Purdue University

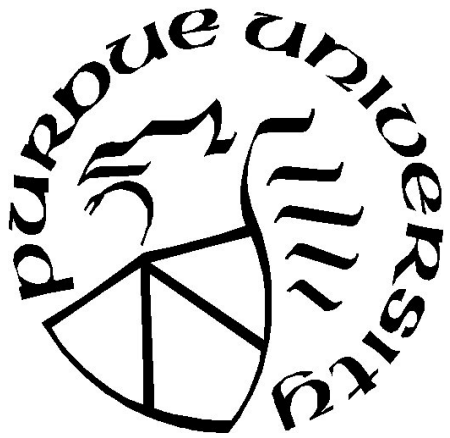
Computational Methods in Optimization

CS 520

Maximum Cut

Author:
Javad Darivandpour
Negin Karisani

Instructor:
Jean Honorio



April 28, 2016

Contents

Abstract

Finding a *maximum cut* in an undirected graph is a well known problem in graph theory; its decision version is known to be NP-complete, and obviously the optimization version, yet cannot be solved accurately with better than exponential complexity. There are brute-force (using branch & bound technique) algorithms to find the optimal solution. Also, greedy approaches have been proposed to find near-optimal solutions. Here, in this project we intend to provide an approximation (upper bound) using a version interior point methods, called dual scaling algorithm, we learned in class.

1 Introduction

In this report, sections 2 and 3 state definitions of *Maximum Cut* problem and *Boolean Optimization* respectively. In section 4 we formalize Maximum cut problem algebraically into semidefinite programs, in two forms including *standard* and *laplacian* forms. Section 5 explains primal form of SDPs and constructs their dual. In section 6, we obtain the dual for each SDP form we had in section 4. On the basis of a solid mathematical formalization of problem, section 7 discusses how to solve optimization problems obtained earlier, using *Dual Scaling* algorithm. Finally in section 8, experimental results are provided to check the performance of discussed optimization method.

2 Maximum Cut Problem

For an undirected graph $G =< V, E >$, a maximum cut is one whose size is larger that or equal all other cuts in the graph. In **optimization version**, we are looking for $\emptyset \neq S \subsetneq V$ such that number (weight) of edges with one end in S and the other in $V \setminus S$ is maximized. On the other hand, in the **decision version**, given input K, one needs to verify whether there exists a cut with value greater than or equal to K [1].

3 Boolean Optimization

Boolean Quadratic Optimization is a combinatorial optimization problem, in which we are trying to minimize a quadratic function, where the decision variables can take only two values. Formally, we can write it as follows: (Q is a symmetric matrix of dimension $n \times n$.)

$$\begin{aligned} \text{minimize} \quad & x^T Q x \\ \text{subject to} \quad & x \in \{-1, 1\}^n \\ & (\text{i.e. } \forall i \;\; x_i \in \{-1, 1\}) \end{aligned} \tag{1}$$

Many combinatorial optimization problems, including *Maximum Cut* problem, can be written in the form above as a *boolean optimization*.

4 Max-Cut Optimization as Boolean Optimization

Here we attempt to express Max-Cut problem in the format of boolean optimization as discussed in previous section. We will do so in *Standard* and *Laplacian* forms.

4.1 Standard Form

In order to formalize Max-Cut problem (for graph $G =< V, E >$) in standard boolean optimization form, we can present any element in S by 1 and others (elements in $V \setminus S$) by

[1] Decision version of Max-Cut problem is known to be NP-complete.

-1, and solve the following problem:

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2} \sum_{\substack{i,j \in [n] \\ i<j}} w_{i,j}(1 - v_i v_j) \\ \text{subject to} \quad & v_i^2 - 1 = 0 \\ & (\text{i.e. } \forall i \;\; v_i \in \{-1, 1\}) \end{aligned} \tag{2}$$

Where $w_{i,j}$ [2] is the weight of edge (v_i, v_j), and value of v_i indicates whether this vertex is in S or not. Now if we remove all constants in 2, we have:

$$\begin{aligned} \text{minimize} \quad & \sum_{i,j \in [n]} w_{i,j} v_i v_j \\ \text{subject to} \quad & v_i^2 - 1 = 0 \end{aligned} \tag{3}$$

Let W be the weighted adjacency matrix for graph G [3]. Now, we can rewrite problem 2 as

$$\begin{aligned} \text{minimize} \quad & v^T W v \\ \text{subject to} \quad & v_i^2 - 1 = 0 \end{aligned} \tag{4}$$

which obviously is boolean quadratic programming.

4.2 Laplacian Form

Starting from equation 2, considering W being symmetric and the fact that $v_i^2 = 1$, we can reform it as

$$\begin{aligned} \text{maximize} \quad & \frac{1}{4} \sum_{i,j \in [n]} w_{i,j}(1 - v_i v_j) \\ \text{subject to} \quad & v_i^2 - 1 = 0 \\ & (\text{i.e. } \forall i \;\; v_i \in \{-1, 1\}) \end{aligned} \tag{5}$$

Rearranging the objective function as follows[4]

$$\begin{aligned} \frac{1}{4} \sum_{i,j \in [n]} w_{i,j}(1 - v_i v_j) &= \frac{1}{4} \sum_{i=1}^{n} (\sum_{j=1}^{n} w_{i,j} v_i v_i - \sum_{j=1}^{n} w_{i,j} v_i v_j) \\ &= \frac{1}{4} v^T (Diag(W\vec{1}) - W) v \end{aligned} \tag{6}$$

where $Diag(x)$ is a diagonal matrix with elements of x on its main diagonal and zero elsewhere. Here, the matrix $L \coloneqq \frac{1}{4}(Diag(W\vec{1}) - W)$ is the **Laplacian Matrix** of the graph W, and it is symmetric due to the symmetry of W. Now, the following optimization problem is equivalent to problem 5

$$\begin{aligned} \text{maximize} \quad & v^T L v \\ \text{subject to} \quad & v_i^2 - 1 = 0 \end{aligned} \tag{7}$$

5 Primal-Dual Pair of SDP

Here we discuss the standard formulation of Dual problem for Semidefinite Programing problems using **adjoint operator**. Later we will use it to obtain the dual for problem 7.

[2] If G is unweighted graph (so number of edges is important) we have $\forall i \; \forall j \; w_{i,j} \in \{0, 1\}$

[3] Since G is undirected, $W \in \mathcal{S}^n$.

[4] Notice that W is symmetric, and $\forall i (v_i^2 = 1)$.

5.1 Primal SDP

Consider the following primal SDP problem, where $W \in S_n$ is the cost matrix, and $V \in S_n$ is the variable matrix

$$\begin{aligned} \text{maximize} \quad & < W \bullet V > \\ \text{subject to} \quad & \mathcal{A}V = b \\ & V \succeq 0 \end{aligned} \tag{8}$$

Also, we have the linear operator $\mathcal{A} : S_n \to \mathbb{R}^m$ defined as below, where $A_i \in S_n$ for $i = 1, \ldots, m$

$$\mathcal{A}V = \begin{pmatrix} < A_1 \bullet V > \\ < A_2 \bullet V > \\ \vdots \\ < A_m \bullet V > \end{pmatrix}$$

The **adjoint operator** with respect to $\mathcal{A}$, shown by $\mathcal{A}^\dagger$, will be defined as follows

$$< \mathcal{A}V \bullet y > = < V \bullet \mathcal{A}^\dagger y >$$

hence we obtain

$$< \mathcal{A}V \bullet y > = \sum_{i=1}^{m} < A_i \bullet V > y_i = < \sum_{i=1}^{m} y_i A_i \bullet V >$$

Thus

$$\mathcal{A}^\dagger y = \sum_{i=1}^{m} y_i A_i \tag{9}$$

5.2 Dual SDP

We use the lagrangian approach to obtain the dual for problem 8. After mathematical manipulation we have the following

$$\begin{aligned} \text{minimize} \quad & b^T \lambda \\ \text{subject to} \quad & \mathcal{A}^\dagger \lambda - Z = W \\ & Z \succeq 0 \end{aligned} \tag{10}$$

where $\lambda \in \mathbb{R}^m$ is the lagrangian multiplier for equality constraints.

6 Dual of Max-Cut

In this section, we attempt to compute the dual for Max-Cut problem. One achieves various results due to different tricks used; each one includes particular constraints. Consequently, various relaxations can be applied[5].

6.1 Dual of Standard Form Max-Cut

For the optimization problem 4, we have

$$\begin{aligned} L(v, \lambda) &= v^T W v - \sum_{i=1}^{n} \lambda_i (v_i^2 - 1) \\ &= v^T (W - \Lambda) v + tr(\Lambda) \end{aligned}$$

[5]Based on the algorithm one uses, one of these will be used

where $\Lambda = diag(\lambda_1, \lambda_2, \ldots, \lambda_n)$. The implicit constraint $W - \Lambda \succeq 0$ is mandatory to make the lagrangian dual $g(\lambda) = \inf_v L(v, \lambda)$ bounded below. Trying to find $\inf_v L(v, \lambda)$ we have the dual problem as follows

$$\begin{aligned} \text{maximize} \quad & tr(\Lambda) \\ \text{subject to} \quad & W - \Lambda \succeq 0 \end{aligned} \tag{11}$$

As we know, this is one of the dual forms for the SDP problems.

6.2 Dual of Laplacian Form Max-Cut

Applying the result of section 5 on problem 7 (Laplacian version of Max-Cut) we obtain its dual, however, before that we need to convert problem 7 to the form of problem 8. In order to do so

$$v^T L v = \sum_{i=1}^{n}\sum_{j=1}^{n} v_i L_{ij} v_j = \sum_{i=1}^{n}\sum_{j=1}^{n} L_{ij} v_i v_j =< L \bullet vv^T >$$

Let's have the notation $V \coloneqq vv^T$ which is symmetric positive semidefinite by construction. Since $v \in \{-1, 1\}^n$, we conclude V has rank one. Also, any element on main diagonal of V is 1. If we relax V to be any matrix with the aforementioned properties[6] we obtain the following problem[7]

$$\begin{aligned} \text{maximize} \quad & < L \bullet V > \\ \text{subject to} \quad & diag(V) = \vec{1} \\ & rank(V) = 1 \\ & V \succeq 0 \end{aligned} \tag{12}$$

The constraint $diag(V) = \vec{1}$ can be written in the form of $\mathcal{A}V = \vec{1}$. However, $rank(V) = 1$ cannot be reformulated as a linear equality constraint. As a result, problem above is not a SDP[8]. By dropping the problematic constraint, $rank(V) = 1$, now we have a SDP[9], which is as follows

$$\begin{aligned} \text{maximize} \quad & < L \bullet V > \\ \text{subject to} \quad & diag(V) = \vec{1} \\ & V \succeq 0 \end{aligned} \tag{13}$$

In order to have problem 13 in the dual format of 10 we first need to figure out adjoint operator for $diag(\cdot)$. Consider $V \in S_n$ and $y \in \mathbb{R}^n$

$$\begin{aligned} < diag(V) \bullet y >= \sum_{i=1}^{n} v_{ii} y_i &= \sum_{i=1}^{n} v_{ii} (Diag(y))_{ii} \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} v_{ij} (Diag(y))_{ij} =< V \bullet Diag(y) > \end{aligned}$$

So, the final dual version for problem 13 is

$$\begin{aligned} \text{minimize} \quad & \vec{1}^T y \\ \text{subject to} \quad & Diag(y) - Z = L \\ & Z \succeq 0 \end{aligned} \tag{14}$$

[6] First relaxation in this approach.
[7] $diag(V) : S_n \to \mathbb{R}^n$ is defined as a vector with values on main diagonal of V
[8] But it is still equivalent to 7
[9] This is the second relaxation in this approach

Now, we will reform equation 13 in the standard minimization form

$$\begin{aligned} \text{minimize} \quad & < -L \bullet V > \\ \text{subject to} \quad & diag(V) = \vec{1} \\ & V \succeq 0 \end{aligned} \tag{15}$$

Consequently, its dual will be as follows

$$\begin{aligned} \text{maximize} \quad & \vec{1}^T y \\ \text{subject to} \quad & Diag(y) + Z = -L \\ & Z \succeq 0 \end{aligned} \tag{16}$$

7 Solving Relaxed Max-Cut SDP

After formalizing the problem and obtaining its dual, it is time to apply optimization algorithms to find the optimum values. So far, we have SDP problems (both in primal and dual version); there are many algorithms proposed for semidefinite programing including (1) subgradient methods such as Spectral Bundle Method, (2) interior point approaches and so on.

7.1 Using Laplacian Form to Estimate Max-Cut

As mentioned earlier in section 4, we can state Max-Cut problem in Laplacian form as follows (problem 7)

$$\begin{aligned} \text{maximize} \quad & v^T L v \\ \text{subject to} \quad & v_i^2 - 1 = 0 \end{aligned}$$

Using any matrix $V \succeq 0$ instead of only $V = vv^T$ where $v \in \{0,1\}^n$ with $rank(V) = 1$, i.e. after applying two steps of relaxation, we obtain problem 13 as follows

$$\begin{aligned} \text{maximize} \quad & < L \bullet V > \\ \text{subject to} \quad & diag(V) = \vec{1} \\ & V \succeq 0 \end{aligned}$$

Since this is a relaxation of Max-Cut problem (specifically problem 7), we expect to gain a larger value than global optimum of Max-Cut. This output is the upper bound (estimation) we talk about. We use this problem in standard form[10] as stated in problem 15.

After all, we solve the latter problem using some interior point method on its dual. Also, strong duality holds for this problem, so optimal value for problem 16 is also the optimal value for problem 15. The reason for strong duality is that in problem 13 first constraint is an equality constraint; about inequality constraint $V \succeq 0$, the matrix V has been initially defined as $V = vv^t$, hence for feasible v in Max-Cut we have $V \succ 0$.[11]

7.2 SDP Optimization Methods

A pair of primal/dual semidefinite programs can be solved in polynomial time. Theoretically speaking, primal/dual interior point methods work well on reasonably defined semidefinite programs. In other words, such methods are practical for problems of sensible size (i.e.

[10] Which is simply negative of problem 13.

[11] A real symmetric matrix H is positive definite iff there exists a real matrix L such that $H = LL^T$

around 7000 to 10000 constraints[12]). There are several polynomial time interior point algorithms to solve semidefinite programs, including *Primal Scaling*, *Primal-Dual Scaling* and *Dual Scaling* algorithms [4].

Primal Scaling Algorithm is analogue to primal path following and potential reduction for linear programming Which uses only primal variables to calculate search direction in each iteration [4].

Dual Scaling Algorithm, which is the algorithm we use in this project, is analogue to dual path following and potential reduction for linear programming Which uses only dual variables to calculate search direction in each iteration [4].

Primal-Dual Scaling Algorithm uses both primal and dual variables to generate the iterate direction [4].

All of these methods update both primal and dual iterates in each step, and their iteration complexity $O(\sqrt{n}\ln(1/\varepsilon))$ to obtain duality gap with accuracy ε [4]. In case of our problem, duality gap is 0 (due to strong duality), so number of iterations we need to achieve optimum point with accuracy ε is proportional to the given bound.

7.3 Dual Scaling Algorithm

As an interior point method we select the *Dual Scaling*[13] algorithm defined in [4, 5]. Dual Scaling is efficient in case of problems which have low-rank solutions; as mentioned earlier in the dual for Max-Cut we have constraint $rank(V) = 1$, which makes dual scaling a good match for our problem.

Benson et al. propose Dual Scaling algorithm for solving large-scale sparse SDPs in [4]. This algorithm is a potential reduction method, and uses a *dual potential function* to solve the problem. Most often in combinatorial problems, the primal is dense while its dual can be very sparse. Dual scaling uses a potential function which captures only the dual problem; this provides the chance to take the most benefit of sparsity. Suppression of computational cost, and exploring sparsity as well as exploiting low-rank structure are significant benefits of dual scaling algorithm.

The dual scaling algorithm proposed in [4] for SDP problems, as a modification of dual scaling for linear programs, reduces the Tanabe-Todd-Ye primal-dual potential function [4, 5]

$$\Psi(V, Z) = \rho \ln(V \bullet Z) - \ln\det V - \ln\det Z \tag{17}$$

The first term in the potential function, aims to reduce the duality gap, while second and third terms of the function keep matrices V and Z in the interior of positive semidefinite cone.

Also, ρ sets the step size for moving in step direction. Larger values of ρ seek the optimal value more aggressively, however, are more likely to yield infeasible solutions [4, 5]. In [4], authors start algorithm with a large initial ρ, and after a couple of iterations set it dynamically based on the iteration gap. Since our experiments are on small input graphs[14], we avoided this idea in our implementation.

[12] This is the case with current technology.

[13] It is implemented in the software DSDP, which is considered particularly efficient for solving problems where the solution is known to have low-rank

[14] Due to computational difficulties.

In the dual scaling algorithm, we use the following reduced potential function (problem 16)

$$\psi(y, \bar{z}) = \rho \ln(\bar{z} - \vec{1}^T y) - \ln \det Z \tag{18}$$

where $\bar{z}$ is $< L \bullet V >$ for some feasible V [4, 5]. Notice that in the standard form problem 15 we are minimizing $< -L \bullet V >$, hence the potential function used in our implementation is

$$\psi(y, \bar{z}) = -\rho \ln(\bar{z} - \vec{1}^T y) - \ln \det Z \tag{19}$$

and $\bar{z} =< -L \bullet V >$ for some feasible V.

The aforementioned algorithm only uses a dual feasible iterate to construct the search direction, but it can be interpreted as a method that also attempts to compute a primal value for some primal feasible solution at each iteration [6]. All details of dual scaling algorithm for SDPs, most importantly reduction estimation in potential function and computing search directions are based on LEMMA 1 in [4] and its corollaries; for further details the reader may refer to the paper.

8 Experimental Results

After implementing *Dual Scaling* method, and running it to solve[15] problem 16, here we provide some sample results. Due to computation difficulties[16], we run all our experiments using graphs with 10 to 20 vertices. All graphs are generated randomly, with rational weighted edges.

8.1 Internal Behavior of Dual Scaling

Based on the theory behind interior point methods (in particular dual scaling algorithm), we expect step by step improvement as algorithm goes on; i.e., in general, starting with an initial point, we expect to get closer to optimum after each iteration is done.

In order to verify internal progress for dual scaling algorithm, we have plotted the changes after each iteration is done for several graphs. Here we provide some of them with sizes 10, 15 and 20 in Figure 1.[17] Also, we have provided same plot for the graph of 10 vertices separately in Figure 2 to illustrate changes more elaborately. Check captions for further details.

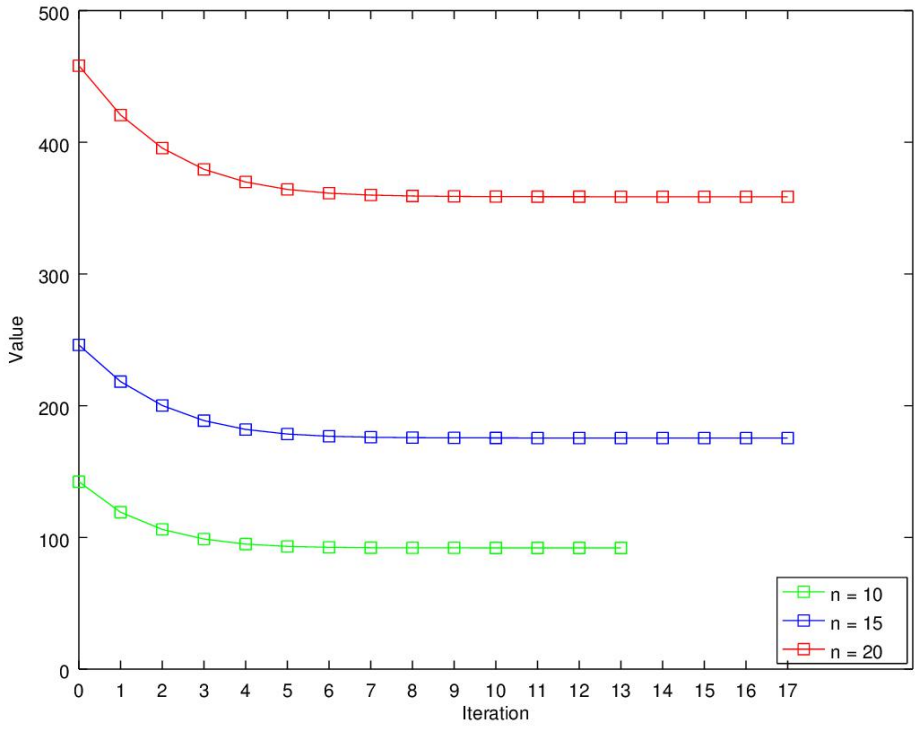


Figure 1: This figure illustrates changes in optimal value after each iteration in 3 separate experiments with input sizes 10, 15 and 20. We used same parameters for all three cases. As expected, in each iteration we get closer to the optimal value. During first few iterations, we encounter rapid decrement in approximation for the Cut value. However, in all cases after some point, this value decreases slowly until stopping criterion is satisfied.

[15] Due to strong duality, optimal solution for problem 15 is the same.
[16] Language we are using and hardware constraints
[17] All edge weights are rational numbers in range $(1, 10)$

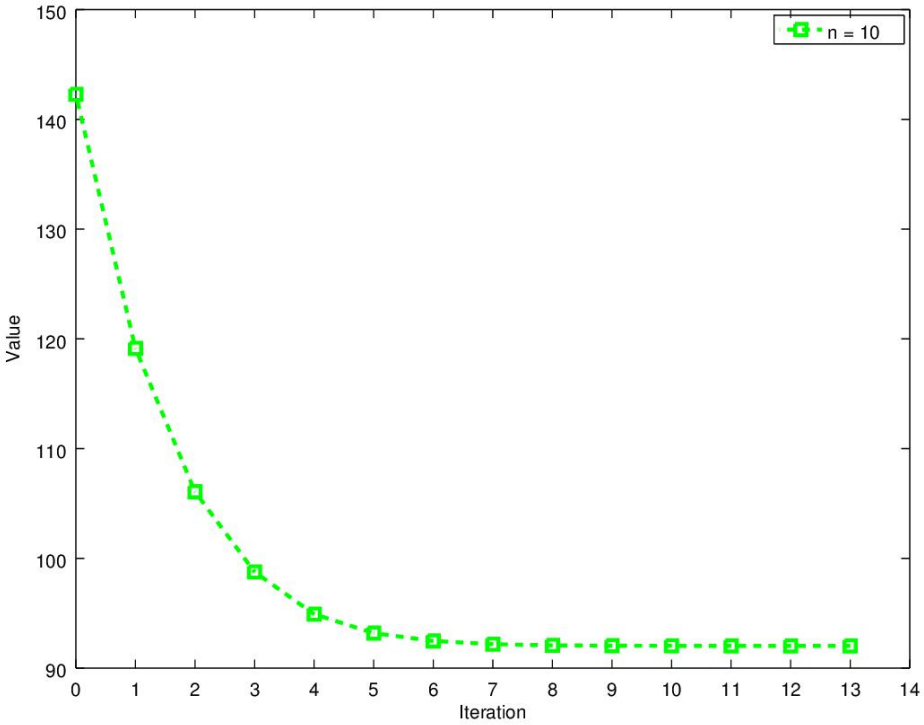


Figure 2: This figure is the same as green component in Figure 1, plotted separately to illustrate more details. Input size is 10, and $\varepsilon = 10^{-6}$ is used for stopping criterion. As shown, dual scaling algorithm needed 13 iterations before meeting stopping criterion ($gap < \varepsilon$). As expected, in each iteration we get closer to the optimal value. During first few iterations, we face dramatic drop in approximation for global optimal, while after 5^{th} iteration the value changes slowly until stopping criterion is met. Two other cases have the same behavior.

8.2 Dual Scaling vs Bruteforce

We can compare results of the discussed method with bruteforce in two different aspects, *time* and *accuracy*. We will provide comparisons for each case, based on random graphs[18].

8.2.1 Time

Table 1 shows time comparison between proposed method and bruteforce for finding optimal value (approximation[19] in SDP case) for graphs with 10 to 17 vertices. As expected, with size of graph increasing, required time for backtrack algorithm increases dramatically, while time for dual scaling method changes negligibly. In Figure 3, we can see the visual graphics for Table 1.

[18] All edge weights are rational numbers in range $(1, 5)$

[19] Upperbound

Input Size	10	11	12	13	14	15	16	17
BruteForce	0.133	0.257	0.485	2.083	8.756	34.528	151.470	568.820
Dual Scailing	0.034	0.039	0.043	0.051	0.085	0.072	0.076	0.084

Table 1: This table compares time required by bruteforce and dual scaling algorithm, for graphs of various size. Time is measured in seconds. As we can see, dual scaling is much more time efficient. One may notice, in case of dual scaling algorithm, time for input size 15 is less than time for input size 14. It is worthy of mention, that convergence rate of this algorithm in addition to input size, depends on various parameters including choice of initial point, stopping criterion, step size, etc.

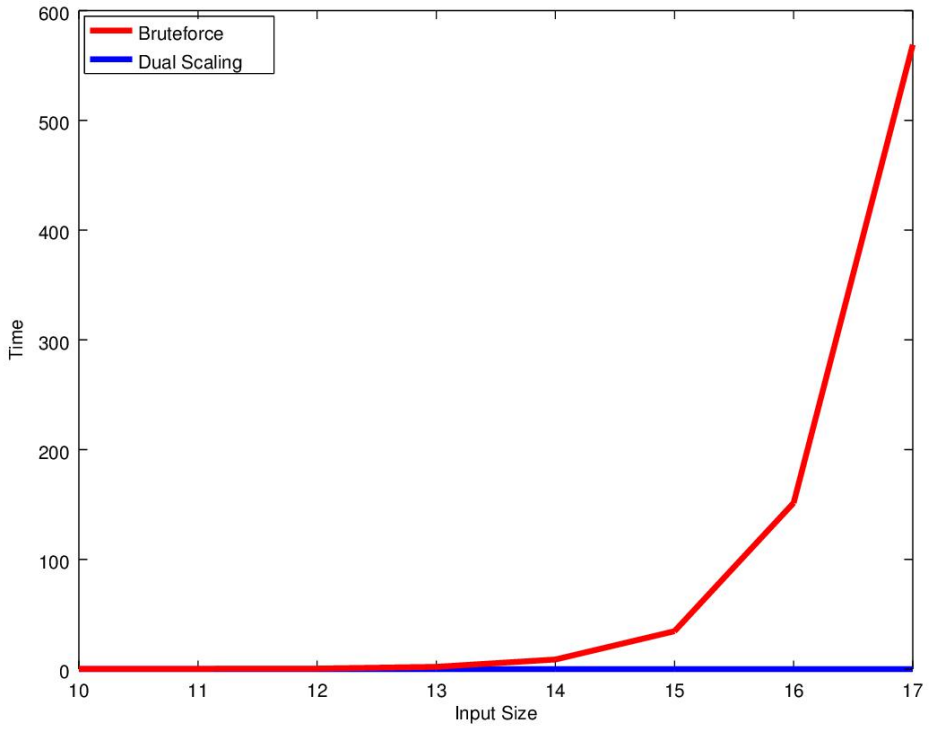


Figure 3: Red line shows time growth for bruteforce algorithm regarding input size. As expected it grows exponentially. On the other hand, blue line clearly shows that change of time for dual scaling algorithm is negligible.

8.2.2 Accuracy

As illustrated in Table 1 and Figure 3, Dual Scaling algorithm beats bruteforce by difference in terms of time. Now we compare accuracy of these algorithms. Obviously, bruteforce method returns global optimum with no error, while dual scaling algorithm provides an upper bound. Table 2 compares outputs of both algorithms for same graphs we used in Table 1.

Input Size	10	11	12	13	14	15	16	17
BruteForce	44.364	41.552	67.045	83.638	109.556	105.798	112.071	132.078
Dual Scailing	45.593	42.087	68.337	84.893	112.257	107.407	113.902	136.432

Table 2: This table checks the accuracy of dual scaling algorithm, for graphs with various size. As we can see, dual scaling returns a larger but very close value compared to the global optimum obtained by bruteforce. It is worthy of mention, that in cases with larger edge weights (For instance around 100) the margin is also small.

References

[1] Pablo A. Parrilo. *Algebraic Techniques and Semidefinite Optimization, 2006*. MIT

[2] Stephen Boyd, Lieven Vandenberghe, *Convex Optimization, 2004*. Cambridge University Press

[3] Julien Hess. *Eigenvalue Optimization for Solving the MAX-CUT Problem, 2012*. Ecole Polytechnique Fdrale de Lausanne

[4] Steven J. Benson, Yinyu Ye, Xiong Zhang, *Mixed Linear and Semidefinite Programming for Combinatorial and Quadratic Optimization, 1999*.

[5] Steven J. Benson, Yinyu Ye, Xiong Zhang, *Solving Large-Scale Sparse Semidefinite Programs for Combinatorial Optimization, 2000*. SIAM, Vol. 10, No. 2, pp. 443-461

[6] E. de Klerk, D.V. Pasechnik, *Solving SDPs in Non-Commutative Algebras, Part I: The Dual-Scaling Algorithm, 2005*