (average 124), and 16 sparser graphs from the Network Repository [23] with 39–1,133 vertices (average 344.5). As previously mentioned, local search was applied to the returned solution from each procedure. Trevisan's algorithm, the SDP, and the BMZ heuristic all use randomization, so the values reported for these procedures are averages over 3 runs each for every individual graph. In addition, the times for these randomized algorithms can vary significantly across runs due to the randomization. If one of these three procedures found an optimal cut in at least one run but not all, the value is underlined in the table. In addition, the SDP algorithm tested 10 random vectors, Trevisan's algorithm was run 5 times, and the BMZ heuristic was run once but allowed 10 nonimproving steps for each run. The simple spectral and sweep cuts algorithms are deterministic, so no further implementation choices were required. We considered ways to streamline the presentation of the results by allowing each procedure the same time budget determined by the length of time required for the SDP algorithm to run on an instance. This could potentially benefit Trevisan's algorithm and the BMZ heuristic since both rely on randomness, but greedy, sweep cuts, and simple spectral would be unaffected by the extra time allowance. Ultimately, Trevisan's algorithm and the BMZ heuristic showed little to no benefit from the extra time, largely due to their ability to outperform the SDP and find the optimal or a near-optimal solution in a much smaller amount of time. Figure 2(a) and (b) provides some illustration of the behavior.

Table 1. The Time in Seconds Each Algorithm took to Compute a Cut of an Erdős–Renyi Random Graph

Graph	Greedy	Trevisan	Simple Spectral	Sweep Cuts	SDP	BMZ
G(50, 0.1)	$\mathbf{4.692 \times 10^{-4}}$	6.318×10^{-2}	9.575×10^{-2}	3.304×10^{-2}	8.485×10^{-1}	1.553×10^{-1}
G(50, 0.25)	$\mathbf{8.511 \times 10^{-4}}$	3.608×10^{-2}	1.037×10^{-3}	5.121×10^{-3}	7.527×10^{-1}	1.700×10^{-1}
G(50, 0.5)	1.597×10^{-3}	3.605×10^{-2}	$\mathbf{9.414 \times 10^{-4}}$	7.364×10^{-3}	1.011	1.510×10^{-1}
G(50, 0.75)	2.164×10^{-3}	3.562×10^{-2}	$\mathbf{1.415 \times 10^{-3}}$	1.097×10^{-2}	1.269	1.496×10^{-1}
G(100, 0.1)	$\mathbf{2.455 \times 10^{-3}}$	9.961×10^{-2}	3.530×10^{-3}	1.938×10^{-2}	4.173	8.627×10^{-1}
G(100, 0.25)	1.246×10^{-2}	1.159×10^{-1}	$\mathbf{3.380 \times 10^{-3}}$	3.469×10^{-2}	5.023	8.122×10^{-1}
G(100, 0.5)	1.172×10^{-2}	1.524×10^{-1}	$\mathbf{3.988 \times 10^{-3}}$	5.726×10^{-2}	4.845	1.193
G(100, 0.75)	1.732×10^{-2}	1.987×10^{-1}	$\mathbf{4.246 \times 10^{-3}}$	8.832×10^{-2}	7.067	1.312
G(200, 0.1)	1.822×10^{-2}	3.303×10^{-1}	$\mathbf{1.431 \times 10^{-2}}$	1.666×10^{-1}	2.330×10^{1}	6.292
G(200, 0.25)	4.492×10^{-2}	7.392×10^{-1}	$\mathbf{1.308 \times 10^{-2}}$	3.158×10^{-1}	3.768×10^{1}	3.783
G(200, 0.5)	8.742×10^{-2}	1.188	$\mathbf{1.404 \times 10^{-2}}$	5.555×10^{-1}	3.905×10^{1}	1.037×10^{1}
G(200, 0.75)	1.228×10^{-1}	1.449	$\mathbf{1.496 \times 10^{-2}}$	7.773×10^{-1}	7.843×10^{1}	1.284×10^{1}
G(350, 0.1)	9.105×10^{-2}	1.694	$\mathbf{3.216 \times 10^{-2}}$	9.708×10^{-1}	1.083×10^{2}	2.827×10^{1}
G(350, 0.25)	2.336×10^{-1}	3.031	$\mathbf{3.273 \times 10^{-2}}$	1.998	1.574×10^{2}	2.092×10^{1}
G(350, 0.5)	4.550×10^{-1}	4.833	$\mathbf{3.716 \times 10^{-2}}$	3.652	1.921×10^{2}	4.273×10^{1}
G(350, 0.75)	6.599×10^{-1}	5.959	$\mathbf{4.533 \times 10^{-2}}$	5.227	4.975×10^{2}	9.232×10^{1}
G(500, 0.1)	2.692×10^{-1}	4.121	$\mathbf{7.329 \times 10^{-2}}$	3.200	3.027×10^{2}	5.290×10^{1}
G(500, 0.25)	6.734×10^{-1}	8.013	$\mathbf{7.483 \times 10^{-2}}$	6.742	4.818×10^{2}	1.106×10^{2}
G(500, 0.5)	1.342	1.325×10^{1}	$\mathbf{9.525 \times 10^{-2}}$	1.270×10^{-1}	6.469×10^{2}	1.504×10^{2}
G(500, 0.75)	1.906	1.858×10^{1}	$\mathbf{1.005 \times 10^{-1}}$	1.836×10^{-1}	7.501×10^{2}	2.530×10^{2}

3.1 Erdős–Renyi Random Graphs

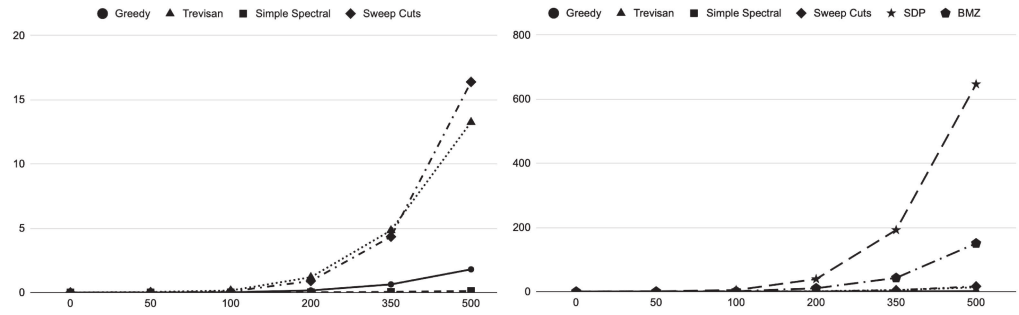
The first class of graphs tested was Erdős–Renyi random graphs. An Erdős–Renyi random graph $G(n, p)$ is a graph on n vertices where each possible edge is included independently with probability p. We tested random graphs with $n = 50, 100, 200, 350, 500$ and $p = .1, .25, .5, .75$. In our model, each included edge was given an edge weight of 1.

In terms of speed, the simple spectral algorithm significantly outperforms the other algorithms on all but three tested random graphs (where greedy was faster). On the other end of the spectrum, the SDP is far slower than the alternative algorithms. In general, the BMZ heuristic has computation times faster than SDP but about one magnitude slower than Trevisan's algorithm. The time statistics are presented in Table 1. The plots in Figure 3(a) and (b) illustrates how the computation times of each algorithm grow as the number of vertices increases. For these plots, we use the data from Table 1 with $p = .5$ fixed.

The BMZ heuristic by far performs the best in terms of the returned cut quality for random graphs. On average, it finds the best cut for 14 of the 20 graphs. Additionally, in at least one run, it finds an optimal cut for seven of the eight random graphs for which the optimal cut value is known. The only algorithm to find better cuts than the BMZ heuristic for multiple random graphs was sweep cuts, which found the best cut for five of the remaining graphs. Greedy finds the best cut on one graph. These results are provided in Table 2.

3.2 Complete Graphs

The algorithms were also tested on 16 complete graphs from TSPLIB, an online library of sample instances for the Traveling Salesman Problem and related graph problems. Each graph has varying positive edge weights between 1 and 15,000. The performance in regards to time largely mirrors that of the random graphs. The simple spectral algorithm is significantly faster than the rest of the algorithms on the vast majority of graphs, with the greedy algorithm in a close second. Sweep



(a) Spectral and greedy algorithms.

(b) All tested algorithms.

Fig. 3. Plots depicting the effects on runtime of increasing the number of vertices of an Erdős–Renyi graph with $p = .5$. The X and Y axes are the number of vertices and the computation time in seconds, respectively.

Table 2. The Value of the Cut Each Algorithm Returned for an Erdős–Renyi Random Graph

Graph	Greedy	Trevisan	Simple Spectral	Sweep Cuts	SDP	BMZ	OPT	Achieved
G(50, 0.1)	1.070×10^2	1.150×10^2	$\mathbf{1.160 \times 10^2}$	$\mathbf{1.160 \times 10^2}$	1.143×10^2	$\mathbf{1.160 \times 10^2}$	1.160×10^2	X
G(50, 0.25)	2.030×10^2	2.093×10^2	2.100×10^2	$\mathbf{2.110 \times 10^2}$	$\mathbf{2.110 \times 10^2}$	$\mathbf{2.110 \times 10^2}$	2.110×10^2	X
G(50, 0.5)	3.750×10^2	3.753×10^2	3.730×10^2	$\mathbf{3.770 \times 10^2}$	$\mathbf{3.770 \times 10^2}$	$\mathbf{3.770 \times 10^2}$	3.770×10^2	X
G(50, 0.75)	$\mathbf{5.240 \times 10^2}$	$\mathbf{5.240 \times 10^2}$	$\mathbf{5.240 \times 10^2}$	$\mathbf{5.240 \times 10^2}$	5.237×10^2	$\mathbf{5.240 \times 10^2}$	5.240×10^2	X
G(100, 0.1)	2.860×10^2	3.030×10^2	3.020×10^2	3.090×10^2	3.087×10^2	$\mathbf{3.107 \times 10^2}$	3.110×10^2	X
G(100, 0.25)	7.630×10^2	7.743×10^2	7.740×10^2	7.800×10^2	7.760×10^2	$\mathbf{7.807 \times 10^2}$	7.820×10^2	X
G(100, 0.5)	1.385×10^3	1.405×10^3	1.403×10^3	1.412×10^3	1.407×10^3	$\mathbf{1.412 \times 10^3}$	1.416×10^3	
G(100, 0.75)	2.018×10^3	2.025×10^3	2.033×10^3	2.033×10^3	2.033×10^3	$\mathbf{2.034 \times 10^3}$	2.035×10^3	X
G(200, 0.1)	1.243×10^3	1.283×10^3	1.289×10^3	$\mathbf{1.296 \times 10^3}$	1.285×10^3	$\mathbf{1.296 \times 10^3}$	$1.310 \times 10^{3*}$	
G(200, 0.25)	2.912×10^3	2.958×10^3	2.969×10^3	$\mathbf{2.970 \times 10^3}$	2.962×10^3	2.970×10^3	$3.006 \times 10^{3*}$	
G(200, 0.5)	$\mathbf{5.538 \times 10^3}$	5.518×10^3	5.520×10^3	5.529×10^3	5.524×10^3	5.537×10^3	$5.594 \times 10^{3*}$	
G(200, 0.75)	7.811×10^3	7.874×10^3	7.880×10^3	$\mathbf{7.899 \times 10^3}$	7.887×10^3	7.899×10^3	$7.945 \times 10^{3*}$	
G(350, 0.1)	3.630×10^3	3.693×10^3	3.703×10^3	3.720×10^3	3.704×10^3	$\mathbf{3.728 \times 10^3}$	$3.829 \times 10^{3*}$	
G(350, 0.25)	8.557×10^3	8.646×10^3	8.663×10^3	8.690×10^3	8.668×10^3	$\mathbf{8.699 \times 10^3}$	$8.860 \times 10^{3*}$	
G(350, 0.5)	1.639×10^4	1.642×10^4	1.645×10^4	$\mathbf{1.646 \times 10^4}$	1.645×10^4	1.646×10^4	$1.665 \times 10^{4*}$	
G(350, 0.75)	2.384×10^4	2.387×10^4	2.390×10^4	$\mathbf{2.394 \times 10^4}$	2.393×10^4	2.393×10^4	$2.412 \times 10^{4*}$	
G(500, 0.1)	7.345×10^3	7.447×10^3	7.440×10^3	7.466×10^3	7.461×10^3	$\mathbf{7.499 \times 10^3}$	$7.725 \times 10^{3*}$	
G(500, 0.25)	1.720×10^4	1.731×10^4	1.732×10^4	1.734×10^4	1.734×10^4	$\mathbf{1.738 \times 10^4}$	$1.770 \times 10^{4*}$	
G(500, 0.5)	3.308×10^4	3.312×10^4	3.315×10^4	$\mathbf{3.320 \times 10^4}$	3.318×10^4	3.318×10^4	$3.361 \times 10^{4*}$	
G(500, 0.75)	4.833×10^4	4.847×10^4	4.845×10^4	4.853×10^4	4.850×10^4	$\mathbf{4.853 \times 10^4}$	$4.890 \times 10^{4*}$	

cuts, Trevisan's algorithm, and the BMZ heuristic are a magnitude or two slower, and the SDP is even slower still. This data is presented in Table 3.

This time several of the procedures perform very well. Greedy, Trevisan's algorithm, sweep cuts, and the BMZ heuristic all find best cuts for all but one or two of the complete graphs each (and hence finding equally good cuts on a majority of the graphs). Optimal cut values are known for 15 of the 16 graphs, and corresponding optimal cuts are found by multiple procedures for each of these graphs. Simple spectral and the SDP also find many good cuts, but not quite as many as the other procedures. These results are presented in Table 4.

3.3 Sparser Graphs

The third group of graphs is composed of a variety of graphs from the Network Repository, an online and interactive collection of network graph data coming from a variety of sources and applications. Though more structured than a random graph, these 16 graphs are sparser than the complete graphs tested in Section 3.2 and are chosen from a range of real-world scenarios. All

Table 3. The Time in Seconds Each Algorithm Took to Compute a Cut of a Complete Graph from TSPLIB

Graph	Greedy	Trevisan	Simple Spectral	Sweep Cuts	SDP	BMZ
bayg29	$\mathbf{6.936 \times 10^{-4}}$	8.519×10^{-2}	2.246×10^{-2}	2.808×10^{-3}	3.679×10^{-1}	7.586×10^{-2}
bays29	$\mathbf{6.615 \times 10^{-4}}$	1.165×10^{-2}	7.907×10^{-4}	2.777×10^{-3}	3.977×10^{-1}	3.426×10^{-2}
berlin52	3.051×10^{-3}	3.577×10^{-2}	$\mathbf{2.236 \times 10^{-3}}$	1.405×10^{-2}	1.254	1.579×10^{-1}
bier127	4.474×10^{-2}	3.977×10^{-1}	$\mathbf{1.811 \times 10^{-2}}$	2.223×10^{-1}	7.971	1.685
brazil58	3.772×10^{-3}	4.448×10^{-2}	$\mathbf{2.794 \times 10^{-3}}$	2.175×10^{-2}	1.911	1.149×10^{-1}
brg180	$\mathbf{1.135 \times 10^{-1}}$	1.265	1.165×10^{-1}	7.213×10^{-1}	1.850×10^{1}	4.241
ch130	4.779×10^{-2}	4.529×10^{-1}	$\mathbf{1.514 \times 10^{-2}}$	2.557×10^{-1}	9.934	1.316
ch150	6.942×10^{-2}	7.625×10^{-1}	$\mathbf{3.236 \times 10^{-2}}$	3.983×10^{-1}	1.261×10^{1}	1.985
d198	1.789×10^{-1}	1.545	$\mathbf{5.201 \times 10^{-2}}$	1.034	3.874×10^{1}	3.786
eil101	2.121×10^{-2}	2.411×10^{-1}	$\mathbf{1.290 \times 10^{-2}}$	1.140×10^{-1}	5.521	6.425×10^{-1}
gr120	3.516×10^{-2}	3.259×10^{-1}	$\mathbf{1.377 \times 10^{-2}}$	1.965×10^{-1}	1.707×10^{1}	7.915×10^{-1}
gr137	6.499×10^{-2}	5.163×10^{-1}	$\mathbf{1.747 \times 10^{-2}}$	3.016×10^{-1}	1.660×10^{1}	1.176
gr202	1.933×10^{-1}	1.549	$\mathbf{4.802 \times 10^{-2}}$	1.065	2.990×10^{1}	5.493
gr96	2.220×10^{-2}	2.145×10^{-1}	$\mathbf{8.834 \times 10^{-3}}$	9.609×10^{-2}	6.027	4.647×10^{-1}
kroA100	2.055×10^{-2}	2.103×10^{-1}	$\mathbf{8.645 \times 10^{-3}}$	1.130×10^{-1}	3.925	4.501×10^{-1}
a280	5.086×10^{-1}	3.653	$\mathbf{9.850 \times 10^{-2}}$	3.123	1.559×10^{2}	1.202×10^{1}

Table 4. The Value of the Cut Each Algorithm Returned for a Complete Graph from TSPLIB

Graph	Greedy	Trevisan	Simple Spectral	Sweep Cuts	SDP	BMZ	OPT	achieved
bayg29	$\mathbf{4.269 \times 10^{4}}$	$\mathbf{4.269 \times 10^{4}}$	$\mathbf{4.269 \times 10^{4}}$	$\mathbf{4.269 \times 10^{4}}$	$\mathbf{4.269 \times 10^{4}}$	$\mathbf{4.269 \times 10^{4}}$	4.269×10^{4}	X
bays29	$\mathbf{5.399 \times 10^{4}}$	$\mathbf{5.399 \times 10^{4}}$	$\mathbf{5.399 \times 10^{4}}$	$\mathbf{5.399 \times 10^{4}}$	$\underline{5.397 \times 10^{4}}$	$\mathbf{5.399 \times 10^{4}}$	5.399×10^{4}	X
berlin52	$\mathbf{4.707 \times 10^{5}}$	$\mathbf{4.707 \times 10^{5}}$	4.695×10^{5}	$\mathbf{4.707 \times 10^{5}}$	$\mathbf{4.707 \times 10^{5}}$	$\mathbf{4.707 \times 10^{5}}$	4.707×10^{5}	X
bier127	$\mathbf{2.342 \times 10^{7}}$	$\mathbf{2.342 \times 10^{7}}$	2.337×10^{7}	2.340×10^{7}	2.337×10^{7}	$\mathbf{2.342 \times 10^{7}}$	2.342×10^{7}	X
brazil58	$\mathbf{2.319 \times 10^{6}}$	$\mathbf{2.319 \times 10^{6}}$	$\mathbf{2.319 \times 10^{6}}$	$\mathbf{2.319 \times 10^{6}}$	$\mathbf{2.319 \times 10^{6}}$	$\mathbf{2.319 \times 10^{6}}$	2.319×10^{6}	X
brg180	4.118×10^{7}	4.626×10^{7}	4.621×10^{7}	$\mathbf{4.634 \times 10^{7}}$	4.624×10^{7}	4.631×10^{7}	$4.648 \times 10^{7*}$	
ch130	1.885×10^{6}	$\underline{1.888 \times 10^{6}}$	$\mathbf{1.888 \times 10^{6}}$	$\mathbf{1.888 \times 10^{6}}$	1.887×10^{6}	$\mathbf{1.888 \times 10^{6}}$	1.888×10^{6}	X
ch150	$\mathbf{2.526 \times 10^{6}}$	$\mathbf{2.526 \times 10^{6}}$	$\mathbf{2.526 \times 10^{6}}$	$\mathbf{2.526 \times 10^{6}}$	$\mathbf{2.526 \times 10^{6}}$	$\mathbf{2.526 \times 10^{6}}$	2.526×10^{6}	X
d198	$\mathbf{1.294 \times 10^{7}}$	$\mathbf{1.294 \times 10^{7}}$	$\mathbf{1.294 \times 10^{7}}$	$\mathbf{1.294 \times 10^{7}}$	$\mathbf{1.294 \times 10^{7}}$	$\mathbf{1.294 \times 10^{7}}$	1.294×10^{7}	X
eil101	$\mathbf{1.071 \times 10^{5}}$	$\mathbf{1.071 \times 10^{5}}$	1.071×10^{5}	$\mathbf{1.071 \times 10^{5}}$	$\mathbf{1.071 \times 10^{5}}$	$\mathbf{1.071 \times 10^{5}}$	1.071×10^{5}	X
gr120	$\mathbf{2.157 \times 10^{6}}$	$\mathbf{2.157 \times 10^{6}}$	$\mathbf{2.157 \times 10^{6}}$	$\mathbf{2.157 \times 10^{6}}$	$\mathbf{2.157 \times 10^{6}}$	$\mathbf{2.157 \times 10^{6}}$	2.157×10^{6}	X
gr137	$\mathbf{3.070 \times 10^{7}}$	$\mathbf{3.070 \times 10^{7}}$	$\mathbf{3.070 \times 10^{7}}$	$\mathbf{3.070 \times 10^{7}}$	$\mathbf{3.070 \times 10^{7}}$	$\mathbf{3.070 \times 10^{7}}$	3.070×10^{7}	X
gr202	$\mathbf{1.600 \times 10^{7}}$	$\mathbf{1.600 \times 10^{7}}$	1.599×10^{7}	$\mathbf{1.600 \times 10^{7}}$	1.599×10^{7}	$\mathbf{1.600 \times 10^{7}}$	1.600×10^{7}	X
gr96	$\mathbf{1.166 \times 10^{7}}$	$\mathbf{1.166 \times 10^{7}}$	$\mathbf{1.166 \times 10^{7}}$	$\mathbf{1.166 \times 10^{7}}$	$\mathbf{1.166 \times 10^{7}}$	$\mathbf{1.166 \times 10^{7}}$	1.166×10^{7}	X
kroA100	$\mathbf{5.897 \times 10^{6}}$	$\mathbf{5.897 \times 10^{6}}$	$\mathbf{5.897 \times 10^{6}}$	$\mathbf{5.897 \times 10^{6}}$	$\mathbf{5.897 \times 10^{6}}$	$\mathbf{5.897 \times 10^{6}}$	5.897×10^{6}	X
a280	$\mathbf{3.210 \times 10^{6}}$	$\mathbf{3.210 \times 10^{6}}$	$\mathbf{3.210 \times 10^{6}}$	$\mathbf{3.210 \times 10^{6}}$	$\mathbf{3.210 \times 10^{6}}$	$\mathbf{3.210 \times 10^{6}}$	3.210×10^{6}	X

graphs have edge weights of 1 except for inf-USAir-97 which has positive real weights between 0 and 1. The relationships between relative computation times remain mostly unchanged with the exception of the greedy algorithm replacing simple spectral as the fastest algorithm most often (Table 5).

For this group of graphs, the BMZ heuristic is again the most dominant. Of the six tested procedures, it returns the best cut on average for all but three of these graphs, finding an optimal cut for at least five graphs. The other procedures find cuts that are competitive with those found by the BMZ heuristic, but no procedure matched the consistency of the quality of the BMZ heuristic cuts Table 6.

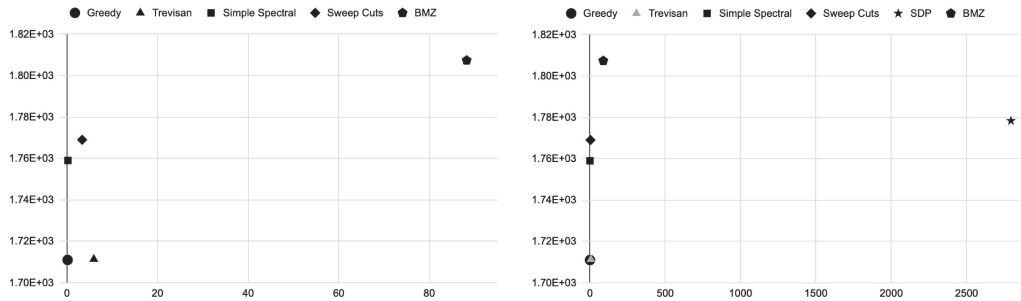
In Figure 4(a), (b) and Figure 5(a), (b), we provide a representative sample of the tradeoff between runtime and returned cut value of the algorithms using the graphs DD687, email-enron-only, and dwt_503.

Table 5. The Time in Seconds Each Algorithm Took to Compute a Cut of a Graph from the Network Repository Arising in the Real-world

Graph	$\|V\|$	$\|E\|$	Greedy	Trevisan	Simple Spectral	Sweep Cuts	SDP	BMZ
ENZYMES8	88	133	$\mathbf{8.904 \times 10^{-4}}$	9.541×10^{-2}	2.966×10^{-3}	1.007×10^{-2}	2.441×10^{1}	4.619×10^{-1}
johnson16-2-4	120	5460	2.795×10^{-2}	2.478×10^{-1}	$\mathbf{8.734 \times 10^{-3}}$	1.462×10^{-1}	4.083×10^{-1}	1.025
hamming6-2	64	1824	4.899×10^{-3}	8.449×10^{-2}	$\mathbf{2.191 \times 10^{-3}}$	2.469×10^{-2}	1.297	2.149×10^{-1}
ia-infect-hyper	113	2196	1.167×10^{-2}	2.235×10^{-1}	$\mathbf{6.717 \times 10^{-3}}$	7.248×10^{-2}	9.224	1.357
soc-dolphins	62	159	$\mathbf{6.412 \times 10^{-4}}$	2.868×10^{-2}	1.634×10^{-3}	6.459×10^{-3}	2.616	2.553×10^{-1}
email-enron-only	143	623	$\mathbf{5.029 \times 10^{-3}}$	1.473×10^{-1}	8.221×10^{-3}	5.102×10^{-2}	6.712×10^{1}	2.369
dwt_209	209	976	$\mathbf{9.041 \times 10^{-3}}$	2.570×10^{-1}	1.190×10^{-2}	1.101×10^{-1}	9.190×10^{1}	2.861
inf-USAir97	332	2126	$\mathbf{4.977 \times 10^{-2}}$	6.497	1.244×10^{-1}	5.658×10^{-1}	4.039×10^{2}	2.723×10^{1}
ca-netscience	379	914	$\mathbf{2.796 \times 10^{-2}}$	8.056×10^{-1}	4.444×10^{-2}	4.575×10^{-1}	3.582×10^{2}	2.961×10^{1}
ia-infect-dublin	410	2765	5.727×10^{-2}	1.034	$\mathbf{4.686 \times 10^{-2}}$	8.552×10^{-1}	6.253×10^{2}	3.775×10^{1}
road-chesapeake	39	170	$\mathbf{4.167 \times 10^{-4}}$	1.461×10^{-2}	6.285×10^{-4}	3.671×10^{-3}	5.055×10^{-1}	8.947×10^{-2}
Erdos991	492	1417	$\mathbf{5.026 \times 10^{-2}}$	3.256	7.632×10^{-2}	1.062	6.654×10^{2}	5.013×10^{1}
dwt_503	503	3265	8.141×10^{-2}	1.878	$\mathbf{7.045 \times 10^{-2}}$	1.408	1.215×10^{3}	1.771×10^{1}
p-hat700-1	700	60999	1.894	1.924×10^{1}	$\mathbf{1.830 \times 10^{-1}}$	2.253×10^{1}	1.286×10^{3}	2.152×10^{2}
DD687	725	2600	$\mathbf{1.116 \times 10^{-1}}$	5.923	1.858×10^{-1}	2.955	2.793×10^{3}	8.822×10^{1}
email-univ	1133	5451	$\mathbf{2.807 \times 10^{-1}}$	2.504×10^{1}	1.116×10^{-1}	5.940×10^{1}	5.940×10^{3}	5.278×10^{2}

Table 6. The Value of the Cut Each Algorithm Returned for a Graph from the Network Repository

Graph	Greedy	Trevisan	Simple Spectral	Sweep Cuts	SDP	BMZ	OPT	achieved
ENZYMES8	1.200×10^{2}	1.257×10^{2}	$\mathbf{1.260 \times 10^{2}}$	$\mathbf{1.260 \times 10^{2}}$	1.253×10^{2}	$\mathbf{1.260 \times 10^{2}}$	1.260×10^{2}	X
johnson16-2-4	$\mathbf{3.036 \times 10^{3}}$	$\mathbf{3.036 \times 10^{3}}$	$\mathbf{3.036 \times 10^{3}}$	$\mathbf{3.036 \times 10^{3}}$	$\mathbf{3.036 \times 10^{3}}$	$\mathbf{3.036 \times 10^{3}}$	3.077×10^{3}*	
hamming6-2	$\mathbf{9.920 \times 10^{2}}$	$\mathbf{9.920 \times 10^{2}}$	9.720×10^{2}	$\mathbf{9.920 \times 10^{2}}$	$\mathbf{9.920 \times 10^{2}}$	$\mathbf{9.920 \times 10^{2}}$	9.920×10^{2}	X
ia-infect-hyper	1.259×10^{3}	1.275×10^{3}	1.276×10^{3}	1.276×10^{3}	1.275×10^{3}	$\mathbf{1.278 \times 10^{3}}$	1.282×10^{3}*	
soc-dolphins	1.170×10^{2}	1.183×10^{2}	$\mathbf{1.220 \times 10^{2}}$	$\mathbf{1.220 \times 10^{2}}$	$\mathbf{1.220 \times 10^{2}}$	$\mathbf{1.220 \times 10^{2}}$	1.220×10^{2}	X
email-enron-only	4.030×10^{2}	4.080×10^{2}	4.100×10^{2}	4.160×10^{2}	4.217×10^{2}	$\mathbf{4.260 \times 10^{2}}$	4.270×10^{2}	
dwt_209	5.340×10^{2}	5.357×10^{2}	5.370×10^{2}	5.470×10^{2}	5.507×10^{2}	$\mathbf{5.570 \times 10^{2}}$	5.570×10^{2}	X
inf-USAir97	1.066×10^{2}	1.080×10^{2}	1.058×10^{2}	1.080×10^{2}	$\mathbf{1.080 \times 10^{2}}$	1.080×10^{2}	1.081×10^{2}	
ca-netscience	6.000×10^{2}	6.003×10^{2}	6.220×10^{2}	6.260×10^{2}	6.340×10^{2}	$\mathbf{6.343 \times 10^{2}}$	6.393×10^{2}*	
ia-infect-dublin	1.700×10^{3}	1.710×10^{3}	1.733×10^{3}	1.750×10^{3}	1.750×10^{3}	$\mathbf{1.767 \times 10^{3}}$	1.789×10^{3}*	
road-chesapeake	$\mathbf{1.260 \times 10^{2}}$	$\mathbf{1.260 \times 10^{2}}$	1.250×10^{2}	$\mathbf{1.260 \times 10^{2}}$	1.250×10^{2}	$\mathbf{1.260 \times 10^{2}}$	1.260×10^{2}	X
Erdos991	9.760×10^{2}	9.793×10^{2}	9.550×10^{2}	9.990×10^{2}	1.019×10^{3}	$\mathbf{1.031 \times 10^{3}}$	1.043×10^{3}*	
dwt_503	1.903×10^{3}	1.903×10^{3}	1.812×10^{3}	1.912×10^{3}	$\mathbf{1.934 \times 10^{3}}$	1.931×10^{3}	1.938×10^{3}	
p-hat700-1	3.316×10^{4}	3.333×10^{4}	3.336×10^{4}	$\mathbf{3.346 \times 10^{4}}$	3.345×10^{4}	3.344×10^{4}	3.405×10^{4}*	
DD687	1.711×10^{3}	1.711×10^{3}	1.759×10^{3}	1.769×10^{3}	1.778×10^{3}	$\mathbf{1.807 \times 10^{3}}$	1.833×10^{3}*	
email-univ	3.615×10^{3}	3.622×10^{3}	3.636×10^{3}	3.665×10^{3}	3.736×10^{3}	$\mathbf{3.765 \times 10^{3}}$	3.885×10^{3}*	



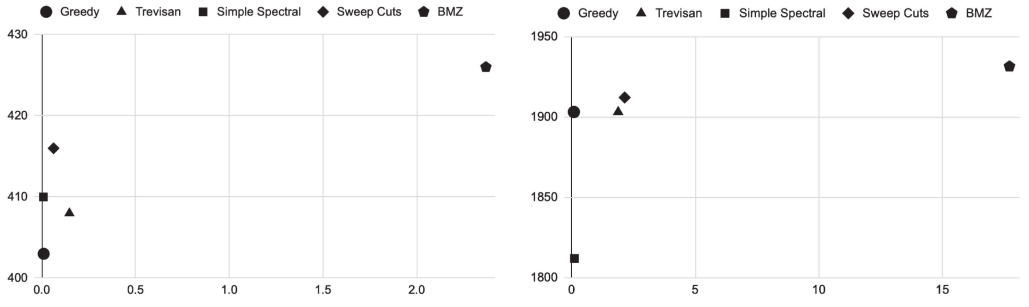
(a) All procedures except the SDP.

(b) All tested procedures.

Fig. 4. Plots depicting the computation time and returned cut values of procedures on the DD687 graph. The X and Y axes are the runtime in seconds and the returned cut value, respectively.

4 CONCLUSION

The goal of this article was to compare Max Cut algorithms with varying approximation guarantees in practice. In particular, we know the SDP has the provably best approximation guarantee;



(a) email-enron-only (b) dwt_503

Fig. 5. Plots depicting the computation time and returned cut values of the greedy, spectral, and BMZ procedures on the email-enron-only and dwt_503 graphs. The X and Y axes are the runtime in seconds and the returned cut value, respectively.

however, it is also the costliest in terms of computational space and time. This raises the question of whether or not the "cheaper" spectral Max Cut algorithms can perform competitively to the SDP in practice. Furthermore, if yes, can the approximation guarantees be improved?

We tested six procedures on three types of graphs—random, complete, and sparse real world. For the complete graphs, the experiments show that all tested procedures find strong, competitive cuts, and the main difference comes from their computational times. For this reason, we would expect the faster, spectral algorithms to be desirable algorithms for near-complete graphs. For the other two cases, random and sparse, the BMZ heuristic performs most strongly across the board. We would expect the BMZ heuristic to be an appropriate choice for these classes of graphs if the goal is to find the cut with the largest value but believe the spectral algorithms would find competitive cuts in less time if this is a main concern.

As demonstrated, the spectral and greedy algorithms provide a significant speed advantage over the SDP. Additionally, they often compute cuts better than or comparable to the cuts returned by the SDP, despite the disparity in approximation guarantees. Furthermore, the BMZ heuristic computes the highest number of best cuts and is overall competitive and consistent, even though it has no approximation guarantee, and the computation time is not quite as fast as the spectral options. The results of this experiment appear to illustrate spectral algorithms are in fact competitive with the SDP algorithm in practice and reiterate the strength of the BMZ heuristic. This suggests that the investigation into approximation guarantees for the spectral algorithms is a direction for further theoretical study. Furthermore, because the BMZ heuristic relies on finding a local minimum of a function which is not guaranteed to bound the optimal cut value, there is no known approximation guarantee for the heuristic. It would be interesting to prove a guarantee better than .5 or to show an example where the local minima correspond to cuts far from optimal.

In terms of practical implementations, for the graphs on which the SDP seems to outperform spectral algorithms, one could consider running Trevisan's algorithm for even more than five iterations and choosing the best cut returned. The magnitude of the speed advantage of Trevisan's algorithm allows for many runs before being as costly as the SDP, especially since the initial eigenvector only needs to be computed once. Additionally, finding a viable heuristic to use when choosing multiple t^2 values would also provide implementation benefits. We attempted to improve Trevisan's algorithm through drawing additional random t^2 values and greedily choosing one. However, it is not obvious that this choice in heuristic is optimal. In particular, perhaps it is more useful to draw a fixed number of t^2 values but finish the algorithm's entire partitioning instead of

estimating at that point in time. The magnitude by which the spectral algorithms are faster than the SDP allows this to be a reasonable option.

It is also worth noting the performance of the simple spectral and sweep cuts algorithms. Particularly for large graphs, these two algorithms along with the greedy algorithm are much faster than even Trevisan's algorithm, with the simple spectral almost always being several times faster than greedy (and sweep cuts being slightly slower than greedy). It is known that the greedy algorithm has a .5 approximation guarantee, but to the best of our knowledge, there is no known approximation guarantee for the simple spectral or sweep cuts algorithms. This raises the question of whether any approximation guarantee can be proven for either of these algorithms. A desired guarantee would be greater than greedy's .5; given the performance results presented here, it seems possible that this is achievable.

Relatedly, there is no indication that Soto's .614 approximation guarantee for Trevisan's algorithm is tight. It is clear that the algorithm often far surpasses this bound in practice. Can the analysis of this algorithm be improved?

ACKNOWLEDGMENTS

We thank the referees for their useful comments and feedback that improved the article.

REFERENCES

[1] Adi Avidor and Uri Zwick. 2005. Rounding two and three dimensional solutions of the SDP relaxation of MAX CUT. In *Proceedings of the Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*. Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan (Eds.), Springer, Berlin, 14–25.

[2] Francisco Barahona, Martin Grötschel, Michael Jünger, and Gerhard Reinelt. 1988. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research* 36, 3 (1988), 493–513. DOI: https://doi.org/10.1287/opre.36.3.493

[3] Jonathan W. Berry and Mark K. Goldberg. 1999. Path optimization for graph partitioning problems. *Discrete Applied Mathematics* 90, 1–3 (1999), 27–50. DOI: https://doi.org/10.1016/S0166-218X(98)00084-5

[4] Alberto Bertoni, Paola Campadelli, and Giuliano Grossi. 2001. An approximation algorithm for the maximum cut problem and its experimental analysis. *Discrete Applied Mathematics* 110, 1 (2001), 3–12. DOI: https://doi.org/10.1016/S0166-218X(00)00299-7

[5] Samuel Burer, Renato Monteiro, and Yin Zhang. 2001. Rank-two relaxation heuristics for max-cut and other binary quadratic programs. *SIAM Journal on Optimization* 12, 2 (2001), 503–521. DOI: https://doi.org/10.1137/S1052623400382467

[6] F. Della Croce, M. J. Kaminski, and V. Th. Paschos. 2007. An exact algorithm for MAX-CUT in sparse graphs. *Operations Research Letters* 35, 3 (2007), 403–408. DOI: https://doi.org/10.1016/j.orl.2006.04.001

[7] Oliver Dolezal, Thomas Hofmeister, and Hanno Lefmann. 2000. A comparison of approximation algorithms for the MaxCut-problem. (2000). DOI: https://doi.org/10.17877/DE290R-5013

[8] Iain Dunning, Swati Gupta, and John Silberholz. 2018. What works best when? A Systematic evaluation of heuristics for Max-Cut and QUBO. *INFORMS Journal on Computing* 30, 3 (2018), 608–624. DOI: https://doi.org/10.1287/ijoc.2017.0798

[9] Michel X. Goemans and David P. Williamson. 1995. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM* 42, 6 (1995), 1115–1145. DOI: https://doi.org/10.1145/227683.227684

[10] Alexander Golovnev. 2012. New upper bounds for MAX-2-SAT and MAX-2-CSP w.r.t. the average variable degree. In *Proceedings of the Parameterized and Exact Computation*. Dániel Marx and Peter Rossmanith (Eds.), Springer, Berlin, 106–117.

[11] F. Hadlock. 1975. Finding a maximum cut of a planar graph in polynomial time. *SIAM Journal on Computing* 4, 3 (1975), 221–225. DOI: https://doi.org/10.1137/0204019

[12] Refael Hassin and Nikita Leshenko. 2021. Greedy differencing edge-contraction heuristic for the max-cut problem. *Operations Research Letters* 49, 3 (2021), 320–325. DOI: https://doi.org/10.1016/j.orl.2021.02.006

[13] Timotej Hrga, Borut Lužar, Janez Povh, and Angelika Wiegele. 2021. BiqBin: Moving boundaries for NP-hard problems by HPC. In *Proceedings of the Advances in High Performance Computing*. Ivan Dimov and Stefka Fidanova (Eds.), Springer International Publishing, Cham, 327–339.

[14] Timotej Hrga and Janez Povh. 2021. MADAM: A parallel exact solver for max-cut based on semidefinite programming and ADMM. *Computational Optimization and Applications* 80, 2 (2021), 347–375. DOI: https://doi.org/10.1007/s10589-021-00310-6

[15] Richard Karp. 1972. Reducibility among combinatorial problems. In *Proceedings of the Complexity of Computer Computations*. 85–103. DOI: https://doi.org/10.1007/978-3-540-68279-0_8

[16] Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. 2007. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM Journal on Computing* 37, 1 (2007), 319–357. DOI: https://doi.org/10.1137/S0097539705447372

[17] Nathan Krislock, Jérôme Malick, and Frédéric Roupin. 2017. BiqCrunch: A semidefinite branch-and-bound method for solving binary quadratic problems. *ACM Transactions on Mathematical Software* 43, 4, (2017), 23 pages. DOI: https://doi.org/10.1145/3005345

[18] G. I. Orlova and Ya. G. Dorfman. 1972. Finding the maximal cut in a graph. *Engineering Cybernetics* 10, 3 (1972), 502–506.

[19] Jan Poland and Thomas Zeugmann. 2006. Clustering pairwise distances with missing data: maximum cuts versus normalized cuts. In *Proceedings of the Discovery Science*. Ljupčo Todorovski, Nada Lavrač, and Klaus P. Jantke (Eds.), Springer, Berlin, 197–208.

[20] Svatopluk Poljak and Zsolt Tuza. 1995. Maximum cuts and largest bipartite subgraphs. In *Proceedings of the DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. 181–244. DOI: https://doi.org/10.1090/dimacs/020/04

[21] Gerhard Reinelt. 1991. TSPLIB–A traveling salesman problem library. *ORSA Journal on Computing* 3, 4 (1991), 376–384.

[22] Franz Rendl, Giovanni Rinaldi, and Angelika Wiegele. 2010. Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations. *Mathematical Programming* 121, 2 (2010), 307–335. DOI: https://doi.org/10.1007/s10107-008-0235-8

[23] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The network data repository with interactive graph analytics and visualization. *Proceedings of the AAAI Conference on Artificial Intelligence* 29, 1 (2015), 4292–4293. DOI: https://doi.org/10.1609/aaai.v29i1.9277

[24] Sartaj Sahni and Teofilo Gonzalez. 1976. P-complete approximation problems. *Journal of the ACM* 23, 3 (1976), 555–565. DOI: https://doi.org/10.1145/321958.321975

[25] José A. Soto. 2015. Improved analysis of a max-cut algorithm based on spectral partitioning. *SIAM Journal on Discrete Mathematics* 29, 1 (2015), 259–268. DOI: https://doi.org/10.1137/14099098X

[26] Luca Trevisan. 2012. Max cut and the smallest eigenvalue. *SIAM Journal on Computing* 41, 6 (2012), 1769–1786. DOI: https://doi.org/10.1137/090773714

[27] Ryan Williams. 2005. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science* 348, 2 (2005), 357–365. DOI: https://doi.org/10.1016/j.tcs.2005.09.023

Received 30 November 2022; revised 23 June 2023; accepted 26 June 2023