Advanced Algorithms - Weighted MAXCUT

Lu Dai (4506677)
Yuxuan Kang (4504410)

16 Jan 2017

1 Problem Description

Weighted MAXCUT problem: Given an undirected graph $G = (V, E)$ with non-negative weights w_{ij} for edge between node v_i and v_j. The weighted MAXCUT problem is to determine the max weight cut in Graph G. A cut of a graph is a partitioning of the vertices $V = \{S, \overline{S}\}$, the max weight is the sum of the weights of all edges between the partition S and $\overline{S}$.

2 Methodology

2.1 IP model of MAXCUT

MAXCUT can be written in to integer programming form:

$$x_e = \begin{cases} 1 & \text{if } e \in (S, \overline{S}) \\ 0 & \text{Otherwise} \end{cases}$$

$$max \sum_{e \in E} w_e x_e$$

$$s.t. \sum_{e \in F} x_e + \sum_{e \in C \setminus F} (1 - x_e) \leq |C| - 1 \quad \forall \text{ cycles} \subseteq \text{E}, \forall F \subseteq C \text{ with } |F| \text{ odd}$$

The constraint states that for any cycle in the graph, the cut must intersect in an even number of edges.

2.2 Semidefinite programming

In order to solve MAXCUT more efficiently, the problem can also be translated to semidefinite programming. The quadratic integer programming formulation for MAXCUT is as follows:

$$let \;\; y_i = \begin{cases} 1 & \text{if } i \in S \\ \text{-1} & \text{if } i \in \overline{S} \end{cases}$$

$$W = max \frac{1}{2} \sum_{i<j} w_{ij}(1 - y_i y_j)$$

$$s.t. \;\; y_i \in \{-1, 1\} \;\; \text{for } i = 1, ..., m$$

So when $y_i y_j = 1$, vertex i and j are in the same partition, when $y_i y_j = -1$, they are in different partitions. The above fomulation can be written in vector form, and it is also the relaxation of the MAXCUT problem:

$$W_{RP} = max \frac{1}{2} \sum_{i<j} w_{ij}(1 - v_i \cdot v_j)$$

$$s.t. \;\; v_i \in \mathbb{R}^n \;\; ||v_i|| = 1, \;\; i = 1, ..., n$$

For feasible solutions, v_i and v_j are in one-dimensional space. Thus the above relaxation is equivalent to quadratic programming formulation of MAXCUT. If v_i and v_j are not limited to one-dimensional space but n-dimensional space, then the solution will become relaxed. Semidefinite programming of MAXCUT produces a solution which is larger than optimal solution, thus it provides an upper bound.

In order to formulate the problem in $max\ c \cdot x$ form, a Laplacian matrix is introduced. A Laplacian matrix of the graph is: $L = D - A$ where D is the degree m atrix of the graph and A is the adjacency matrix of the graph. Based on Laplacian matrix, MAXCUT problem can be tranlated into:

$$max \;\; \frac{1}{4} L \cdot x$$

$$s.t. \;\; diag(x) = e$$

$$x \in S_+^n \;\; \text{semidefinite positive matrix constraint}$$

According to above SDP, MAXCUT problem polynomially sovable.

2.3 Randomized rounding algorithm

Based on SDP, Goemans and Williamson proposed an randomized rounding algorithm which can produces a feasible solution smaller than optimal solution in MAXCUT problem, thus a lower bound can be obtained.

x obtained in SDP is semidefinite positive matrix, so it can be decomposed as $x = v^T v$. Columns in v are the vectors v_i, $i = 1, ..., n$. By generating a random vector r from gaussian distribution and normalizing it, a certain partition of the set can be obtained. Use the dot product of v_i and r:

$$\begin{cases} v_i \cdot r \geq 0 & \text{then } i \in S \\ v_i \cdot r < 0 & \text{then } i \in \overline{S} \end{cases}$$

By rounding the solution matrix x to either 1 (if non-negative) or -1 (if negative), the set can be partitioned. And the solution will be feasible by this rounding method. In order to get a tighter lower bound, the random vector r is generated multiple times, and the maximum solution will be chosen as lower bound.

3 Result and discussion

An experiment of SDP and rounding alrogithm is carried out in Matlab. There are 16 instances in the experiment:

Non-complete graphs:

Instance size	5	10	20	30	40	50	60	70
Possibility of edges	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
max weight of the edges	10	10	15	15	20	20	25	20

Complete graphs:

Instance size	5	10	20	30	40	50	60	70
Possibility of edges	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
max weight of the edges	10	10	15	15	20	20	25	20

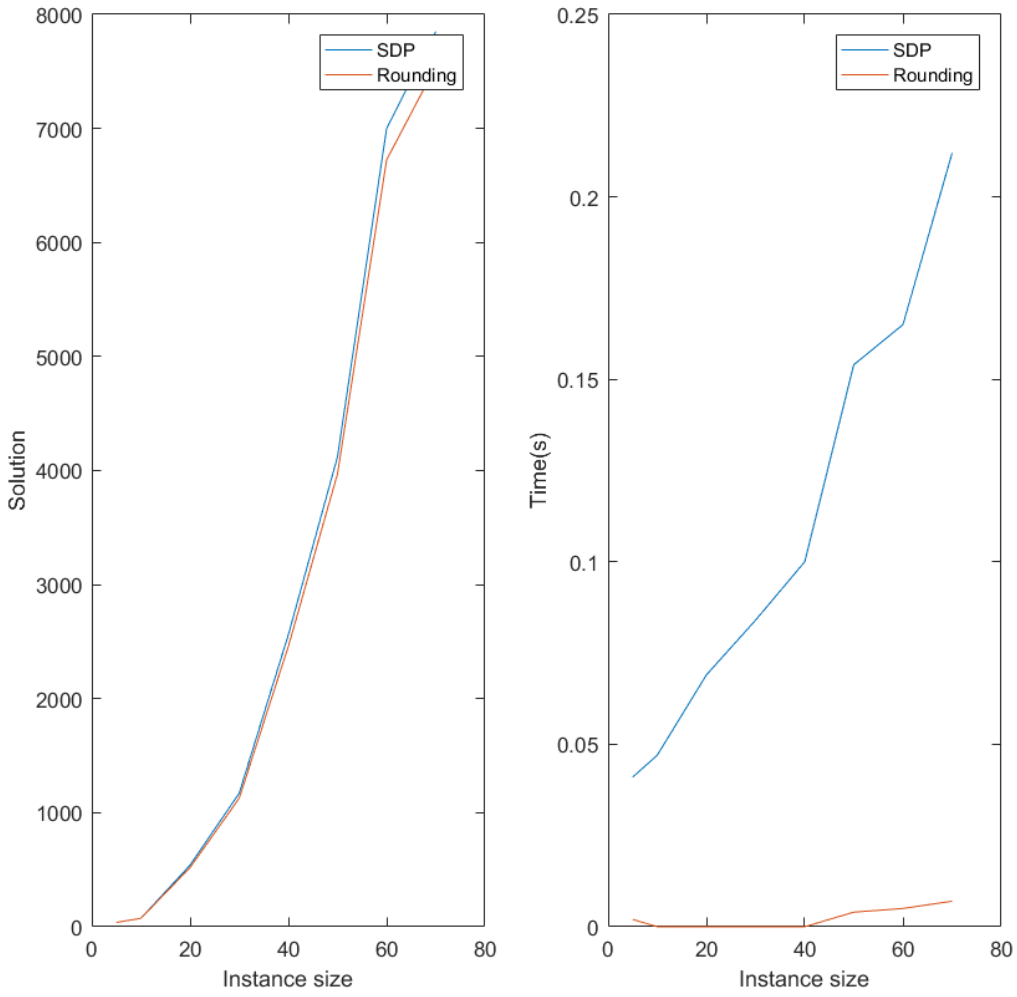

Figure 1: Solutions of non-complete graphs

Figure 1 shows the solutions and runtimes of SDP and rounding algorithm over non-complete graph. Both SDP and rounding algorithm are solvable in polynomial time as rounding takes much less time. However, since the result from rounding relies on the result of

SDP, the runtimes between SDP and rouning are not comparable. The solutions from SDP and rounding are close, so the upper bound and the lower bound are relatively tight.

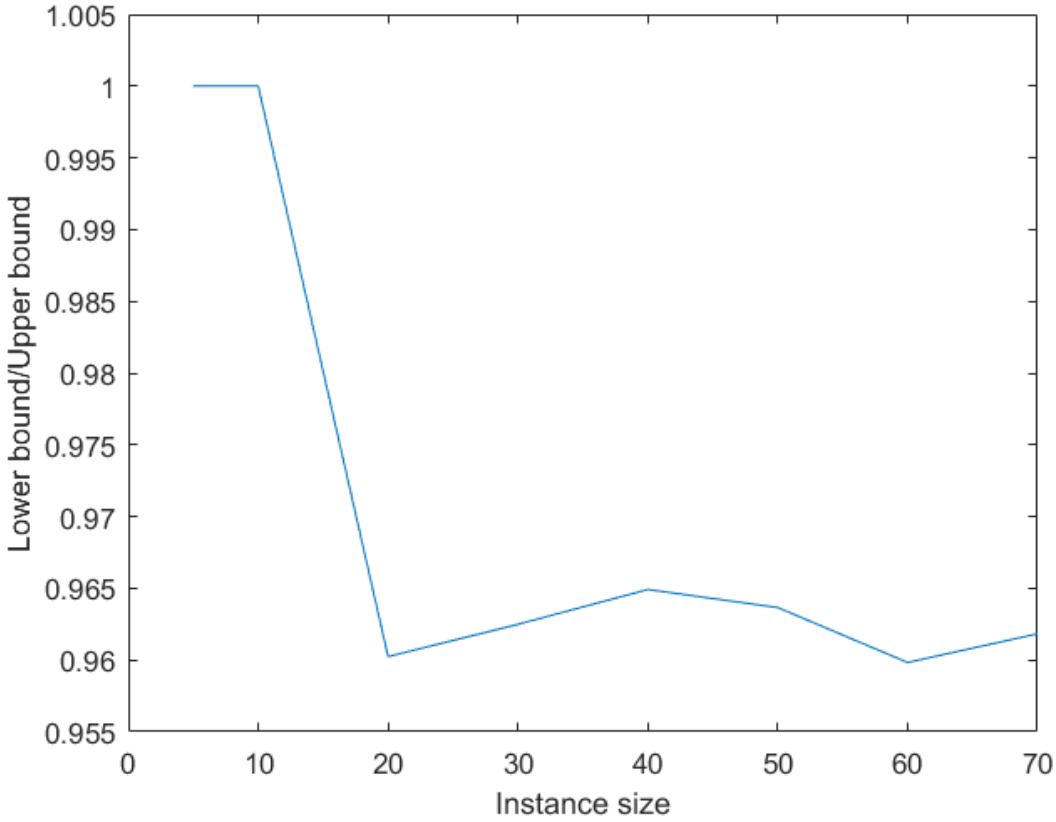

Figure 2: Accuracy of SDP and rounding

Figure 2 shows the ratio: $\frac{lowerbound}{uppderbound}$, it to some degree reflects the accuracy of SDP and rounding algorithm. For instance size of 5 and 10, the ratio equals to 1 which means both SDP and rounding algorithm provide optimal solution. The radio reflects the gap between upper bound and lower bound. The closer the ratio is to 1, the more accurate results from SDP and rounding algorithm are. In these 8 instances, the limit for the ratio is 0.96.

Similar analysis can be carried out for complete graphs. The results from 8 complete graphs are shown below:

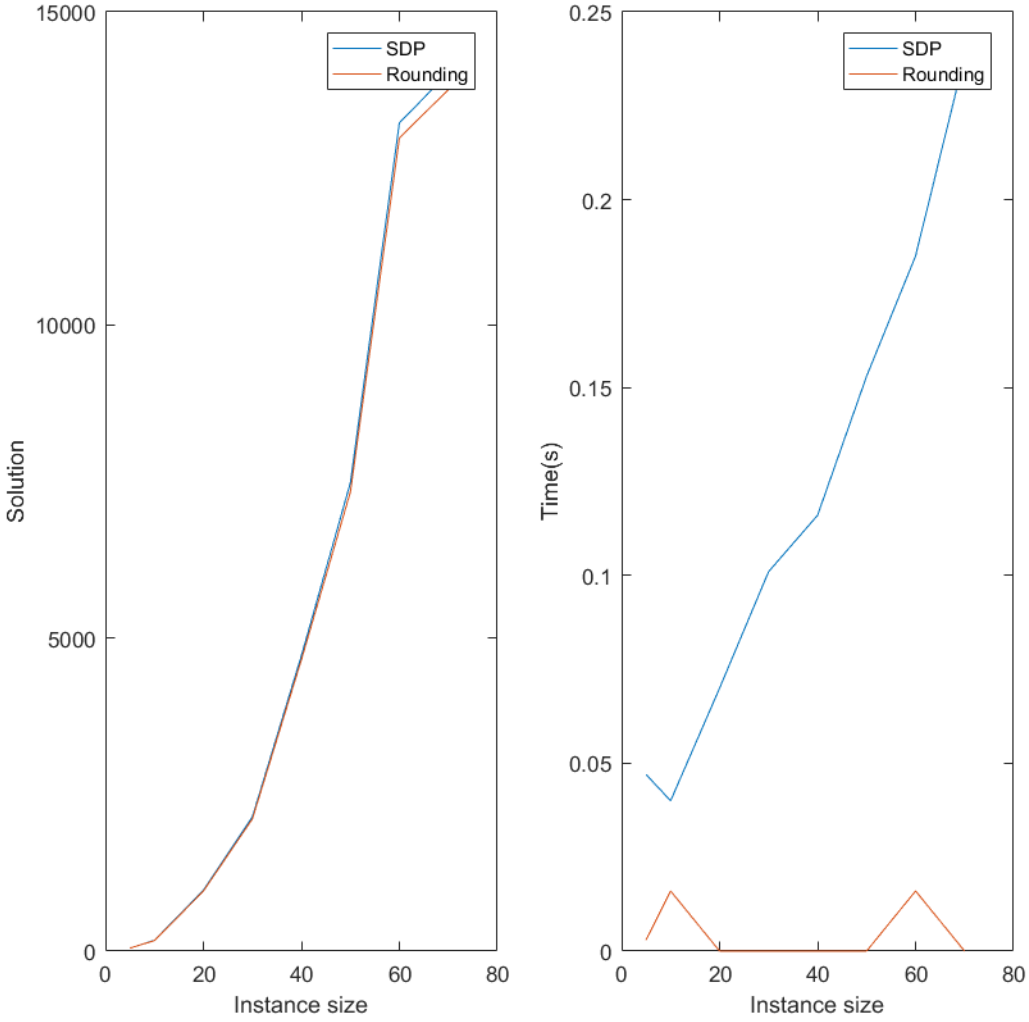

Figure 3: Solutions of non-complete graphs

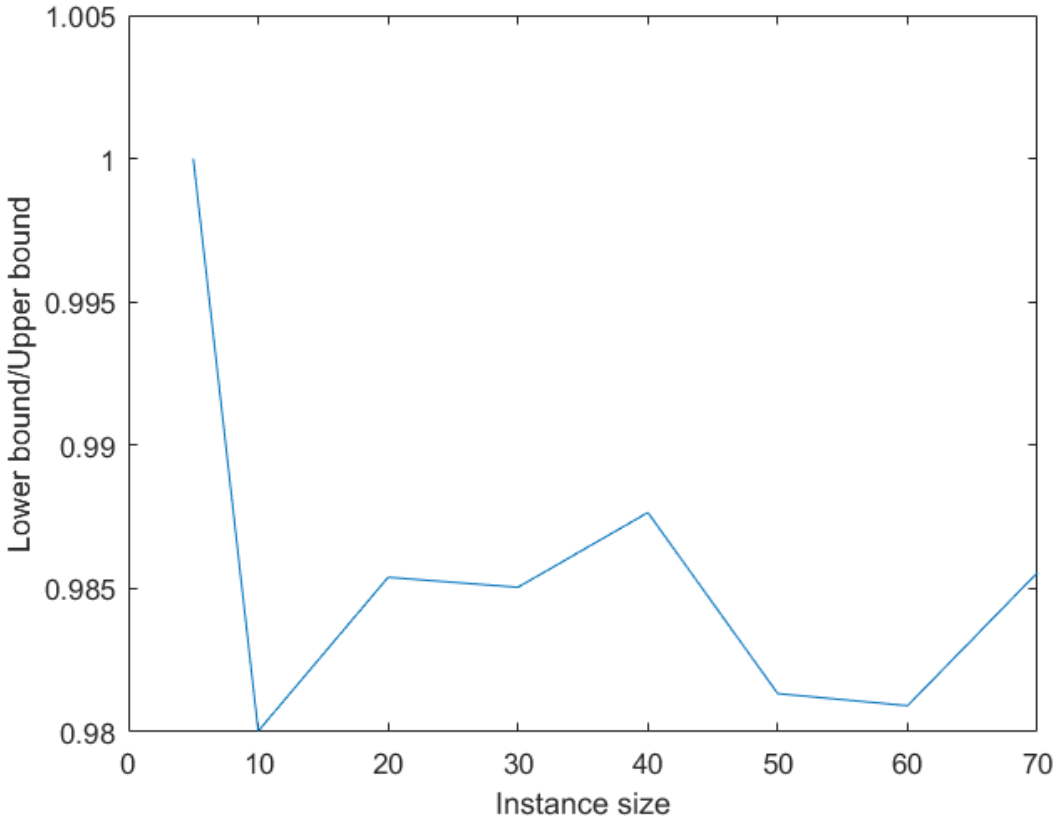

Figure 4: Solutions of non-complete graphs

Compared to non-complete graph, the runtime almost does not change, which indicates the number of the edges rarely have influence on the runtime.