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

Renee Mirka
Cornell University, Ithaca, NY, USA

David P. Williamson
Cornell University, Ithaca, NY, USA

Abstract

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 and two additional spectral algorithms. 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 return cuts with values which are competitive with those of the SDP.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases Max Cut, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.SEA.2022.19

Supplementary Material *Software (Source Code)*: `https://github.com/rmirka/max-cut-experiments`; archived at `swh:1:dir:eb13652be65db33c0ea45e66314475a4327cae0d`

Funding This work was supported by the National Science Foundation [NSF CCF-2007009].

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 [9, 15], and is well-known to be NP-hard in general; it appears on Karp's original list of NP-complete problems [12]. Additionally, Max Cut has applications in fields such as data clustering [16], circuit design, and statistical physics [1]; see Poljak and Tuza for a comprehensive survey [17].

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 $\omega < 2.376$ [24]. 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 [4]. Golovnev improved this

to $O^*(2^{(1-3/(\Delta+1))n})$ [8]. Results from Hrga et al., Hrga and Povh, Krislock, Malick, and Roupin, and Rendl, Rinaldi, and Wiegele utilize branch and bound techniques to produce further exact solvers [10, 11, 14, 19]. 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 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 [21]. 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 [7] presented a .878-approximation algorithm, which is the best possible guarantee assuming the Unique Games Conjecture [13]. 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 [23] introduced a simple .531-approximation for Max Cut based on spectral partitioning. Soto [22] 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 trade off between the computational speed and solution quality of Goemans and Williamson's SDP algorithm versus Trevisan's spectral algorithm. This paper seeks to determine whether this trade off exists in practice or if Trevisan's algorithm returns solutions competitive with those of the SDP.

Several previous papers have experimentally compared Max Cut algorithms and heuristics. Bertoni, Campadelli, and Grossi compare cuts computed by their .39-approximation Lorena algorithm, inspired by Goemans-Williamns SDP, to the SDP and a neural .5-approximation algorithm [3]. 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 [5]. Goemans and Williamson also included computational results in their initial paper 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 [2]. Dunning, Gupta, and Silberholz performed a systematic review of Max Cut heuristics and computationally tested 19 of them [6]. As far as we are aware there are no previously published results comparing Trevisan's spectral algorithm.

In this paper, we evaluate the performances of the SDP, spectral, and greedy algorithms on a variety of graphs. Section 2 provides more complete descriptions of the five algorithms 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 five algorithms that we implemented for Max Cut. Section 2.1 describes the benchmark greedy .5-approximation algorithm for Max Cut. Section 2.2 describes Trevisan's spectral algorithm for Max Cut, while Section 2.3 describes two simplifications of this algorithm. Finally Section 2.4 describes the SDP algorithm.

2.1 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 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.

2.2 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 $a_{ij} = 1$ 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 $d(i)$ the 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 $\max_i |x_i| = 1$, 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 = total weight of the edges between L and R,

X = total weight of the edges between $L \cup R$ and V', and

M = total weight of all edges − 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 $(L \cup R', R \cup L')$.

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 more than one t^2 value at each stage of the algorithm. In particular, $T = \max(5, n/k)$ values were chosen where k was tested for $k = 1, 2, 5, 10, 15, 25, 50, 100$. 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 corresponding 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 are not affected by the partial cut and aren't 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 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.

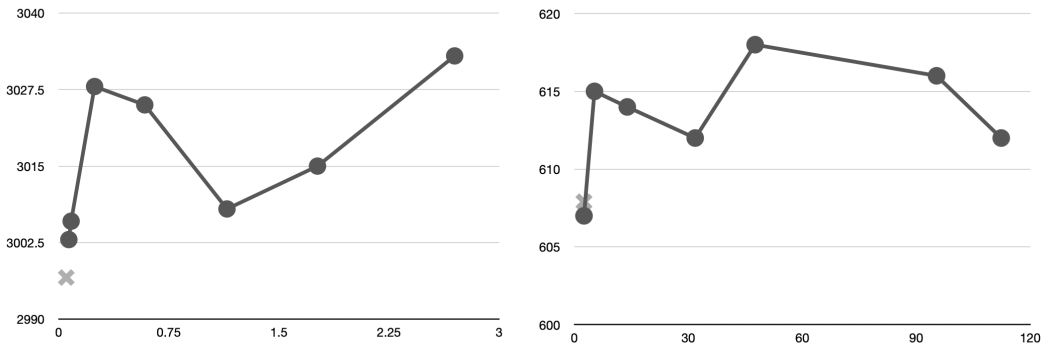
The results of these modifications for two of the tested graphs are provided in Figure 1. In each figure, the line represents the results from trials of running the algorithm multiple times (1, 2, 5, 10, 20, 35, and 50 times). Note that the line is not monotonically increasing. This is because each group 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 these are a representative sample. In general, 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 5 are beneficial. Due to this observation, we use this method of running Trevisan's algorithm 5 times and keeping the best cut for the experiments presented in this paper.

2.3 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 $t = 0$. 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.

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 $x_{i_1} \leq x_{i_2} \leq \cdots \leq x_{i_n}$. 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.



(a) johnson16-2-4.

(b) ca-netscience.

Figure 1 Plots depicting the effects on runtime and returned cut quality of running Trevisan's algorithm multiple times on the johnnson16-2-4 and ca-netscience graphs. 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.

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.

2.4 SDP Algorithm

Goemans and Williamson introduced a .878 approximation algorithm for Max Cut. Instead of directly solving

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

where again w_{ij} is the weight of edge (i, j), they relax this program to one 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

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

This gives the .878 approximation in expectation. For our testing purposes, we draw 100 random vectors instead of 1. In terms of computation time, this is a cheap modification as the SDP does not have to be rerun. We return the maximum cut resulting from these 100 random vectors.

3 Experiments

All algorithms were implemented in Julia. They were run on a machine with a 2 GHz Intel Core i5 processor with 8 GB 1867 MHz LPDDR3 memory. The SDP algorithm was computed with the JuMP modeling language for Julia and the SCS package providing the splitting cone solver. The LinearAlgebra package was used for the eigenvector computations of the spectral algorithms.

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 [18] with 29-280 vertices (average 124), and 17 sparser graphs from the Network Repository [20] with 39-1133 vertices (average 327).

3.1 Erdős-Renyi Random Graphs

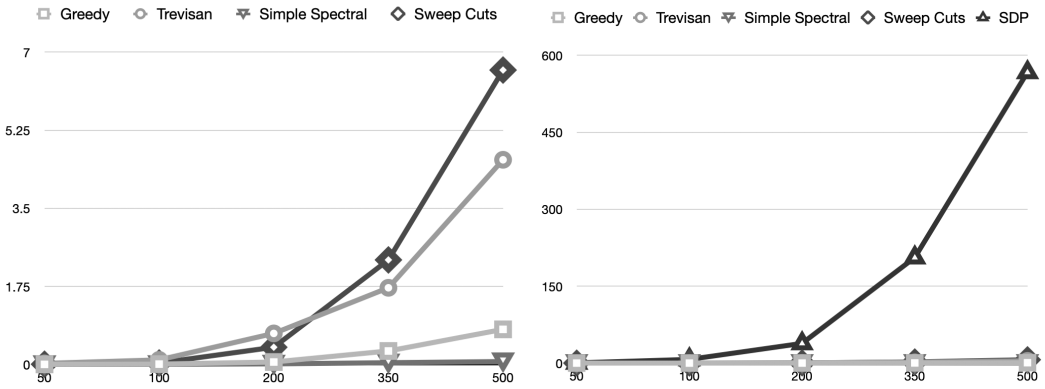
The first class of graphs tested was Erdős-Renyi random graphs. An Erdős-Renyi random graph $G(n, p)$ is a graph on n vertices where each possible edge is included independently with probability p. We tested random graphs with $n = 50, 100, 200, 350, 500$ and $p = .1, .25, .5, .75$. In our model, each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

In terms of speed, the simple spectral algorithm significantly outperformed the other algorithms on all but three tested random graphs (where greedy was faster). On the other end of the spectrum, the SDP was far slower than the alternative algorithms. The time statistics are presented in Table 1. The plots in Figure 2a and Figure 2b illustrate 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.

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
G(50,0.1)	$\mathbf{5.560 \times 10^{-3}}$	2.503×10^{-1}	5.192×10^{-2}	3.485×10^{-2}	5.556×10^{-1}
G(50,0.25)	7.600×10^{-4}	1.533×10^{-2}	$\mathbf{6.600 \times 10^{-4}}$	2.410×10^{-3}	4.711×10^{-1}
G(50,0.5)	1.280×10^{-3}	2.354×10^{-2}	$\mathbf{7.800 \times 10^{-4}}$	4.760×10^{-3}	4.502×10^{-1}
G(50,0.75)	1.870×10^{-3}	1.741×10^{-2}	$\mathbf{8.000 \times 10^{-4}}$	8.690×10^{-3}	9.727×10^{-1}
G(100,0.1)	$\mathbf{2.000 \times 10^{-3}}$	3.597×10^{-2}	2.380×10^{-3}	1.106×10^{-2}	2.929
G(100,0.25)	3.860×10^{-3}	6.945×10^{-2}	$\mathbf{2.340 \times 10^{-3}}$	1.849×10^{-2}	3.440
G(100,0.5)	7.330×10^{-3}	1.021×10^{-1}	$\mathbf{2.370 \times 10^{-3}}$	3.206×10^{-2}	7.235
G(100,0.75)	1.064×10^{-2}	1.162×10^{-1}	$\mathbf{9.960 \times 10^{-3}}$	2.653×10^{-1}	5.823
G(200,0.1)	$\mathbf{1.222 \times 10^{-2}}$	2.464×10^{-1}	3.299×10^{-2}	6.941×10^{-2}	2.575×10^{1}
G(200,0.25)	2.963×10^{-2}	2.444×10^{-1}	$\mathbf{8.650 \times 10^{-3}}$	1.892×10^{-1}	2.942×10^{1}
G(200,0.5)	5.428×10^{-2}	6.949×10^{-1}	$\mathbf{1.266 \times 10^{-2}}$	3.853×10^{-1}	3.848×10^{1}
G(200,0.75)	7.809×10^{-2}	6.463×10^{-1}	$\mathbf{9.900 \times 10^{-3}}$	4.740×10^{-1}	4.945×10^{1}
G(350,0.1)	6.192×10^{-2}	1.022	$\mathbf{2.407 \times 10^{-2}}$	4.184×10^{-1}	1.216×10^{2}
G(350,0.25)	1.737×10^{-1}	1.201	$\mathbf{3.009 \times 10^{-2}}$	1.138	1.726×10^{2}
G(350,0.5)	3.013×10^{-1}	1.718	$\mathbf{3.848 \times 10^{-2}}$	2.342	2.058×10^{2}
G(350,0.75)	4.438×10^{-1}	2.015	$\mathbf{3.324 \times 10^{-2}}$	3.015	2.798×10^{2}
G(500,0.1)	1.668×10^{-1}	1.875	$\mathbf{7.049 \times 10^{-2}}$	1.622	3.355×10^{2}
G(500,0.25)	3.937×10^{-1}	2.239	$\mathbf{5.936 \times 10^{-2}}$	3.325	3.919×10^{2}
G(500,0.5)	7.859×10^{-1}	4.587	$\mathbf{6.472 \times 10^{-2}}$	6.598	5.669×10^{2}
G(500,0.75)	1.260	5.263	$\mathbf{7.195 \times 10^{-2}}$	9.837	8.116×10^{2}

The spectral algorithms also performed the best in terms of the returned cut quality for random graphs. The SDP returned the best result for three graphs but one of the cuts was matched by Trevisan's algorithm. Trevisan's algorithm provided the best cut for 5 graphs, and the sweep cuts algorithm was the second best option for all of these, in addition to being the best for 14 graphs. These results are provided in Table 2.



(a) All tested algorithms excluding the SDP. **(b)** All tested algorithms.

Figure 2 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
G(50,0.1)	8.700×10^1	$\mathbf{9.600 \times 10^1}$	9.400×10^1	9.500×10^1	9.200×10^1
G(50,0.25)	1.970×10^2	2.060×10^2	2.060×10^2	2.080×10^2	$\mathbf{2.100 \times 10^2}$
G(50,0.5)	3.480×10^2	$\mathbf{3.600 \times 10^2}$	3.560×10^2	$\mathbf{3.600 \times 10^2}$	$\mathbf{3.600 \times 10^2}$
G(50,0.75)	5.140×10^2	5.140×10^2	4.990×10^2	5.190×10^2	$\mathbf{5.240 \times 10^2}$
G(100,0.1)	3.210×10^2	3.290×10^2	3.420×10^2	$\mathbf{3.430 \times 10^2}$	3.290×10^2
G(100,0.25)	7.640×10^2	7.830×10^2	7.850×10^2	$\mathbf{7.880 \times 10^2}$	7.860×10^2
G(100,0.5)	1.351×10^3	1.363×10^3	1.346×10^3	$\mathbf{1.375 \times 10^3}$	1.361×10^3
G(100,0.75)	2.019×10^3	2.024×10^3	2.020×10^3	$\mathbf{2.026 \times 10^3}$	2.016×10^3
G(200,0.1)	1.212×10^3	$\mathbf{1.250 \times 10^3}$	1.234×10^3	1.242×10^3	1.211×10^3
G(200,0.25)	2.795×10^3	2.859×10^3	2.847×10^3	$\mathbf{2.861 \times 10^3}$	2.778×10^3
G(200,0.5)	5.388×10^3	5.420×10^3	5.412×10^3	$\mathbf{5.423 \times 10^3}$	5.326×10^3
G(200,0.75)	7.784×10^3	7.855×10^3	7.831×10^3	$\mathbf{7.875 \times 10^3}$	7.815×10^3
G(350,0.1)	3.556×10^3	3.582×10^3	3.639×10^3	$\mathbf{3.651 \times 10^3}$	3.611×10^3
G(350,0.25)	8.378×10^3	8.544×10^3	8.583×10^3	$\mathbf{8.585 \times 10^3}$	8.236×10^3
G(350,0.5)	1.623×10^4	1.627×10^4	1.643×10^4	$\mathbf{1.649 \times 10^4}$	1.603×10^4
G(350,0.75)	2.356×10^4	$\mathbf{2.378 \times 10^4}$	2.374×10^4	2.374×10^4	2.353×10^4
G(500, .1)	7.155×10^3	7.155×10^3	7.303×10^3	$\mathbf{7.329 \times 10^3}$	7.097×10^3
G(500, .25)	1.673×10^4	1.697×10^4	1.712×10^4	$\mathbf{1.714 \times 10^4}$	1.652×10^4
G(500, .5)	3.272×10^4	3.275×10^4	3.313×10^4	$\mathbf{3.314 \times 10^4}$	3.311×10^4
G(500, .75)	4.820×10^4	$\mathbf{4.852 \times 10^4}$	4.847×10^4	4.849×10^4	4.813×10^4

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
bayg29	4.300×10^{-4}	5.040×10^{-3}	$\mathbf{3.100 \times 10^{-4}}$	9.700×10^{-4}	1.945×10^{-1}
bays29	7.500×10^{-4}	9.660×10^{-3}	$\mathbf{6.900 \times 10^{-4}}$	1.160×10^{-3}	3.002×10^{-1}
berlin52	$\mathbf{2.190 \times 10^{-3}}$	3.058×10^{-2}	2.540×10^{-3}	7.580×10^{-3}	9.291×10^{-1}
bier127	3.674×10^{-2}	3.520×10^{-1}	$\mathbf{1.370 \times 10^{-2}}$	1.125×10^{-1}	6.832
brazil58	$\mathbf{3.260 \times 10^{-3}}$	3.237×10^{-2}	4.370×10^{-3}	8.490×10^{-3}	1.229
brg180	$\mathbf{9.482 \times 10^{-2}}$	1.223	2.028×10^{-1}	3.072×10^{-1}	1.548×10^{1}
ch130	3.371×10^{-2}	3.355×10^{-1}	$\mathbf{1.221 \times 10^{-2}}$	1.352×10^{-1}	7.503
ch150	5.701×10^{-2}	9.769×10^{-1}	$\mathbf{1.811 \times 10^{-2}}$	1.745×10^{-1}	1.209×10^{1}
d198	1.094×10^{-1}	1.402	$\mathbf{3.997 \times 10^{-2}}$	4.869×10^{-1}	3.844×10^{1}
eil101	1.912×10^{-2}	5.535×10^{-1}	$\mathbf{8.870 \times 10^{-3}}$	7.724×10^{-2}	3.982
gr120	2.376×10^{-2}	4.412×10^{-1}	$\mathbf{1.050 \times 10^{-2}}$	1.153×10^{-1}	1.372×10^{1}
gr137	4.262×10^{-2}	6.078×10^{-1}	$\mathbf{1.493 \times 10^{-2}}$	1.665×10^{-1}	1.452×10^{1}
gr202	1.194×10^{-1}	2.471	$\mathbf{2.662 \times 10^{-2}}$	4.779×10^{-1}	3.067×10^{1}
gr96	1.632×10^{-2}	3.044×10^{-1}	$\mathbf{4.690 \times 10^{-3}}$	5.674×10^{-2}	4.753
kroA100	1.880×10^{-2}	2.037×10^{-1}	$\mathbf{7.720 \times 10^{-3}}$	5.644×10^{-2}	3.285
a280	2.988×10^{-1}	4.439	$\mathbf{6.127 \times 10^{-2}}$	1.534	1.555×10^{2}

3.2 Complete Graphs

The algorithms were also tested on 16 complete graphs from TSPLIB, an online library of sample instances for the Travelling Salesman Problem and related graph problems. The performance in regards to time largely mirrored that of the random graphs. The simple spectral algorithm was significantly faster than the rest of the algorithms on the vast majority of graphs, followed by the greedy, Trevisan's, and sweep cuts algorithms with relatively quick computation times, and the SDP with a massive slowdown. This data is presented in Table 3.

Again, the spectral algorithms most frequently returned the highest quality cut; these results are summarized in Table 4. For $\frac{15}{16}$ (93.75%) of these graphs, the best cut was found by either the simple spectral algorithm (5 times), Trevisan's algorithm (3 times) or the sweep cuts algorithm (12 times). Furthermore, for the graph d198 where the SDP computed the best cut, the loss in quality from the spectral solutions was quite small. These values are given in Table 5.

In Figure 3a, Figure 3b, Figure 4a, and Figure 4b, we provide a representative sample of the trade-off between runtime and returned cut value of the algorithms using the a280, ch150, and eil101 graphs.

3.3 Sparser Graphs

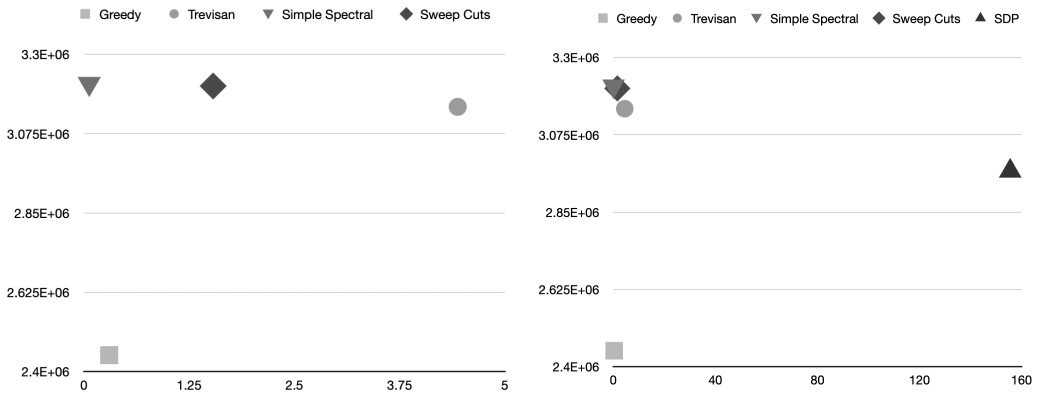
The third group of graphs is composed of a variety of graphs from the Network Repository, an online and interactive collection of network graph data coming from a variety of sources and applications. Though more structured than a random graph, these 17 graphs are sparser than the complete graphs tested in Section 3.2 and were chosen from a range of real-world scenarios. Unsurprisingly, the relationships between relative computation times remains unchanged. The simple spectral and greedy algorithms each accounted for about half of the fastest times while the SDP was consistently considerably slower (Table 6).

Table 4 The value of the cut each algorithm returned for a complete graph from TSPLIB.

Graph	Greedy	Trevisan	Simple Spectral	Sweep Cuts	SDP
bayg29	3.837×10^4	4.225×10^4	$\mathbf{4.269 \times 10^4}$	$\mathbf{4.269 \times 10^4}$	$\mathbf{4.269 \times 10^4}$
bays29	4.831×10^4	5.393×10^4	5.369×10^4	$\mathbf{5.399 \times 10^4}$	5.386×10^4
berlin52	4.532×10^5	4.616×10^5	4.465×10^5	$\mathbf{4.681 \times 10^5}$	4.522×10^5
bier127	2.162×10^7	2.300×10^7	2.322×10^7	$\mathbf{2.330 \times 10^7}$	2.320×10^7
brazil58	$\mathbf{2.319 \times 10^6}$	$\mathbf{2.319 \times 10^6}$	2.315×10^6	2.315×10^6	2.180×10^6
brg180	4.118×10^7	$\mathbf{4.616 \times 10^7}$	4.531×10^7	4.551×10^7	4.330×10^7
ch130	1.777×10^6	1.885×10^6	$\mathbf{1.888 \times 10^6}$	$\mathbf{1.888 \times 10^6}$	1.887×10^6
ch150	2.500×10^6	2.521×10^6	$\mathbf{2.526 \times 10^6}$	$\mathbf{2.526 \times 10^6}$	2.434×10^6
d198	9.635×10^6	1.286×10^7	1.292×10^7	1.293×10^7	$\mathbf{1.293 \times 10^7}$
eil101	1.052×10^5	$\mathbf{1.070 \times 10^5}$	1.063×10^5	1.064×10^5	1.058×10^5
gr120	2.123×10^6	2.147×10^6	2.156×10^6	$\mathbf{2.157 \times 10^6}$	2.154×10^6
gr137	2.241×10^7	3.044×10^7	3.066×10^7	$\mathbf{3.070 \times 10^7}$	3.070×10^7
gr202	1.372×10^7	1.533×10^7	1.559×10^7	$\mathbf{1.593 \times 10^7}$	1.581×10^7
gr96	8.967×10^6	1.156×10^7	$\mathbf{1.166 \times 10^7}$	$\mathbf{1.166 \times 10^7}$	1.157×10^7
kroA100	5.848×10^6	5.850×10^6	$\mathbf{5.897 \times 10^6}$	$\mathbf{5.897 \times 10^6}$	$\mathbf{5.897 \times 10^6}$
a280	2.447×10^6	3.151×10^6	3.21×10^6	$\mathbf{3.21 \times 10^6}$	2.970×10^6

Table 5 The percent decrease in cut value from the SDP to the spectral cuts.

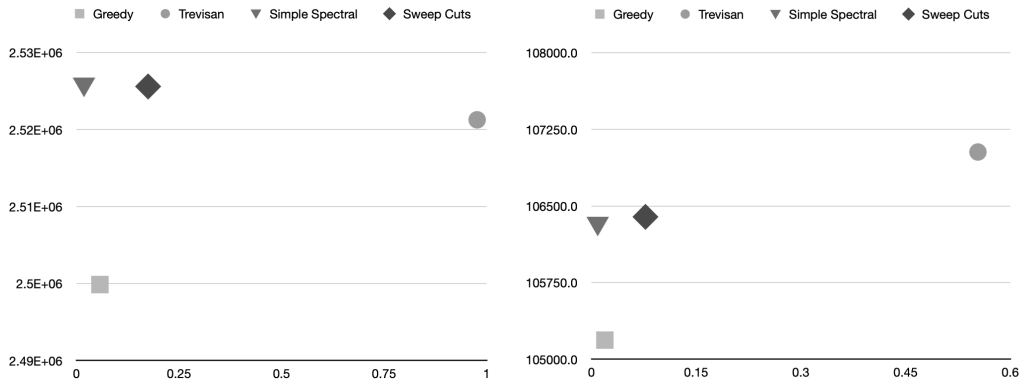
Graph	Trevisan	Simple Spectral	Sweep Cuts
d198	$\sim .6\%$	$\sim .1\%$	$\sim .06\%$



(a) All tested algorithms excluding the SDP.

(b) All tested algorithms.

Figure 3 Plots depicting the computation time and returned cut values of algorithms on the a280 graph. The X and Y axes are the runtime in seconds and the returned cut value, respectively.



(a) ch150.

(b) eil101.

Figure 4 Plots depicting the computation time and returned cut values of algorithms excluding the SDP on the ch150 and eil101 graphs. The X and Y axes are the runtime in seconds and the returned cut value, respectively.

For this group of graphs, the algorithms' relative cut quality is more varied than with the previous. Of the 17 graphs tested, the SDP returned the best cut for 7 instances whereas the spectral algorithms combined for 11 best (with one instance of a tie between the SDP and simple spectral) (Table 7).

In Figure 5a, Figure 5b, Figure 6a, and Figure 6b, we provide a representative sample of the trade-off between runtime and returned cut value of the algorithms using the graphs ia-infect-dublin, email-enron-only, and soc-dolphins.

4 Conclusion

The goal of this paper was to compare Max Cut algorithms with varying approximation guarantees in practice. In particular, we know the SDP has the provably best approximation guarantee; 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? 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. The results of this experiment appear to illustrate spectral algorithms are competitive with the SDP algorithm in practice. This suggests that the investigation into approximation guarantees is a direction for further theoretical study.

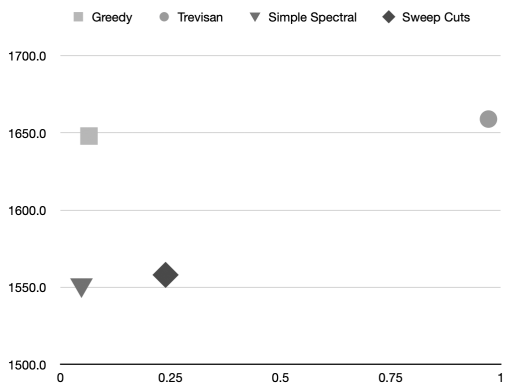
In terms of practical implementations, for the graphs that the SDP seems to perform better on, one could consider running Trevisan's algorithm for even more than 5 iterations and choosing the best cut returned. The magnitude of the speed advantage of Trevisan's algorithm would allow for many runs before 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.

Table 6 The time in seconds each algorithm took to compute a cut of a graph from the Network Repository arising in the real-world.

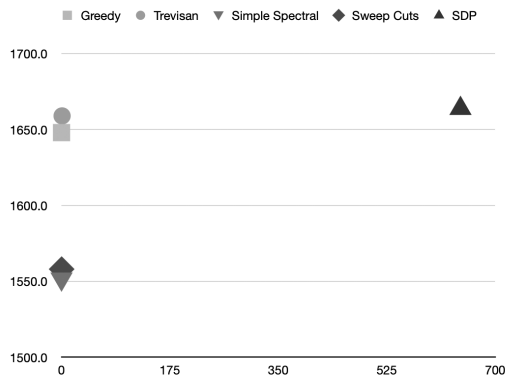
Graph	# vertices	# edges	Greedy	Trevisan	Simple Spectral	Sweep Cuts	SDP
ENZYMES8	88	133	$\mathbf{1.370 \times 10^{-3}}$	4.342×10^{-2}	1.776×10^{-1}	1.015×10^{-2}	2.356×10^{1}
eco-stmarks	54	356	$\mathbf{5.900 \times 10^{-4}}$	2.146×10^{-2}	1.993×10^{-1}	2.597×10^{-2}	6.939
johnson16-2-4	120	5460	2.519×10^{-2}	1.603×10^{-1}	$\mathbf{2.600 \times 10^{-3}}$	8.544×10^{-2}	8.178×10^{-1}
hamming6-2	64	1824	3.540×10^{-3}	5.081×10^{-2}	$\mathbf{1.190 \times 10^{-3}}$	1.558×10^{-2}	9.926×10^{-1}
ia-infect-hyper	113	2196	8.920×10^{-3}	1.248×10^{-1}	$\mathbf{3.280 \times 10^{-3}}$	4.649×10^{-2}	6.279
soc-dolphins	62	159	$\mathbf{4.600 \times 10^{-4}}$	1.845×10^{-2}	8.900×10^{-4}	1.690×10^{-3}	2.464
email-enron-only	143	623	5.960×10^{-3}	1.174×10^{-1}	$\mathbf{5.650 \times 10^{-3}}$	1.575×10^{-2}	5.681×10^{1}
dwt_209	209	976	1.349×10^{-2}	3.012×10^{-1}	$\mathbf{8.380 \times 10^{-3}}$	4.641×10^{-2}	7.073×10^{1}
inf-USAir97	332	2126	$\mathbf{5.780 \times 10^{-2}}$	2.944	7.350×10^{-2}	2.258×10^{-1}	3.361×10^{2}
ca-netscience	379	914	$\mathbf{2.590 \times 10^{-2}}$	5.124×10^{-1}	8.440×10^{-2}	1.146×10^{-1}	3.584×10^{2}
ia-infect-dublin	410	2765	6.480×10^{-2}	9.720×10^{-1}	$\mathbf{4.770 \times 10^{-2}}$	2.387×10^{-1}	6.438×10^{2}
road-chesapeake	39	170	$\mathbf{4.000 \times 10^{-4}}$	8.000×10^{-3}	5.000×10^{-4}	1.200×10^{-3}	2.759×10^{-1}
Erdos991	492	1417	$\mathbf{4.490 \times 10^{-2}}$	2.634	6.090×10^{-2}	1.933×10^{-1}	5.143×10^{2}
dwt_503	503	3265	7.240×10^{-2}	2.039	$\mathbf{6.640 \times 10^{-2}}$	3.471×10^{-1}	1.081×10^{3}
p-hat700-1	700	60999	1.264	1.009×10^{1}	$\mathbf{1.591 \times 10^{-1}}$	9.273	1.270×10^{3}
DD687	725	2600	$\mathbf{9.390 \times 10^{-2}}$	4.332	3.816×10^{-1}	6.521×10^{-1}	3.320×10^{3}
email-univ	1133	5451	$\mathbf{2.081 \times 10^{-1}}$	1.345×10^{1}	1.179	2.703	7.572×10^{3}

Table 7 The value of the cut each algorithm returned for a graph from the Network Repository.

Graph	# vertices	# edges	Greedy	Trevisan	Simple Spectral	Sweep Cuts	SDP
ENZYMES8	88	133	1.170×10^{2}	$\mathbf{1.260 \times 10^{2}}$	$\mathbf{1.260 \times 10^{2}}$	$\mathbf{1.260 \times 10^{2}}$	$\mathbf{1.260 \times 10^{2}}$
eco-stmarks	54	356	8.891×10^{2}	$\mathbf{1.190 \times 10^{3}}$	9.354×10^{2}	9.354×10^{2}	9.601×10^{2}
johnson16-2-4	120	5460	$\mathbf{3.036 \times 10^{3}}$	$\mathbf{3.036 \times 10^{3}}$	2.958×10^{3}	2.986×10^{3}	2.918×10^{3}
hamming6-2	64	1824	$\mathbf{9.920 \times 10^{2}}$	$\mathbf{9.920 \times 10^{2}}$	9.680×10^{2}	9.690×10^{2}	9.760×10^{2}
ia-infect-hyper	113	2196	1.213×10^{3}	$\mathbf{1.233 \times 10^{3}}$	1.227×10^{3}	1.227×10^{3}	1.211×10^{3}
soc-dolphins	62	159	1.120×10^{2}	1.120×10^{2}	1.190×10^{2}	$\mathbf{1.210 \times 10^{2}}$	1.150×10^{2}
email-enron-only	143	623	3.920×10^{2}	$\mathbf{4.130 \times 10^{2}}$	3.710×10^{2}	3.800×10^{2}	3.960×10^{2}
dwt_209	209	976	5.250×10^{2}	5.270×10^{2}	5.250×10^{2}	5.270×10^{2}	$\mathbf{5.400 \times 10^{2}}$
inf-USAir97	332	2126	9.661×10^{1}	9.820×10^{1}	8.184×10^{1}	9.337×10^{1}	$\mathbf{1.074 \times 10^{2}}$
ca-netscience	379	914	5.830×10^{2}	5.880×10^{2}	5.270×10^{2}	5.270×10^{2}	$\mathbf{6.110 \times 10^{2}}$
ia-infect-dublin	410	2765	1.648×10^{3}	1.659×10^{3}	1.550×10^{3}	1.558×10^{3}	$\mathbf{1.664 \times 10^{3}}$
road-chesapeake	39	170	1.230×10^{2}	1.230×10^{2}	1.210×10^{2}	1.230×10^{2}	$\mathbf{1.250 \times 10^{2}}$
Erdos991	492	1417	9.330×10^{2}	$\mathbf{9.340 \times 10^{2}}$	7.350×10^{2}	7.580×10^{2}	9.240×10^{2}
dwt_503	503	3265	1.822×10^{3}	1.822×10^{3}	$\mathbf{1.921 \times 10^{3}}$	$\mathbf{1.921 \times 10^{3}}$	1.909×10^{3}
p-hat700-1	700	60999	3.261×10^{4}	3.269×10^{4}	3.215×10^{4}	$\mathbf{3.305 \times 10^{4}}$	3.304×10^{4}
DD687	725	2600	1.669×10^{3}	1.671×10^{3}	1.616×10^{3}	1.617×10^{3}	$\mathbf{1.680 \times 10^{3}}$
email-univ	1133	5451	$\mathbf{3.546 \times 10^{3}}$	$\mathbf{3.546 \times 10^{3}}$	3.341×10^{3}	3.344×10^{3}	3.264×10^{3}

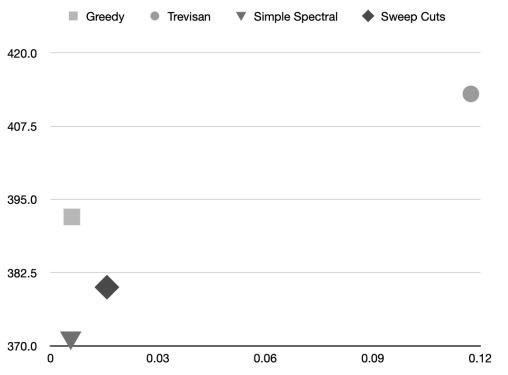


(a) All tested algorithms excluding the SDP.

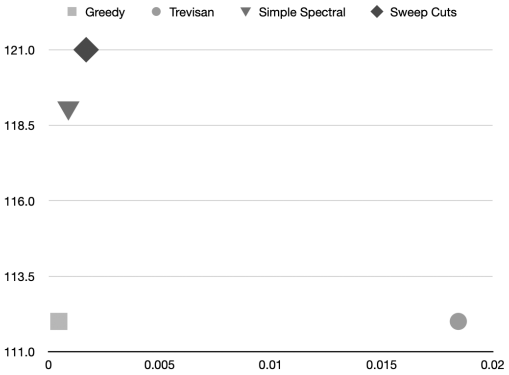


(b) All tested algorithms.

Figure 5 Plots depicting the computation time and returned cut values of algorithms on the ia-infect-dublin graph. The X and Y axes are the runtime in seconds and the returned cut value, respectively.



(a) email-enron-only.



(b) soc-dolphins.

Figure 6 Plots depicting the computation time and returned cut values of algorithms excluding the SDP on the email-enron-only and soc-dolphins graphs. The X and Y axes are the runtime in seconds and the returned cut value, respectively.

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 performances 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 in practice. Can the analysis of this algorithm be improved?

References

1. Francisco Barahona, Martin Grötschel, Michael Jünger, and Gerhard Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36(3):493–513, 1988. doi:10.1287/opre.36.3.493.
2. Jonathan W. Berry and Mark K. Goldberg. Path optimization for graph partitioning problems. *Discrete Appl. Math.*, 90(1–3):27–50, January 1999. doi:10.1016/S0166-218X(98)00084-5.
3. Alberto Bertoni, Paola Campadelli, and Giuliano Grossi. An approximation algorithm for the maximum cut problem and its experimental analysis. *Discrete Applied Mathematics*, 110:3–12, 2001. doi:10.1016/S0166-218X(00)00299-7.
4. F. Della Croce, M.J. Kaminski, and V.Th. Paschos. An exact algorithm for MAX-CUT in sparse graphs. *Operations Research Letters*, 35(3):403–408, 2007. doi:10.1016/j.orl.2006.04.001.
5. Oliver Dolezal, Thomas Hofmeister, and Hanno Lefmann. A comparison of approximation algorithms for the maxcut-problem, May 2000. doi:10.17877/DE290R-5013.
6. Iain Dunning, Swati Gupta, and John Silberholz. What works best when? A systematic evaluation of heuristics for max-cut and QUBO. *INFORMS Journal on Computing*, 30(3):608–624, 2018. doi:10.1287/ijoc.2017.0798.
7. Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, November 1995. doi:10.1145/227683.227684.
8. Alexander Golovnev. New upper bounds for MAX-2-SAT and MAX-2-CSP w.r.t. the average variable degree. In Dániel Marx and Peter Rossmanith, editors, *Parameterized and Exact Computation*, pages 106–117, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
9. F. Hadlock. Finding a maximum cut of a planar graph in polynomial time. *SIAM Journal on Computing*, 4(3):221–225, 1975. doi:10.1137/0204019.
10. Timotej Hrga, Borut Lužar, Janez Povh, and Angelika Wiegele. BiqBin: Moving boundaries for NP-hard problems by HPC. In Ivan Dimov and Stefka Fidanova, editors, *Advances in High Performance Computing*, pages 327–339, Cham, 2021. Springer International Publishing.
11. Timotej Hrga and Janez Povh. MADAM: A parallel exact solver for max-cut based on semidefinite programming and ADMM. *Comput. Optim. Appl.*, 80(2):347–375, November 2021. doi:10.1007/s10589-021-00310-6.
12. Richard Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, volume 40, pages 85–103, January 1972. doi:10.1007/978-3-540-68279-0_8.

13 Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? *SIAM J. Comput.*, 37(1):319–357, April 2007. doi:10.1137/S0097539705447372.

14 Nathan Krislock, Jérôme Malick, and Frédéric Roupin. BiqCrunch: A semidefinite branch-and-bound method for solving binary quadratic problems. *ACM Trans. Math. Softw.*, 43(4), January 2017. doi:10.1145/3005345.

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

16 Jan Poland and Thomas Zeugmann. Clustering pairwise distances with missing data: Maximum cuts versus normalized cuts. In Ljupčo Todorovski, Nada Lavrač, and Klaus P. Jantke, editors, *Discovery Science*, pages 197–208, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

17 Svatopluk Poljak and Zsolt Tuza. Maximum cuts and largest bipartite subgraphs. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, volume 20, pages 181–244, 1995. doi:10.1090/dimacs/020/04.

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

19 Franz Rendl, Giovanni Rinaldi, and Angelika Wiegele. Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations. *Mathematical Programming*, 121:307–335, February 2010. doi:10.1007/s10107-008-0235-8.

20 Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015. URL: https://networkrepository.com.

21 Sartaj Sahni and Teofilo Gonzalez. P-Complete approximation problems. *J. ACM*, 23(3):555–565, July 1976. doi:10.1145/321958.321975.

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

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

24 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005. Automata, Languages and Programming: Algorithms and Complexity (ICALP-A 2004). doi:10.1016/j.tcs.2005.09.023.