An Approximation Algorithm for the Maximum Cut Problem and its Experimental Analysis ⋆

A. Bertoni, P. Campadelli and G. Grossi

Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano
via Comelico 39, 20135 Milano, Italy

Abstract

An approximation algorithm for the Maximum Cut Problem is designed and analyzed; its performance is experimentally compared with that of a neural algorithm and that of Goemans and Williamson's algorithm.

Although the guaranteed quality of our algorithm in the worst case analysis is poor, we give experimental evidence that its average behavior is better than that of Goemans and Williamson's algorithm.

Key words: Optimization, Complexity, Relaxation, Local Search, Goemans and Williamson's algorithm.

1 Introduction

The Max Cut problem is the problem of partitioning the vertex set of an undirected graph into two parts in order to maximize the cardinality of the set of edges cut by the partition. This problem has long been known to be NP–hard, it is solvable in polynomial time only for some special classes of graphs [5]. Because of its theoretical and practical importance and because efficient algorithms for NP–hard combinatorial optimization problems are unlikely to exist, many polynomial time approximation algorithms have been proposed to solve it. Among these we consider a simple neural algorithm that guarantees to find a solution of value at least 0.5 time the optimal solution, and the one designed

⋆ Partially supported by MURST project: *Modelli di calcolo innovativi: metodi sintattici e combinatori.*

by Goemans and Williamson [4], that guarantees to find a solution of measure at least 0.878 time the optimal one. Although extremely interesting because it has one of the best worst case performance, Goemans and Williamson's algorithm is of complex design and its computation time may be prohibitive on large problem instances (graphs with more than 1000 vertices). For this reason we present here a very simple algorithm, called LORENA, which is inspired by Goemans and Williamson's main idea. We estimate the computation time of LORENA, and we find that it is comparable with that of the neural algorithm. As regard as the approximation quality we can only state a weak result, in fact we are only able to prove that it finds a solution of value at least 0.39 time the optimal one. However, in experimental tests on p-random graphs it behaves better than Goemans and Williamson's, and on standard benchmarks it gives the same cut values but it is faster.

The paper is organized as follows. In the next section the Max Cut problem is presented together with a brief description of the two approximation algorithms that we compare with LORENA. In Section 3 LORENA is described and an analysis of either its computation time and worst case performance is given. Section 4 presents the experimental results.

2 Preliminary Definitions and Results

In this section we briefly recall the optimization problem Max Cut and two approximation algorithms to solve it: a neural algorithm and the Goemans and Williamson's algorithm [4].

The problem Max Cut is formally defined as follows:

Max Cut

INSTANCE: Graph $G = \langle V, E \rangle$.

SOLUTION: A partition (V_1, V_2) of V into disjoint sets V_1 and V_2.

MEASURE: The cardinality of the cut, i.e., the number of edges with one end point in V_1 and one endpoint in V_2.

The Max Cut problem is one of the Karp's original NP–hard problems [10] and it is solvable in polynomial time for some special classes of graphs (e.g. planar graphs). Since efficient algorithms are unlikely to exist for NP–hard problems, a typical approach for solving them consists in finding an *α-approximation algorithm*, that is a polynomial time algorithm that delivers a solution of value at least α times the optimal one.

A simple 0.5-approximation algorithm is a neural algorithm (H_NET) based on the Hopfield's network model [8,9]. At this regard we observe that Max Cut can be equivalently formulated as the following problem of integer quadratic programming:

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2}\sum_{i<j} a_{ij}(1 - x_i x_j) \\ \text{subject to} \quad & x_i \in \{-1, 1\}, \quad i = 1, \ldots, n, \end{aligned} \tag{P}$$

where $(a_{ij})_{n\times n}$ is the graph adjacency matrix, $n = |V|$ and for each vertex $i \in V$ x_i is a boolean variable. An assignment to the variables $\{x_1, \ldots, x_n\}$ gives a partition (V_1, V_2) of V, where $i \in V_1$ if and only if $x_i = 1$. This formulation can naturally be extended to the weighted case. The neural algorithm simulates an Hopfield's network that locally minimizes the energy function $-1/2 \sum_{i<j} a_{ij}(1 - x_i x_j)$. Given an arbitrary initial state the network reaches, in its dynamical evolution, an equilibrium point $(\tilde{x}_1, \ldots, \tilde{x}_n)$ which is interpreted as a solution for Max Cut by considering the partition $(V_1, V \backslash V_1)$, where $V_1 = \{k \mid \tilde{x}_k = 1\}$. Essentially, this algorithm implements a local search in the hypercube; it has reasonable performances relative to either the computation time and the approximation quality. Moreover its simplicity makes it easy to design a hardware implementation [1].

A better result from the point of view of the worst case analysis is obtained by Goemans and Williamson's algorithm. Instead of solving (P), they considered the following relaxed problem:

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2}\sum_{i<j} a_{ij}(1 - \mathbf{v}_i \cdot \mathbf{v}_j) \\ \text{subject to} \quad & \mathbf{v}_i \in \mathbf{S}_n, \quad i = 1, \ldots, n, \end{aligned} \tag{R}$$

where $\mathbf{v}_i$ is a vector in the n-dimensional unit sphere $\mathbf{S}_n = \{\mathbf{x} \in \mathbb{R}^n : \|x\| = 1\}$, and $\mathbf{v}_i \cdot \mathbf{v}_j$ denotes the inner product of $\mathbf{v}_i$ and $\mathbf{v}_j$. Given an arbitrary hyperplane $\mathbf{r}$, let A_1 and A_2 be the halfspaces generated by $\mathbf{r}$; a partition (V_1, V_2) of V can be obtained by the vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ by setting $i \in V_1$ if and only if $\mathbf{v}_i \in A_1$.

Let $\varepsilon > 0$ be any fixed scalar. To find an optimal solution of (R) up to ε in polynomial time, Goemans and Williamson reformulated (R) as a semidefinite program and solve it using a variation of the interior point method for linear programming [2,7].

Let $Y = (y_{ij})$ where $y_{ij} = \mathbf{v}_i \cdot \mathbf{v}_j$. Then

(1) $\|\mathbf{v}\| = 1$ implies $y_{ii} = 1$ for all i.

(2) $y_{ij} = \mathbf{v}_i \cdot \mathbf{v}_j$ implies $Y \succeq 0$, i.e., Y is positive semi-definite ($\forall \mathbf{x} \in \mathbb{R}^n$: $\mathbf{x}^T Y \mathbf{x} \geq 0$). This is true because:

$$\mathbf{x}^T Y \mathbf{x} = \sum_i \sum_j x_i x_j (\mathbf{v}_i \cdot \mathbf{v}_j) = \left\| \sum_i x_i \mathbf{v}_i \right\|^2 \geq 0.$$

Conversely, if $Y \succeq 0$ and $y_{ii} = 1$ for all i then it can be shown that there exists a set of $\mathbf{v}_i$'s such that $y_{ij} = \mathbf{v}_i \cdot \mathbf{v}_j$.

Hence (R) is equivalent to

$$\begin{aligned} &\textsf{maximize} \quad \frac{1}{2} \sum_{\{i,j\} \in E} (1 - y_{ij}) \\ &\textsf{subject to} \quad \begin{cases} y_{ii} = 1, & i = 1, \ldots, n \\ Y \succeq 0. \end{cases} \end{aligned} \tag{R'}$$

Goemans and Williamson's algorithm (RR_SDP) consists then of two main steps:

RR_SDP

Input: The program (R');

STEP 1 Solve (R') to get an optimum solution $\{\mathbf{v}_1^*, \ldots, \mathbf{v}_n^*\}$;

STEP 2 Take a random vector $\mathbf{r}$ uniformly distributed on $\mathbf{S}_n$ and set $V_1 = \{k \mid \mathbf{v}_k^* \cdot \mathbf{r} \geq 0\}$;

Output: the set V_1.

The STEP 2 in the above algorithm can be derandomized giving a 0.878-approximation algorithm. This is a very good result from the point of view of the worst case analysis, since it has been proved that, if $\mathsf{P} \neq \mathsf{NP}$, no 0.941-approximation algorithm can exist [6].

3 The LORENA Algorithm

Goemans and Williamson's algorithm has a very good worst case performance but it can handle efficiently only graphs of small size ($n \approx 100$), while it becomes very slow for larger instances ($n \approx 500$) and prohibitive for really large scale problems ($n \approx 1000$). Besides, because of its complex design it cannot easily be implemented on dedicated circuits. However, the idea on which it is based is very interesting and in this section we present and analyze a simple algorithm which is inspired by it.

Let us consider the following relaxed problem:

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2}\sum_{i<j} a_{ij}(1 - \mathbf{v}_i \cdot \mathbf{v}_j) \\ \text{subject to} \quad & \mathbf{v}_i \in \mathbf{S}_2, \quad i = 1, \ldots, n. \end{aligned} \tag{Q}$$

Since (Q) is not a semidefinite programming problem, we look for approximate solutions of (Q) by a local search algorithm.

The LORENA algorithm consists of two main steps:

Step 1: *Solve the relaxation* (Q) *by a local search algorithm obtaining a locally optimal set of vectors* $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$*;*

Step 2: *Find the vector* $\mathbf{r} \in \mathbf{S}_2$ *that minimizes* $\frac{1}{2}\sum_{i<j} a_{ij}\,\mathrm{sgn}(\mathbf{v}_i \cdot \mathbf{r}) \cdot \mathrm{sgn}(\mathbf{v}_j \cdot \mathbf{r})$*; output the set* $V_1 = \{k \mid \mathbf{v}_k \cdot \mathbf{r} \geq 0\}$.

To detail the algorithm, which is sketched in Figure 1, let $\mathbf{v}_k = (\cos\theta_k, \sin\theta_k)$, $\mathbf{r} = (-\sin\gamma, \cos\gamma)$ and let sgn denote the *signum* function defined as $\mathrm{sgn}(x) = 1$ if $x \geq 0$, -1 otherwise. In this setting the STEP 1 is equivalent to locally minimize the function

$$f(\theta_1, \ldots, \theta_n) = \frac{1}{2}\sum_{i<j} a_{ij}\cos(\theta_i - \theta_j).$$

Note that statements 1., 2., and 3. in Figure 1 implement **Step 1**, while statement 4. implements **Step 2**. In fact statement 1. randomly chooses the initial condition, while the body of the statement 3. locally improves the function f until, for all k, it holds that:

$$|f(\Theta_1, \ldots, \Theta_k, \ldots, \Theta_n) - f(\Theta_1, \ldots, \Omega_k, \ldots, \Theta_n)| < \varepsilon, \tag{1}$$

Input: The adjacency matrix $(a_{ik})_{n\times n}$ of a graph $G = \langle V, E\rangle$, a real $\varepsilon > 0$;

1. **for** $k := 1$ **to** n **do** $\Theta_k :=$ a random number in $[0, 2\pi]$;
2. **for** $k := 1$ **to** n **do**
   $A_k := \sum_j a_{kj} \cos\Theta_j; \quad B_k := \sum_j a_{kj} \sin\Theta_j;$
3. **repeat**
   $F := 0$
   for $k := 1$ **to** n **do**
   $\Omega_k := \Theta_k;$
   $\Theta_k := \arg\min_{\theta\in[0,2\pi]} \{A_k \cos\theta + B_k \sin\theta\};$
   if $2\sqrt{A_k^2 + B_k^2}\sin^2\left(\frac{\Theta_k - \Omega_k}{2}\right) \geq \varepsilon$ **then**
   $$\left[\begin{array}{l} F := 1 \\ A_j := A_j + a_{kj}\left(\cos\Theta_k - \cos\Omega_k\right) \ \forall j; \\ B_j := B_j + a_{kj}\left(\sin\Theta_k - \sin\Omega_k\right) \ \forall j; \end{array}\right.$$
   else $\Theta_k := \Omega_k$
   until $F := 1$
4. $l := \arg\min_{k\in\{1,\ldots,n\}} \left\{\sum_{i<j} a_{ij}\,\mathrm{sgn}\left(\sin(\Theta_j - \Theta_k)\sin(\Theta_i - \Theta_k)\right)\right\};$

Output: $V_1 = \{k \mid \mathrm{sgn}(\sin(\Theta_k - \Theta_l)) = 1\}.$

Figure 1. LORENA algorithm

To prove (1), we observe that

$$|f(\Theta_1, \ldots, \Theta_k, \ldots, \Theta_n) - f(\Theta_1, \ldots, \Omega_k, \ldots, \Theta_n)| = 2\sqrt{A_k^2 + B_k^2}\sin^2\left(\frac{\Omega_k - \Theta_k}{2}\right)$$

where $A_k = \sum_j a_{kj}\cos\Theta_j$ and $B_k = \sum_j a_{kj}\sin\Theta_j$.

In fact:

$$\begin{aligned} f(\Theta_1,\ldots,\Theta_k,\ldots,\Theta_n) - f(\Theta_1,\ldots,\Omega_k,\ldots,\Theta_n) \\ &= A_k\cos\Theta_k + B_k\sin\Theta_k - A_k\cos\Omega_k + B_k\sin\Omega_k \\ &= \sqrt{A_k^2+B_k^2}\,(\cos(\Theta_k-\alpha_k)-\cos(\Omega_k-\alpha_k)) \end{aligned}$$

where $\cos\alpha_k = \frac{A_k}{\sqrt{A_k^2+B_k^2}}$ and $\sin\alpha_k = \frac{B_k}{\sqrt{A_k^2+B_k^2}}$.

Since

$$\begin{aligned} \Theta_k &= \arg\min_{\theta\in[0,2\pi]}\{A_k\cos\theta + B_k\sin\theta\} \\ &= \arg\min_{\theta\in[0,2\pi]}\{\cos(\theta-\alpha_k)\}\,, \end{aligned}$$

we obtain:

$$\Theta_k = \alpha_k + \pi.$$

Therefore:

$$\begin{aligned} |f(\Theta_1,\ldots,\Theta_k,\ldots,\Theta_n) - f(\Theta_1,\ldots,\Omega_k,\ldots,&\Theta_n)| \\ &= \sqrt{A_k^2+B_k^2}\,|1-\cos(\Omega_k-\Theta_k)| \\ &= 2\sqrt{A_k^2+B_k^2}\sin^2\left(\frac{\Omega_k-\Theta_k}{2}\right). \end{aligned}$$

As regards the computation time required by LORENA, let us give a rough evaluation of the time complexity of statement 3, the more demanding of the algorithm.

Let T be the number of operations required, N_{True} be the number of times that the condition of the **if** statement is satisfied and N_{False} be the number of times that the condition of the **if** statement is not satisfied. We observe that $N_{\text{False}} \leq n\left(N_{\text{True}}+1\right)$.

When the condition of the **if** statement is satisfied an increment $|\Delta f| \geq \varepsilon$ is guaranteed and $\mathcal{O}(n)$ operations are done; this implies that $N_{\text{True}} = \mathcal{O}(\frac{|E|}{\varepsilon})$. When the condition of the **if** statement is not satisfied only $\mathcal{O}(1)$ operations are done. Therefore:

$$T = N_{\text{False}}\mathcal{O}(1) + N_{\text{True}}\mathcal{O}(n) = \mathcal{O}\left(\frac{n|E|}{\varepsilon}\right).$$

To summarize:

Proposition 1 *Lorena algorithm works in time* $\mathcal{O}\left(\frac{n|E|}{\varepsilon}\right)$.

Once ε is fixed, the computation time of LORENA is comparable with that of the neural algorithm H_NET.

As far as the approximation quality is concerned, we can state:

Proposition 2 *Given a graph* $\langle\{1,\ldots,n\},E\rangle$, *let* M *be the size of the cut found by* LORENA. *Then:*

$$M \geq 0.39|E| + \frac{1}{4}\sum_{i=1}^{n}\frac{|A_i|}{|\cos\theta_i|},$$

where $(\theta_1,\ldots,\theta_n)$ *is the local maximum found by step 3 of* LORENA, *and* $A_i = \sum_k a_{ik}\cos\theta_k$.

PROOF. Since $\boldsymbol{\theta} = (\theta_1,\ldots,\theta_n)$ is a local maximum of

$$f(x_1,\ldots,x_n) = \frac{1}{2}\sum_{i<j} a_{ij}(1-\cos(x_i-x_j)),$$

then:

$$\left.\frac{\partial f}{\partial x_i}\right|_{\boldsymbol{\theta}} = \sum_k a_{ik}\sin(\theta_i-\theta_k) = 0, \quad i=1,\ldots,n;$$
$$\left.\frac{\partial^2 f}{\partial x_i^2}\right|_{\boldsymbol{\theta}} = \sum_k a_{ik}\cos(\theta_i-\theta_k) < 0, \quad i=1,\ldots,n.$$

Setting $A_i = \sum_k a_{ik}\cos\theta_k$ and $B_i = \sum_k a_{ik}\sin\theta_k$, we can write

$$A_i\sin\theta_i - B_i\cos\theta_i = 0, \quad i=1,\ldots,n;$$
$$A_i\cos\theta_i + B_i\sin\theta_i < 0, \quad i=1,\ldots,n.$$

That implies:

$$\sum_k a_{ik}\cos(\theta_i-\theta_k) = -\frac{|A_i|}{|\cos\theta_i|} < 0$$

Now, fixed an angle α, the size of the cut defined by the set $\{k \mid \sin(\theta_k-\alpha)\geq 0\}$ is found evaluating the function $C = \frac{1}{2}\sum_{i<j} a_{ij}(1-\mathrm{sgn}(\sin(\theta_i-\alpha)\sin(\theta_j-\alpha)))$.

The expected value $\mathcal{E}$ of C, obtained selecting α randomly on $[0, 2\pi)$ accordingly to the uniform distribution, can easily be computed:

$$\mathcal{E} = \sum_{i<j} a_{ij} S(|\theta_i - \theta_j|), \qquad \text{where} \qquad S(x) = \begin{cases} \frac{x}{\pi} & \text{if } 0 \leq x < \pi; \\ 2 - \frac{x}{\pi} & \text{if } \pi \leq x < 2\pi. \end{cases}$$

Finally, observing that $S(|x|) \geq 0.39 - \dfrac{cosx}{2}$, we conclude:

$$\begin{aligned} M \geq \mathcal{E} &= \sum_{i<j} a_{ij} S(|\theta_i - \theta_j|) \\ &\geq 0.39|E| - \frac{1}{4} \sum_{i,j=1}^{n} a_{ij} \cos(\theta_i - \theta_j) \\ &= 0.39|E| + \frac{1}{4} \sum_{i=1}^{n} \frac{|A_i|}{|\cos \theta_i|}. \end{aligned}$$

□

Owing to the difficulty to estimate the term $\sum_{i=1}^{n} \frac{|A_i|}{|\cos \theta_i|}$, we are not able to evaluate the worst case performance of LORENA. We can only guarantee the weak result that LORENA is a 0.39-approximation algorithm, even if the experimental analysis gives evidence that it works much better (see next Section).

4 Experimental Results

To give some evidence that LORENA behaves very well in practice we directly compare its performances with those of RR_SDP, presented in Section 2. Such performances are analyzed both in terms of *solution quality* and in terms of *computation time*.

The problem instances we consider are of two types: unweighted p-random graphs[1] of several size, and weighted graphs taken from the TSPLib benchmark (complete geometric graphs defined by Traveling Salesman Problems). The two algorithms have been implemented in MatLab code and run on a PC Pentium 100.

To implement the first step of the Goemans and Williamson's algorithm we used the MatLab code [7] that solves the semidefinite programming problems via the interior point method; for the second step we generated 30 random vectors uniformly distributed on the n-dimensional sphere, reporting the best

[1] A graph in which an edge (i, j) is given with probability p, uniformly and independently for each pair of distinct vertices $\{i, j\}$.

of the 30 cuts induced (see the description in Section 2). To generate the random vectors we use the fact that the random vector $\frac{1}{\sqrt{X_1^2+\cdots+X_n^2}}(X_1,\ldots,X_n)$ is uniformly distributed on the sphere if $X_1,\ldots,X_n$ are iid normal random variables [3].

With regard to the experiments on p-random graphs, instances with a number of vertices that varies between 50 and 400 have been generated . For each size n we considered three different edge density: low ($p = 0.1$), medium ($p = 0.5$) and high ($p = 0.9$). For each pair (n, p) we generated 50 p-random graphs and compute the average cut value and the corresponding standard deviation. Furthermore, on such instances we give the performances of the algorithm H_Net.

Table 1 summarizes the results of the experiments.

The first two columns, titled **n** and **p**, describe the graphs characteristics. The columns H_Net, RR_SdP and Lorena show the average cut value found by each algorithm; the standard deviation is reported between parenthesis. The last two columns give the average of the differences Δ (with standard deviation σ between parenthesis) of the cuts found by Lorena and H_Net and by Lorena and RR_SdP respectively. The average differences Δ are also graphically pictured in the Figure 2. Observe that for $n \geq 200$ Lorena is better than RR_SdP whit high confidence ($\frac{\Delta}{\sigma} > 2$).

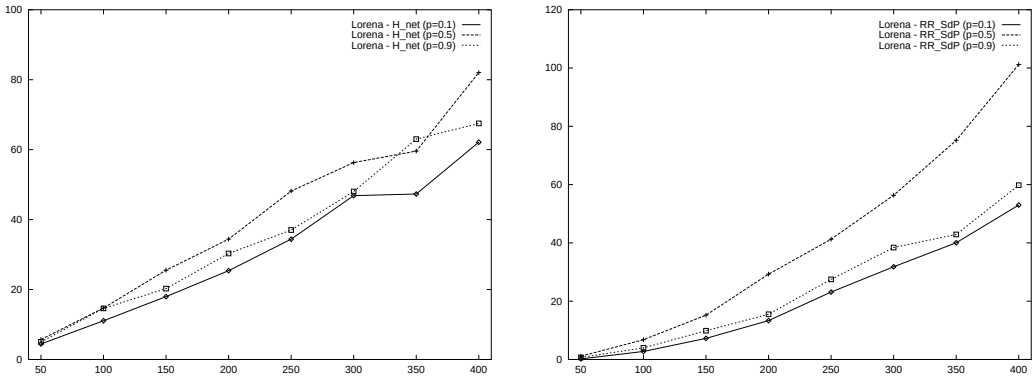

Figure 2. Linear interpolation of the column Lorena-Net (left) and the column Lorena-SdP (right) in Table 1.

As regards the computation time, in Table 2 the results on 0.5-random graphs with 100, 200, 300 and 400 vertices are given. In columns named RR_SdP and Lorena the average cut values found by the two algorithms and the standard deviations are reported. Columns named RR_SdP **time** and Lorena **time** show the average computation times (in seconds) and their standard deviations. Observe that data show a significant difference in favor of Lorena for $n \geq 200$.

Table 1
Average cut (and standard deviation) found by the three algorithms H_Net, RR_SdP and Lorena on p-random graphs of size n and average differences on cuts (std. deviations) found by Lorena and the others two algorithms.

n	**p**	H_Net	RR_SdP	Lorena	Lor−Net	Lor−SdP
50	0.1	94.7 (6.7)	98.9 (6.7)	99.1 (6.8)	4.4 (2.5)	0.2 (0.9)
50	0.5	365.7 (10.1)	370.3 (9.3)	371.4 (9.8)	5.7 (4.1)	1.1 (2.1)
50	0.9	592.2 (4.6)	596.6 (4.0)	597.3 (3.9)	5.0 (2.7)	0.7 (1.0)
100	0.1	339.3 (12.5)	347.6 (12.1)	350.4 (12.6)	11.1 (5.5)	2.7 (3.2)
100	0.5	1416.8 (18.9)	1424.7 (18.2)	1431.5 (17.9)	14.7 (10.1)	6.7 (4.6)
100	0.9	2339.3 (11.0)	2349.9 (8.5)	2353.9 (9.2)	14.6 (7.2)	3.9 (3.4)
150	0.1	737.0 (16.3)	747.8 (16.1)	755.0 (16.0)	17.9 (8.1)	7.2 (5.0)
150	0.5	3113.5 (28.1)	3123.8 (24.4)	3139.0 (24.3)	25.5 (15.9)	15.2 (6.6)
150	0.9	5238.0 (17.6)	5248.4 (16.9)	5258.2 (16.6)	20.2 (10.5)	9.8 (6.0)
200	0.1	1274.6 (25.5)	1286.7 (22.2)	1300.0 (21.7)	25.3 (13.5)	13.3 (5.5)
200	0.5	5464.6 (34.7)	5469.7 (32.7)	5499.0 (33.6)	34.4 (23.0)	29.2 (14.8)
200	0.9	9267.8 (26.3)	9282.5 (22.0)	9298.1 (20.3)	30.3 (15.3)	15.5 (7.0)
250	0.1	1949.1 (41.1)	1960.3 (37.0)	1983.5 (37.4)	34.4 (15.3)	23.1 (8.9)
250	0.5	8463.0 (50.0)	8469.9 (44.5)	8511.1 (44.2)	48.1 (32.1)	41.2 (15.4)
250	0.9	14441.5 (33.5)	14451.0 (30.1)	14478.5 (26.2)	37.0 (17.4)	27.4 (10.7)
300	0.1	2756.9 (37.3)	2771.9 (36.7)	2803.7 (38.4)	46.8 (21.4)	31.8 (11.7)
300	0.5	12112.6 (57.3)	12112.5 (48.5)	12168.9 (49.8)	56.3 (30.8)	56.3 (21.6)
300	0.9	20751.4 (38.5)	20761.1 (34.0)	20799.5 (32.4)	48.0 (20.6)	38.4 (12.9)
350	0.1	3710.3 (44.3)	3717.5 (42.2)	3757.6 (46.0)	47.3 (22.6)	40.0 (11.9)
350	0.5	16413.4 (69.1)	16397.8 (61.4)	16472.9 (63.7)	59.5 (36.2)	75.1 (24.0)
350	0.9	28187.8 (44.9)	28207.9 (47.9)	28250.8 (43.0)	63 (21.3)	42.9 (15)
400	0.1	790.8 (42.4)	4800 (43.9)	4853 (41.9)	62.1 (28.7)	53 (19.6)
400	0.5	21329.3 (93.3)	21310.1 (79.3)	21411.3 (78.0)	82.0 (47.7)	101.2 (25.5)
400	0.9	36785.5 (41.4)	36793.1 (42.6)	36853 (40.5)	67.5 (26.5)	59.8 (18.3)

In the second experiment Lorena and RR_SdP have been compared on some instances from the TSPLib. In Table 3 are reported both the cut values and the

Table 2
Average computation time (in seconds) and standard deviation of the two algorithms RR_SDP and LORENA on 0.5-random graphs of various size n.

n	RR_SDP **time**	LORENA **time**
100	15.7 (0.6)	12.6 (4.6)
200	197.1 (7.5)	61.9 (29.5)
300	825.8 (31.9)	153.3 (56.1)
400	2374.8 (63.2)	243.2 (82.6)

computation time of the two algorithms. The cut values are given in columns RR_SDP and LORENA, whereas in columns RR_SDP **time** and LORENA **time** are given the results of the MatLab implementations of LORENA and RR_SDP. These results confirm the previous observations since LORENA obtains the same cut values in shorter time. In addition, in column LORENA **time** (C code) we also give the computation times of a more efficient implementation of LORENA (C code) run on PC Pentium 100.

Table 3
Cut values (columns 3, 5) and computation time (columns 4, 6) in seconds of RR_SDP and LORENA implemented in MatLab code and applied to some TSPLib instances. In column 7 is reported the time of a a more efficient implementation in C code.

Inst.	**size**	RR_SDP	RR_SDP **time** MatLab code	LORENA	LORENA **time** MatLab code	LORENA **time** C code
dant42	42	42638	1.7	42638	1.6	0.15
gr48	48	321815	2.3	321815	1.1	0.16
gr120	120	2156775	45.0	2156775	6.2	0.87
hk48	48	771712	2.6	771712	1.8	0.15

References

[1] M. A. Alberti, A. Bertoni, P. Campadelli, G. Grossi, and R. Posenato. A neural algorithm for MAX-2SAT: Performance analysis and circuit implementation. *Neural Networks*, 10(3):555–560, 1997.

[2] F. Alizadeh. Optimization over the positive semi-definite cone: Interior–point methods and combinatorial applications. In P. M. Pardalos, editor, *Advances in Optimization and Parallel Computing*, pages 1–25. North Holland, Amsterdam, The Netherlands, 1992.

[3] L. Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, 1986.

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

[5] F. Hadlock. Finding a maximum cut of a planar graph in polynomial time. *SIAM Journal of Computing*, (4):221–225, 1975.

[6] J. Håstad. Some optimal inapproximability results. In *Proceedings of the 29th ACM Symposium on the Theory of Computation*, pages 1–10. ACM, 1997.

[7] C. Helmberg, F. Rendl, R. J. Vanderbei, and H. Wolkowicz. An interior point method for semidefinite programming. Technical report, University of Graz, 1994.

[8] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences*, pages 2554–2558, 1982.

[9] J. J. Hopfield and D. W. Tank. Neural computation of decision in optimization problems. *Biological Cybernetics*, (52):141–152, 1985.

[10] R. M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Complexity of Computer Computations. Plenum Press, New York, 1972.