# Computational Experience with a SDP-based Algorithm for Maximum Cut with Limited Unbalance

**Giulia Galbiati**
*Dipartimento di Informatica e Sistemistica, University of Pavia (Italy)*
*giulia.galbiati@unipv.it*
**Stefano Gualandi, Francesco Maffioli**
*Dipartimento di Elettronica e Informazione, Politecnico di Milano (Italy)*
*gualandi@elet.polimi.it, maffioli@elet.polimi.it*

**Keywords:** network design, randomized algorithm, approximation algorithm, semidefinite programming, maximum cut

## 1. Introduction

We address the following problem: given an undirected graph $G = (V, E)$, with vertex set $V$ of cardinality $n$ and edge set $E$, where each edge $(i, j)$ has a non-negative integer weight $w_{ij}$, and given a constant $B$, $0 \leq B < n$, find a cut $(S, V \setminus S)$ of $G$ of maximum weight such that the difference between the cardinalities of the two shores of the cut is not greater than $B$. This problem has been introduced in [4] with the name Maximum Cut with Limited Unbalance (MaxCUT-LU for short). When $B$ is equal to zero it is known as the Max Bisection problem, whereas when $B$ is equal to $n - 1$ it is the well-known Maximum Cut problem.

In [4] approximation algorithms with non-trivial performance guarantees for MaxCUT-LU have been proposed and analyzed. When $B$ is equal to zero the approximation ratio guaranteed in [4] coincides with the one of [8] for Max Bisection, which is equal to 0.699; otherwise it is always greater than this value, and, when $B$ approaches the number $n$ of nodes, it tends to the approximation ratio 0.87856 of the algorithm of [5] for Maximum Cut. The results in [4] have been obtained by extending to MaxCUT-LU the methodology used in [8] and by combining this technique with another one that uses the algorithm of [5].

In this paper we present extensive computational experience obtained by applying the algorithms in [4] to a large set of graphs. The types of graphs considered are:

1) random unweighted graphs generated using the graph generator `rudy` reported in [6];
2) random graphs generated using `rudy`, but with weights $w_{i,j}$ generated randomly in the set of integers $\{i, \ldots, j\}$, to try to force maximum cuts to be unbalanced;
3) graphs from circuit design problems [3].

It turns out that the approximation ratios obtained on these graphs are, in practice, always much better than those theoretically guaranteed in [4].

In [7] several applications of Maximum Cut are reported in different fields such as network planning, circuit design, scheduling, logic, psychology. For most of them we think that the generalization to MaxCUT-LU makes sense. For instance, in circuit design, the problem of dividing the vertex set of the graph underlined by the circuit into two parts of equal cardinalities is of interest, and relaxing the equal cardinality constraint to that of limited unbalance can allow to get better results as far as approximating the optimum weight of the cut obtained, without affecting the suitability of the partition from the point of view of the circuit designer.

In Section 2 MaxCUT-LU is formulated as a semidefinite programming (SDP) problem; Section 3 describes the algorithm that we experiment and specifies some implementation details; Section 4 reports and analyzes the obtained computational experience.

## 2. The Formulation

MaxCUT-LU can be formulated by assigning to each vertex $i$ a binary variable $x_i \in \{-1, 1\}$, with vertices on the same shore of the cut receiving the same value, and by setting $w_{ij} = 0$ if $(i, j) \notin E$, as:

$$w^* := \max \left\{ \frac{1}{4} \sum_{i,j} w_{ij}(1 - x_i x_j) : \sum_{i,j} x_i x_j \leq B^2; x_i \in \{-1, 1\}, i = 1, \ldots, n \right\}. \tag{1}$$

The semidefinite relaxation of this binary quadratic program can be formulated as follows:

$$w^{SDP} := \max \left\{ \frac{1}{4} \sum_{i,j} w_{ij}(1 - X_{ij}) : \sum_{i,j} X_{ij} \leq B^2; X_{ii} = 1, i = 1, .., n; \mathbf{X} \in M_n \right\} \tag{2}$$

where $M_n$ is the set of real, symmetric, positive semidefinite matrices of order $n$. It is easy to see that any solution $\mathbf{x}$ of (1) yields a solution $\mathbf{X}$ of (2) with $X_{ij} = x_i x_j$. Hence obviously $w^* \leq w^{SDP}$.

It is known (see e.g. [1]) that this type of SDP program can be solved to any degree of accuracy in polynomial time. From an almost optimal solution, one can then derive a solution of the integer program using appropriate rounding techniques. Rounding techniques applied to the solution of SDP relaxations of Combinatorial Optimization problems in order to get integral solutions of guaranteed degree of approximation have been pioneered by Goemans and Williamson [5] for the MAX CUT and MAX SAT problems. In the algorithm presented in Section 3 we use as rounding technique a generalization of that of [8], which refines the one in [5].

Our computational experiments have been done using only the first of the two algorithms in [4]. This algorithm is the most innovative of the two and deals with instances having the ratio $\eta = B/n$ not greater than 0.8. In practice, for the set of instances under test, it turns out that the solutions obtained by this algorithm are already insensitive to higher values of $\eta$, when $\eta$ is well below 0.8, therefore vanishing the need to use the second algorithm of [4].

## 3. The Algorithm

We now present the algorithm that we use in our computational experiments. In the algorithm, $I$ indicates the identity matrix, the parameter $\theta$ and $k$ are fixed in an appropriate way, as specified in [4]. The algorithm uses the following rounding technique: from a solution $\widetilde{X}$ of the SDP relaxation first it constructs a new matrix $X$ as a convex combination of $\widetilde{X}$ and the identity matrix $I$; then to matrix $X$, which is positive definite, it applies the Cholesky decomposition to obtain vectors $(\mathbf{v_1}, ..., \mathbf{v_n})$ on the unit $n$-dimensional sphere $S_n$. The algorithm then uses the so called random hyperplane technique, i.e. it repeatedly generates a uniformly distributed vector $\mathbf{r}$ on the unit sphere, computes vector $\mathbf{u} = (\mathbf{r} \cdot \mathbf{v_1}, ..., \mathbf{r} \cdot \mathbf{v_n})$ and then rounds $\mathbf{u}$ to a vector $\widehat{\mathbf{x}}$ with $\widehat{x}_i \in \{-1, 1\}$, and $\widehat{x}_i = -1$ iff $u_i \geq 0$, $i = 1, ..., n$. Each vector $\widehat{\mathbf{x}}$ hence identifies a cut $(S, V \backslash S)$ of $G$, where $S = \{i : \widehat{x}_i = 1\}$ or $S = \{i : \widehat{x}_i = -1\}$; in our algorithm w.l.o.g. we always choose $S$ to be the set of vertices with the larger cardinality. Finally function $rebalance(S)$, when invoked by the algorithm, moves the nodes which least contribute to the weight of the cut from set $S$ to the other set of the cut, so as to reduce the number of nodes in $S$ to $(n + B)/2$.

Throughout this paper, $w(S)$ denotes the weight of the cut $(S, V \backslash S)$. The analysis of this algorithm, its correctness and performance guarantee have been developed in [4].

**Algorithm.**

*1 - Solve the SDP problem (2) and let $\widetilde{X}$ be the solution matrix;*

*2 - fix a value $\theta$ with $0 \leq \theta < 1$ and a positive integer $k$;*

*3 - let $X = \theta\widetilde{X} + (1-\theta)I$;*

*4 - apply Cholesky decomposition to $X$ to obtain vectors $(\mathbf{v_1}, ..., \mathbf{v_n})$;*

*5 - $S_R = \phi$;*

*6 - repeat for $k$ times the following* {

*6.1 - generate a uniformly distributed vector* $\mathbf{r}$ *on the unit sphere;*

*6.2 - compute* $\mathbf{u} = (\mathbf{r} \cdot \mathbf{v_1}, ..., \mathbf{r} \cdot \mathbf{v_n})$;

*6.3 - round* $\mathbf{u}$ *to vector* $\widehat{\mathbf{x}} \in \{-1, 1\}^n$ *identifying a cut* $(S, V \backslash S)$;

*6.4 - if* $|S| \leq (n+B)/2$ */* the cut is feasible for MaxCUT-LU */*

*let* $\widetilde{S} = S$ *else let* $\widetilde{S} = rebalance(S)$;

*6.5 - if* $w(\widetilde{S}) > w(S_R)$ */* a better cut for MaxCUT-LU is found */*

*let* $S_R = \widetilde{S}$;}

*7 - return* $S_R$.

The values of $\theta$ used in our experiments, for each specific value of $\eta$, are drawn from [4] and reported in the table below; for the sake of simplicity the value of $k$ has been fixed to $10^4$.

| $\eta$ | 0.0000 | 0.0500 | 0.1000 | 0.1050 | 0.1065 |
|---|---|---|---|---|---|
| $\theta$ | 0.888 | 0.890 | 0.894 | 0.895 | 0.895 |
| $\eta$ | 0.1065 | 0.2000 | 0.3333 | 0.4000 | 0.4500 |
| $\theta$ | 0.893 | 0.941 | 0.966 | 0.972 | 0.975 |
| $\eta$ | 0.4930 | 0.5000 | 0.6000 | 0.7000 | 0.8000 |
| $\theta$ | 0.977 | 0.977 | 0.980 | 0.982 | 0.984 |

## 3..1 Implementation Details

The algorithm is implemented in C/C++. The following is a detailed description of the implementation of some steps of the algorithm:

Step 1: the semidefinite relaxation is solved using the DSDP solver [2], which implements a dual-scaling interior-point algorithm. Since the DSDP solver accepts only equality constraints, a *slack* variable is introduced for the limited unbalance constraint. Instead of using a matrix $\widetilde{X}$ of $n \times n$ elements, we introduce a matrix $\bar{\mathbf{X}}$ of $(n+1) \times (n+1)$ elements and use the following SDP relaxation:

$$\max \left\{ \frac{1}{4} \sum_{i,j} w_{i,j}(1 - \bar{X}_{i,j}) : \sum_{i,j} \bar{X}_{i,j} = B^2; \bar{X}_{i,i} = 1, \bar{X}_{(n+1),i} = 0, \forall i \in \{1, \ldots, n\}; \bar{\mathbf{X}} \in M_{n+1} \right\}$$

where the weights $w_{(n+1),i} = w_{i,(n+1)} = 0, \forall i \in \{1, \ldots, n+1\}$. The element $\bar{X}_{(n+1),(n+1)}$ is our positive slack variable. The parameters used in the DSDP solver are: (i) the initial dual infeasibility $r_0 = B$, (ii) the potential parameter $\rho = 10$. All other parameters are used with their default values as described in [2]. A solution matrix $\widetilde{X}$ of the SDP relaxation is obtained by dropping from $\bar{\mathbf{X}}$ the last row and the last column corresponding to the *slack* variable.

Step 4: to perform the Cholesky decomposition on the positive definite matrix $\mathbf{X} = \theta\widetilde{\mathbf{X}} + (1-\theta)\mathbf{I}$, we use a fast implementation of the LAPACK library (function `dpptrf`).

Step 5: to store the solution computed at each step of the loop, we use a hash set data structure, since it has constant time average complexity for storage and retrieval.

Step 6: for this step there are two remarks: first, to compute the vector $\mathbf{u}$ we use a fast implementation of the BLAS library (function `dtpmv`). Second, to implement the procedure `rebalance(S)` we do the following: while computing the cost of the cut, we iterate over the nodes in the shore $S$ and sum up the contribution given by those nodes. The contribution of a node is equal to the sum of the weights of its incident edges belonging to the cut; the contributions of the nodes in $S$ are stored in a vector. Then, if the balance constraint is violated, i.e. $|S| > \frac{n+B}{2}$, we perform a partial sort on this vector using the node contribution as increasing sort criterion. After partial sorting, the first $|S| - \frac{n+B}{2}$ elements of the contribution vector identify the nodes which we move from $S$ to the other shore. Notice that the partial sort has an amortized linear complexity on the size of the vector, while all other operations have worst-case linear complexity on the number of edges.

The implementation was tested on a Sun-Fire V-240 running Solaris 5.8 with the Sun CC compiler and the Sun Performance Library implementation of BLAS and LAPACK. On both platforms, we use the DSDP solver version 5.8 running on a single processor.

## 4. Computational Results

Tables 1, 2, 3 show some of our computational results for, respectively, the graph $G_{23}$ generated with `rudy` [6], a random graph with 800 nodes and weights $w_{i,j}$ generated randomly in the set of integers $\{i, \ldots, j\}$, and a circuit design graph with 915 nodes. The first two columns give two parameters used by the algorithm: the unbalance parameter $\eta$ and the corresponding $B$. The third and fourth columns give the optimal solution value $w^{SDP}$ of the SDP relaxation, and the corresponding solution time $t^{SDP}$ in seconds. The next three columns give, respectively, the value $w^{rand}$ of the cut returned by our randomized algorithm, the solution time $t^{rand}$ in seconds and the cardinality of the bigger shore $S_R$. Finally, the last two columns give the theoretical approximation ratio $r$ presented in [4] and the experienced ratio $\tilde{r} = \frac{w^{rand}}{w^{SDP}}$, which is a pessimistic estimate of the unknown exact approximation ratio $\frac{w^{rand}}{w^*}$.

Table 1 concerns the performance of our algorithm on graph $G_{23}$ generated with `rudy`. The behaviour exhibited on $G_{23}$ is similar to that for the other uniformly weighted graphs generated with `rudy`. It turns out that for these graphs the maximum cut has the two shores of roughly the same size, that is, the unbalance is very small, so that adding the limited unbalance constraint has little effect. Table 1 also shows that for graphs with 2000 nodes the computation time is already around 20 minutes.

Table 2 gives the results for a weighted random graph. The fourth column shows that, while increasing the unbalance parameter $\eta$, the values $w^{SDP}$ and $w^{rand}$, and the cardinality of the bigger shore $|S_R|$ increase too, but only until $\eta = 0.4$. In addition we remark how, also for the small increase of $\eta$ from $0.1$ to $0.105$, the algorithm is able to slightly increase the objective value $w^{rand}$ at the cost of a bigger $B = 442$ instead of $B = 440$.

In Table 3, we observe that the results are similar to the previous table. In the last three lines $w^{SDP} = w^{rand}$ hence implying that we have an optimum solution with $B = 263$. Therefore for values of $\eta > 0.33$ the

| Parameters | | SDP | | Randomized | | | Ratio | |
|---|---|---|---|---|---|---|---|---|
| $\eta$ | B | $w^{SDP}$ | $t^{SDP}$ | $w^{rand}$ | $t^{rand}$ | $\|S_R\|$ | $r$ | $\tilde{r}$ |
| 0.000 | 1 | 14145.40 | 1185.93 | 12858 | 603.42 | 1000 | 0.699 | 0.909 |
| 0.050 | 100 | 14145.51 | 1053.62 | 12836 | 677.27 | 1009 | 0.731 | 0.907 |
| 0.100 | 200 | 14145.51 | 1098.58 | 12851 | 684.98 | 1008 | 0.759 | 0.908 |
| 0.105 | 210 | 14145.51 | 1049.33 | 12844 | 688.02 | 1010 | 0.761 | 0.908 |
| 0.200 | 400 | 14145.51 | 1024.35 | 12891 | 684.95 | 1000 | 0.788 | 0.911 |
| 0.333 | 660 | 14145.51 | 1141.46 | 12930 | 679.52 | 1005 | 0.797 | 0.914 |
| 0.400 | 800 | 14145.51 | 1077.82 | 12948 | 681.66 | 1017 | 0.798 | 0.915 |
| 0.500 | 1000 | 14145.51 | 1302.02 | 12943 | 684.58 | 1007 | 0.798 | 0.915 |

Table 1: Graph $G_{23}$: $n = 2000$, $e = 19990$, unitary weights.

| Parameters | | SDP | | Randomized | | | Ratio | |
|---|---|---|---|---|---|---|---|---|
| $\eta$ | B | $w^{SDP}$ | $t^{SDP}$ | $w^{rand}$ | $t^{rand}$ | $\|S_R\|$ | $r$ | $\tilde{r}$ |
| 0.000 | 1 | 1.80071e+07 | 87.93 | 1.74253e+07 | 227.07 | 400 | 0.699 | 0.967 |
| 0.050 | 40 | 1.80350e+07 | 71.80 | 1.74349e+07 | 226.13 | 420 | 0.731 | 0.967 |
| 0.100 | 80 | 1.80542e+07 | 232.32 | 1.75018e+07 | 229.89 | 440 | 0.759 | 0.968 |
| 0.105 | 84 | 1.80548e+07 | 203.17 | 1.75047e+07 | 229.55 | 442 | 0.761 | 0.968 |
| 0.200 | 160 | 1.80550e+07 | 75.59 | 1.75656e+07 | 229.30 | 458 | 0.788 | 0.972 |
| 0.333 | 264 | 1.80550e+07 | 85.44 | 1.75705e+07 | 229.86 | 461 | 0.797 | 0.973 |
| 0.400 | 320 | 1.80550e+07 | 89.56 | 1.75758e+07 | 229.13 | 454 | 0.798 | 0.973 |
| 0.500 | 400 | 1.80550e+07 | 97.84 | 1.75793e+07 | 226.60 | 454 | 0.798 | 0.973 |

Table 2: Weighted graph with $n = 800$ nodes, $e = 79900$ edges, and integer weights $w_{i,j} = w_{j,i}$ chosen randomly in the set $\{i, \ldots, j\}$.

| Parameters | | SDP | | Randomized | | | Ratio | |
|---|---|---|---|---|---|---|---|---|
| $\eta$ | B | $w^{SDP}$ | $t^{SDP}$ | $w^{rand}$ | $t^{rand}$ | $\|S_R\|$ | $r$ | $\tilde{r}$ |
| 0.000 | 1 | 1842.25 | 300.94 | 1823 | 143.98 | 458 | 0.699 | 0.989 |
| 0.050 | 46 | 1864.52 | 277.97 | 1845 | 142.55 | 480 | 0.731 | 0.989 |
| 0.100 | 92 | 1885.82 | 306.30 | 1868 | 144.61 | 503 | 0.759 | 0.990 |
| 0.105 | 96 | 1887.60 | 315.44 | 1870 | 141.37 | 505 | 0.761 | 0.990 |
| 0.200 | 183 | 1924.06 | 425.86 | 1914 | 144.34 | 549 | 0.788 | 0.994 |
| 0.333 | 302 | 1954.00 | 380.80 | 1954 | 143.23 | 589 | 0.797 | 1.000 |
| 0.400 | 366 | 1954.00 | 376.48 | 1954 | 142.07 | 589 | 0.798 | 1.000 |
| 0.500 | 458 | 1954.00 | 371.76 | 1954 | 143.69 | 589 | 0.798 | 1.000 |

Table 3: Circuit design graph with $n = 915$ nodes, $e = 1954$ edges, and unitary weights.

| Graphs | $\|N\|$ | $\|E\|$ | $t^{SDP}$ | std-dev | $t^{rand}$ | std-dev |
|---|---|---|---|---|---|---|
| $G_{14}, G_{15}, G_{16}, G_{17}$ | 800 | 4672 | 225 | 116,00% | 106 | 7,11% |
| $G_1, G_2, G_3, G_4, G_5$ | 800 | 19176 | 84 | 15,98% | 131 | 6,76% |
| $G_{51}, G_{52}, G_{53}, G_{54}$ | 1000 | 5900 | 183 | 27,57% | 165 | 6,97% |
| $G_{43}, G_{44}, G_{45}, G_{46}, G_{47}$ | 1000 | 9990 | 189 | 47,08% | 170 | 7,50% |
| $G_{35}, G_{36}, G_{37}, G_{38}$ | 2000 | 11700 | 1281 | 8,16% | 608 | 6,74% |
| $G_{22}, G_{23}, G_{24}, G_{25}, G_{26}$ | 2000 | 19990 | 1012 | 9,92% | 624 | 7,49% |

Table 4: `rudy` generated graphs with unitary weights

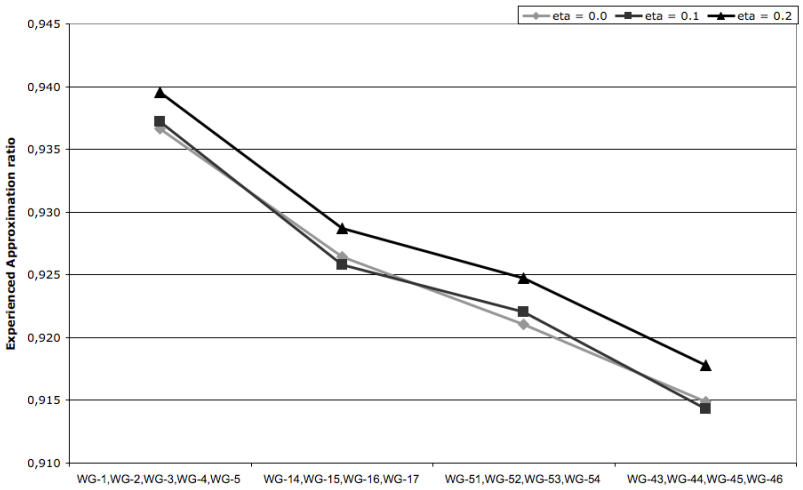


Figure 1: Average approximation ratios for weighted random graphs

unbalance constraint plays no role. Notice also that the pessimistic ratio $\tilde{r}$ is exceeding by far the theoretical worst case ratio $r$.

In Table 4 the average computation times $t^{SDP}$ and $t^{rand}$, given respectively by step 1 and the remaining steps of the algorithm, are reported for all graphs generated with `rudy`, together with the corresponding standard deviations.

Finally Figure 1 reports, for the random graphs of the first four lines of Table 4, with edges randomly weighted as indicated in Table 2, the average approximation ratios obtained for values of $\eta$ equal to 0, 0.1 and 0.2.

## References

[1] Alizadeh, F.: Interior point methods in semidefinite programming with applications to combinatorial optimization. SIAM J. Optimization **5** (1995) 13–51

[2] Benson, S. J., Ye, Y.: DSDP5: Software For Semidefinite Programming. Submitted to ACM Transactions on Mathematical Software

[3] Brambilla, A.: private communication

[4] Galbiati, G., Maffioli, F.: Approximating Maximum Cut with Limited Unbalance In: Erlebach, T., Kaklamanis, C. (eds.): Fourth Workshop on Approximation and Online Algorithms, WAOA 2006 Lecture Notes in Computer Science, Springer-Verlag, Berlin Heidelberg New York, to appear

[5] Goemans, M.X., Williamson, D.P.: Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of ACM **42** (1995) 1115–1145

[6] Helmberg, C..: Semidefinite Programming website.
`http://www-user.tu-chemnitz.de/ helmberg/semidef.html`

[7] Poljak, S., Tuza, Z.: Maximum cuts and large bipartite subgraphs. In: Cook, W., Lovasz, L., Seymour, P.(Eds.): Combinatorial Optimization. AMS - DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 20. American Mathematical Society, Providence, RI (1995) 181–244

[8] Ye, Y.: A .699-approximation algorithm for Max-Bisection. Math. Programming Ser. A (90) (2001) 101–111