Improved approximation of Max-Cut on graphs of bounded degree

Uriel Feige* Marek Karpinski† Michael Langberg‡

December 9, 2001

Abstract

Let $\alpha \simeq 0.87856$ denote the best approximation ratio currently known for the Max Cut problem on general graphs [GW95]. We consider a semidefinite relaxation of the Max Cut problem, round it using the random hyperplane rounding technique of ([GW95]), and then add a local improvement step. We show that for graphs of degree at most Δ, our algorithm achieves an approximation ratio of at least $\alpha + \epsilon$, where $\epsilon > 0$ is a constant that depends only on Δ. Using computer assisted analysis, we show that for graphs of maximal degree 3, our algorithm obtains an approximation ratio of at least 0.921, and for 3-regular graphs, the approximation ratio is at least 0.924. We note that for the semidefinite relaxation of Max Cut used in [GW95], the integrality gap is at least 1/0.885, even for 2-regular graphs.

1 Introduction

Given a graph $G = (V, E)$, the Max-Cut problem on G is the problem of finding a partition (X, Y) of the vertex set V which maximizes the number of edges with one endpoint in X and another in Y. An r approximation algorithm for the Max-Cut problem is an algorithm that when evoked on a graph G yields a partition (X, Y) of value at least r times the value of the optimal partition (such an algorithm is also said to have an approximation ratio of r).

The currently best known approximation algorithm for the Max-Cut problem, is the algorithm of Goemans and Williamson [GW95], we denote this algorithm as the GW algorithm. Given a graph G with vertex size n, the GW algorithm has two steps. In the first step a semidefinite relaxation of the Max-Cut problem on G is solved yielding a set of unit vectors $\{v_1, \ldots, v_n\}$ in R^n. In the second step this set of vectors is randomly rounded into a partition (X, Y). The GW algorithm is shown to have an expected approximation ratio of at least $\alpha \simeq 0.87856$.

Given G, let (X, Y) be the partition obtained by the GW algorithm. It is evident from [GW95] that it might be the case that a vertex of G and more than half its neighbors end

*Department of Computer Science and Applied Mathematics, the Weizmann Institute.

†Department of Computer Science, University of Bonn.

‡Department of Computer Science and Applied Mathematics, the Weizmann Institute.

up on the same side of the partition (X, Y). We denote such vertices as *misplaced* vertices. Clearly in such a case the partition (X, Y) can be improved by moving the misplaced vertex from one side of the partition to the other, thus yielding a new partition cutting more edges. In the following work, we consider enhancing the GW algorithm with an additional local step that moves misplaced vertices from one side of the partition to the other, until no such vertices are left. We prove that the enhanced algorithm improves upon the original approximation ratio of α by a constant fraction when the maximal degree Δ of the given graph is constant. Namely we prove an approximation ratio of 0.921 on graphs of maximal degree three, of 0.924 on regular graphs of degree three, and of $\alpha + \Omega(1/\Delta^4)$ on graphs of maximal degree Δ. Our results apply to graphs G for which we assume that there are no weights associated with the edge set E. Our analysis can be extended to graphs with edge weights as well, although the results we obtain are weeker (we briefly discuss this in Section 5).

The Max-Cut problem on bounded degree graphs is mentioned in [BK98] where it is shown that it is *NP-hard* to approximate the Max-Cut problem on regular graphs of degree 3 beyond the ratio of 0.997. To the best of our knowledge, the best known approximation ratio for the Max-Cut problem on bounded degree graphs is that of the general problem which is $\alpha \simeq 0.87856$. The Max-Cut problem restricted to 3 regular graphs has been previously studied in [TFMP99] (where a parallel approximation algorithm of ratio 3/4 is presented), and in [Pol95] (where it is shown that a local maximum can be found in polynomial time).

Our results are achieved using an algorithm that differs from the one presented in [GW95] not only in the additional improvement step, but also in the fact that an *improved* semidefinite relaxation of the Max-Cut problem is used as a base of our algorithm. Specifically, we add *triangle* constraints (mentioned in [FG95]) to our relaxation. These additional constraints have been studied in the past in the context of several problems including the Max-Cut problem. The results of [BM86] imply that the value of the Max-Cut semidefinite relaxation with triangle constraints is **equal** to the value of the optimal cut on planar graphs. Without these constraints it is shown in [FS01] that the original semidefinite relaxation has an integrality gap arbitrarily close to $1/\alpha$. In [GW95] it is shown that the integrality gap of the original relaxation has an integrality gap of least $1/0.885$ even on two regular graphs (the 5 cycle). In [Zwi99] these constraints are used in order to achieve an improved approximation ratio on the Max-Cut problem on graphs which have a fractional triangle covering. The addition of such triangle constraints to the semidefinite relaxation of the Max-Cut problem on general graphs is not known to improve the *worst case* integrality gap of such a relaxation nor is it known to yield an improved approximation algorithm. In [Kar96] it is shown that the addition of such constraints will not improve the approximation ratio of the algorithm presented in [GW95] as is. In our work, the addition of such triangle constraints to the semidefinite relaxation of the Max-Cut problem on graphs of maximal degree three yields an improved approximation algorithm with a ratio of 0.921. As the original semidefinite relaxation of [GW95] has an integrality gap of at least $1/0.885$ on a two regular graph, we conclude that the addition of such triangle constraints to our semidefinite relaxation is crucial to the success of our algorithm.

As in the algorithm of [GW95], many other algorithms based on semidefinite programming use the *random hyperplane* rounding technique, and are analyzed in a *local* manner (for instance [FG95, KZ97, Zwi99]). That is, for each edge a local expected approximation ratio is computed. This ratio then holds as an expected approximation ratio for the algorithm as a whole due to linearity of expectation.[1] In our algorithm for the Max-Cut problem on graphs with maximal degree three, we present an analysis in which we compute a local approximation ratio not on single edges but on clusters of edges. In particular we concentrate on pairs of edges which share a common vertex. Such an analysis allows us to evaluate the contribution of our additional improvement step as a function of the vector configuration obtained by the semidefinite relaxation.

The paper is structured as follows. In Section 2 we briefly review the Goemans and Williamson algorithm [GW95] for the Max-Cut problem. In Section 3 we present our enhanced algorithm and its analysis for graphs of maximal degree three. In Section 4 we present our results for graphs of maximal degree Δ.

2 Preliminaries

Notation Throughout our work we assume that $G = (V, E)$ is an undirected graph, with vertex set $V = \{1, \ldots, n\}$. We assume that there are no weights associated with the edge set E. In general our results generalize to the case in which G has edge weights, we briefly discuss this in Section 5. For any set $X \subseteq V$ let $w(X)$ be the number of edges with one end point in X and another in $V \setminus X$. Denote the total number of edges by $|E|$, and the number of edges cut by the optimal partition of G as $Opt(G)$. $w(X)$ is referred to as the *value* or *weight* of the cut (X, Y). For any natural number n we denote the set $\{1, \ldots, n\}$ by $[n]$. Finally, throughout our work α will represent the expected approximation ratio of the GW algorithm for Max-Cut [GW95].

The GW algorithm The Max-Cut problem on a graph $G = (V, E)$ with $|V| = n$ can be represented as a quadratic integer program:

(QI-MC)	Maximize	$\sum_{e_{ij} \in E} \frac{1 - x_i \cdot x_j}{2}$	
	subject to:		
		$x_i \in \{-1, 1\}$	for $i \in [n]$

The above program can be understood as follows. With each vertex $i \in V$ we associate a variable x_i, and the value of x_i (which is either $+1$ or -1) indicates in which side of the cut the respective vertex is placed. For each edge $e_{ij} \in E$, if $x_i \neq x_j$ (corresponding to the case in which e_{ij} is cut) then the value of $\frac{1 - x_i \cdot x_j}{2}$ is 1, and if $x_i = x_j$ then this value is 0.

The requirement $x_i \in \{-1, 1\}$ can be relaxed by representing each variable x_i by a unit n-dimensional vector $v_i \in S^n$ (here S^n is the unit sphere) and the multiplication $x_i \cdot x_j$ by the inner product $\langle v_i, v_j \rangle$.

(SDP-MC)	Maximize	$\sum_{e_{ij} \in E} \frac{1 - \langle v_i, v_j \rangle}{2}$

[1] We would like to note that other techniques have been used in the analysis of approximation algorithms based on semidefinite programming. For example Nesterov [Nes98] presents a general framework for analyzing the quality of semidefinite relaxations to quadratic optimization problems. Applying his techniques on the semidefinite relaxation of Max-Cut results with a $2/\pi \simeq 0.6366$ approximation ratio.

subject to:

$$v_i \in S^n \qquad \text{for } i \in [n]$$

As every solution of *(QI-MC)* is also a solution of *(SDP-MC)*, the value of *(SDP-MC)* is at least as large as that of *(QI-MC)*. *(SDP-MC)* can be solved (up to arbitrary precision) in polynomial time using semidefinite programming (see [GW95]).

The algorithm suggested in [GW95] for the Max-Cut problem has two steps. In the first step the relaxation *(SDP-MC)* is solved. A solution to *(SDP-MC)* is a set of unit vectors $v_1, \ldots, v_n$ in R^n, rather than a cut of G. In the second step a cut (X, Y) of G is obtained by *rounding* the set of optimal vectors $v_1, \ldots, v_n$ using the *random hyperplane* rounding technique. That is, a random vector $r \in S^n$ is chosen, and the set X is defined to be the set of vertices i with corresponding vectors v_i that have a positive inner product with r. The vector r can be viewed as the normal to a random hyperplane passing through the origin.

It is shown in [GW95] that the expected contribution of each edge $e_{ij} \in E$ to the cut (X, Y) is $\frac{\theta_{ij}}{\pi}$, where θ_{ij} is the angle between the vectors v_i and v_j. As the contribution of each edge to the objective function of *(SDP-Cut)* is $\frac{1-\cos\theta_{ij}}{2}$ we conclude that the expected approximation ratio achieved on each edge is $\frac{2\theta_{ij}}{\pi(1-\cos\theta_{ij})}$. This expected ratio is minimal when $\theta_{ij} = \theta_0 \simeq 2.3311$, and in this case obtains the value $\alpha \simeq 0.87856$. We conclude that the expected weight $w(X)$ of the partition (X, Y) obtained by the random hyperplane rounding technique is at least $\alpha Opt(G)$, and will be exactly $\alpha Opt(G)$ when for all edges e_{ij} we have that the angle θ_{ij} is θ_0 or zero.

3 Max-Cut on graphs of maximal degree 3

We present an algorithm which given $G = (V, E)$ as above will output a partition (X, Y) of V such that $w(X) \geq rOpt(G)$ for $r \geq 0.921$. Our algorithm is an extension of the Max-Cut algorithm presented in [GW95].

In the first step of our algorithm an enhanced semidefinite relaxation of the Max-Cut problem on G is solved to achieve a set of n unit vectors $\{v_1, \ldots, v_n\}$. These vectors are then rounded into a cut (X, Y) of G using the random hyperplane rounding technique of [GW95]. As mentioned in the Introduction, it may be the case that there are misplaced vertices in the cut (X, Y) (*i.e.* vertices with more neighbors on their side of the cut than on the opposite side). In the last step of our algorithm, we move these vertices from one side to the other until no more misplaced vertices remain. This is done in a *greedy* manner. Following we present the semidefinite relaxation used in the first step of our algorithm and define the greedy procedure used in the last step.

The semidefinite relaxation Let T be the set of triplets (i, j, k) in which i, j, k are vertices in V and both j and k are neighbors of i (for technical reasons we also assume $j < k$). The semidefinite relaxation we use in our algorithm is

(SDP-Cut)	Maximize	$\sum_{e_{ij} \in E} \frac{1-\langle v_i, v_j \rangle}{2}$	
	subject to:		
	(1)	$v_i \in S_n$	for $i \in [n]$

$$
\begin{aligned}
&(2) \qquad && \langle v_i, v_j\rangle + \langle v_i, v_k\rangle + \langle v_j, v_k\rangle \geq -1 && \\
& && \langle v_i, v_j\rangle - \langle v_i, v_k\rangle - \langle v_j, v_k\rangle \geq -1 && \text{for all } i,j,k \in [n] \\
&(3) && \langle v_i, v_j\rangle + \langle v_i, v_k\rangle + \langle v_j, v_k\rangle = -1 && \text{for all } (i,j,k) \in T.
\end{aligned}
$$

Notice that we have added two families of constraints to the original relaxation of [GW95]. The triangle constraints *(2)*, and the *tight* triangle constraints *(3)*. The triangle constraints are valid when the vectors $\{v_1, \ldots, v_n\}$ are restricted to be one dimensional and thus can be added to *(SDP-Cut)*. It is shown in [GW95] that the original semidefinite relaxation *SDP-MC* (without triangle constraints) has an integrality gap of at least $1/0.885$ even on 2 regular graphs (the 5 cycle). Hence these additional triangle constraints are crucial to our analysis which results in an approximation ratio of 0.921 on such graphs.

For the validity of the tight triangle constraints consider an **optimal** partition (X, Y) of $G = (V, E)$. As G has maximal degree three, for every vertex v it cannot be the case that v and two of its neighbors lie on the same side of the partition (*i.e.* v is misplaced). In such a case moving v to the other side of the partition would increase the weight of the partition, which is a contradiction to its optimality. Hence we may add a corresponding constraint to *(SDP-Cut)* which rules out the possibility of misplaced vertices, when the corresponding vectors of *(SDP-Cut)* are restricted to be one dimensional. This constraint is the tight triangle constraint.

The greedy phase As mentioned earlier after the first step of our algorithm it might be the case that for some triplet $(i, j, k) \in T$, defined above, the vertices i, j and k lie on the same side of the partition. In such a case, moving the misplaced vertex i to the other side of the partition will increase its weight. We denote such a triplet $(i, j, k) \in T$ in which i is misplaced as a *good* triplet.

Given a partition (X, Y) we are interested in moving all misplaced vertices until none are left. In general at each step of this greedy process we could decide to move the one misplaced vertex that increases the weight of the partition by most. But as moving one misplaced vertex affects other vertices, we are also interested that the vertex moved does not *destroy* many good triplets (a good triplet is destroyed if it is no longer good). In order to combine these two interests, at each stage of our greedy process we move the vertex for which the ratio between the weight added to the partition by moving it from one side of the partition to the other and the number of good triplets destroyed by this act, is maximal. We denote the resulting algorithm as A_{cut}.

Algorithm A_{cut}:

1. Solve *(SDP-Cut)* to obtain an optimal vector configuration $\{v_1, \ldots, v_n\}$ of value Z. Round the vector configuration using the random hyperplane rounding technique from [GW95]. Denote the partition obtained by (X, Y).

2. Greedily move misplaced vertices from one side of the partition to the other according to the procedure above.

Theorem 3.1 *A_{cut} has an expected approximation ratio of $r \geq 0.921$.*

Proof: First we note that w.l.o.g. we may assume that G does not have any vertices of degree 1. Otherwise we may run algorithm A_{cut} on the graph $\hat{G}$ obtained by iteratively removing all vertices of degree one in G. It can be seen that an approximation algorithm with ratio r on $\hat{G}$ yields an approximation algorithm with ratio at least r on G.

Assume G is as above, let $\{v_1, \ldots, v_n\}$ be the optimal vector configuration, and let (X, Y) be the partition after step *(1)* of algorithm A_{cut}. Let $W = w(X)$ be the weight of this partition. Recall, using the analysis of [GW95] that in expectation $W \geq \alpha Opt(G)$. Furthermore, the expected value of W is $\alpha Opt(G)$ only if for each edge e_{ij} in G the angle between its corresponding vectors v_i and v_j is either 0 or $\theta_0 \simeq 2.3311$. Our main observation is the fact that in both these *worst* cases (and in the case in which we are *close* to being in the worst case) there is some constant probability that a vertex and two of its neighbors lie on the same side of the partition (X, Y). In such cases moving the vertex from one side of the partition to the other will increase the weight of the partition.

Geometrically speaking, one can view our observation in the following way. Let v be some vertex in V, and y_1, y_2 be two of its neighbors. Denote the corresponding vectors in the optimal vector configuration as v, y_1, y_2 respectively. Assume that the angles between the vectors v and y_1, v and y_2 are exactly θ_0 (a similar analysis can be given for the case in which these angles are close to θ_0), then by constraint *(2)* of *(SDP-Cut)*, it **cannot** be the case that all three vectors lie on the same plane. Furthermore, this constraint implies that the angle between the vector v and the plane containing the vectors y_1 and y_2 is at least 0.57, which in turn implies that with probability at least 0.07 the vertex v will be misplaced (after the randomized rounding).

The upcoming lemmas extend this observation. We use a refinement of the set T. For each edge e_{ij} let d_{ij} be the number of triplets in T in which e_{ij} appears. It can be seen that in a graph of minimum degree 2 and maximum degree 3, $d_{ij} = 2$ if the degree of v_i and v_j are 2, $d_{ij} = 3$ if the degree of v_i differs from the degree of v_j, and $d_{ij} = 4$ otherwise. Let T be the set of triplets in G; for $l = 4, 5, \ldots, 8$ denote the set of triplets $(i, j, k) \in T$ in which $d_{ij} + d_{ik} = l$ by T_l. Given a partition (X, Y) we denote by S_l the number of good triplets in T_l.

Lemma 3.2 *Let (X, Y) be some partition in G of weight W. Let T, T_l, and S_l for $l = 4, \ldots, 8$ be defined as above. Executing step* (2) *of algorithm A_{cut} on the partition (X, Y) will yield a new partition of weight at least*

$$W + \frac{2}{3}S_4 + \frac{1}{2}S_5 + \frac{2}{5}S_6 + \frac{3}{8}S_7 + \frac{1}{3}S_8$$

Proof: Roughly speaking, we define the contribution of each triplet (i, j, k) that is good in (X, Y) as the number of edges it adds to the partition the moment it is destroyed by the greedy phase of our algorithm. Let τ be one of the vertices i, j, or k. If (i, j, k) is destroyed by the movement of the vertex τ, and by this act p edges are added to the partition and q triplets (including (i, j, k)) are destroyed, we fix the contribution of the triplet (i, j, k) to be p/q. Hence, due to the nature of our greedy phase, we may bound the contribution of the good triplet (i, j, k) from below by computing the ratio between the number of edges added to the partition and the number of triplets destroyed by the act of moving any one of the vertices i, j, or k from one side of the partition to the other. By finding such a lower bound for each good triplet in T_l ($l = 4, \dots, 8$) we conclude our assertion.

Note that for any good triplet (i, j, k) and any vertex τ as above, the contribution of moving τ from one side of the partition to the other is at least one edge, while this act will destroy at most 9 triplets. It follows that we may trivially claim that executing step *(2)* of algorithm A_{cut} on the given partition (X, Y) will yield a new partition of weight at least

$$W + \frac{1}{9}(S_4 + S_5 + S_6 + S_7 + S_8).$$

The following case analysis refines this trivial analysis and provides a full proof of the asserted lemma.

Case 1: Let (i, j, k) be a good triplet in T_4, *i.e.* the vertices i, j and k are of degree 2, and all lie on the same side of the partition (X, Y). As the greedy step *(2)* of A_{cut} continues as long as there is some misplaced vertex, the triplet (i, j, k) will be destroyed sometime during our greedy procedure. Denote the vertex moved when (i, j, k) is destroyed by τ (τ is either i, j or k). Let p be the number of edges added to the partition when (i, j, k) is destroyed, and q be the number of good triplets destroyed along with (i, j, k).

As mentioned above the degrees of i, j and k are 2. Hence, if (i, j, k) is a good triplet, then by moving the vertex i we destroy at most 3 triplets and increase the weight of the partition by exactly 2 edges. Due to the nature of our greedy phase, we conclude that by moving τ from one side of the partition to the other, the ratio between the number of edges added to the partition and the number of triplets destroyed is at least 2/3. We conclude that when (i, j, k) is destroyed it contributes at least 2/3 to the partition.

Case 2: Let (i, j, k) be a good triplet in T_5; using the same line of analysis displayed in the previous case, it is enough to analyze the movement of the vertex i from one side of the partition to the other. Assume that the vertex j is of degree 3 and vertices i, k are of degree 2 (it cannot be the case that i is of degree 3). In Figure 1 a schematic view of our case is presented. Solid lines represent edges that **preserve** sides of the partition (for instance the vertices i, j and k lie on the same side of the partition), dotted lines represent edges that are **not known to preserve** sides of the partition, and bold lines represent edges that are known **not to preserve** sides of the partition. It can be seen that moving i from one side of the partition to the other we gain exactly 2 edges to the partition, but destroy at most 4 triplets. We conclude that (i, j, k) contributes at least 1/2 edges to the partition.

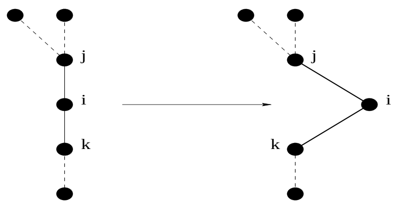

Figure 1: Case 2.

Case 3: Let (i, j, k) be a good triplet in T_6. There are two possibilities; either the vertex i

is of degree 2 and the vertices j, k are of degree 3, or the vertex i is of degree 3 and j, k are of degree 2. Both cases are presented in Figure 2. In the first case *(a)*, moving i to the other

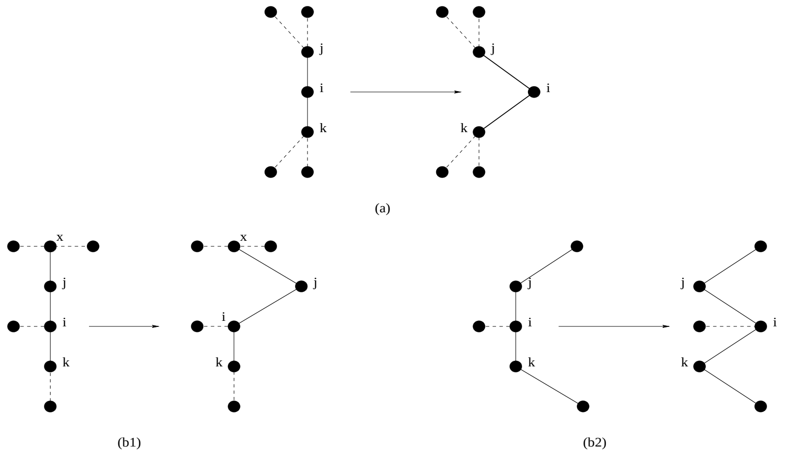

Figure 2: Case 3.

side of the partition increases the weight of the partition by 2 edges and destroys at most 5 triplets. Hence, if we are in this case then (i, j, k) contributes $2/5$ when it is destroyed.

In the second case we consider two sub-cases: *(b1)* the case in which one of the vertices j and k is on the same side as its additional neighbor x, *(b2)* the case in which the additional neighbors of j and k are both on the opposite side of the partition. In the first case we have that moving j from one side of the partition to the other will contribute 2 edges to the cut and destroy at most 5 triplets, yielding a ratio of $2/5$, and in the second we obtain a ratio of $3/5$ by again moving j (note that in the case *(b2)* one must fix the side of the third neighbor x of i and only then analyze); thus we conclude our assertion.

Case 4: Let (i, j, k) be a good triplet in T_7. It must be the case that the vertex i is of degree 3 and the vertices j, k are of different degree (we assume that j is of degree 2). We consider three cases that are presented in Figure 3. In the first case *(a)* all three neighbors

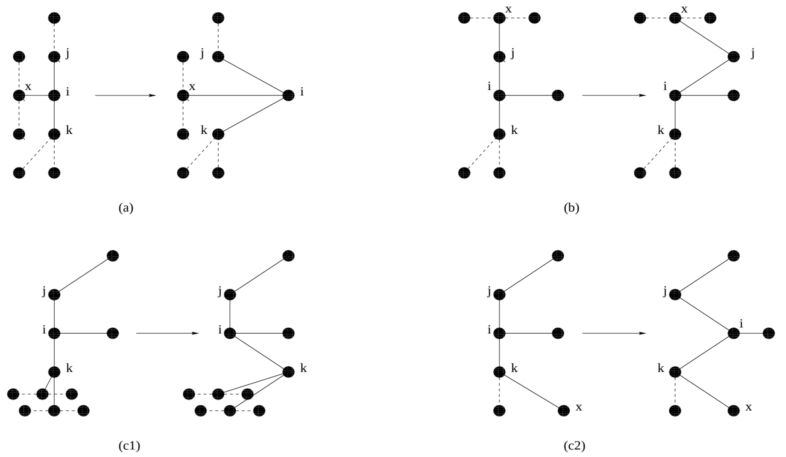

Figure 3: Case 4.

of i are on the same side as i. In such a case in moving i one can see that a ratio of at least $3/8$ is achieved. In the second and third cases we assume that the additional neighbor of i is on the opposite side of the partition.

In the second case *(b)* we assume that the additional neighbor x of j is on the same side as j. In this case moving j we achieve a ratio of at least $2/4$.

At last in the third case *(c1)* and *(c2)* we assume that the additional neighbor of j is also on the opposite side of the partition, and we consider two situations: the first when all additional neighbors of k are on the same side of the partition as k, and the second when there is some neighbor x of k on a different side. It can be seen that moving k in the first case results in a ratio of at least $3/8$, and moving i in the second results in a ratio of at least $1/2$.

Case 5: Let (i, j, k) be a good triplet in T_8. It must be the case that all three vertices are of degree 3. In Figure 4 three sub-cases are considered. In case *(a)* all three neighbors of i

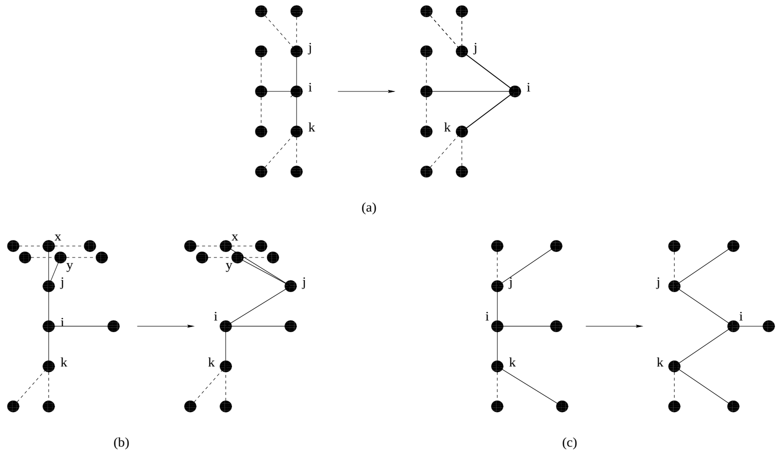

Figure 4: Case 5.

are on the same side of the partition as i; in this case moving i we obtain a ratio of $3/9$. In the remaining cases *(b), (c)* we assume that the additional neighbor of i is on the opposite side of the partition. In case *(b)* we assume that for one of the vertices j or k, all three of its neighbors are on the same side of the partition as it is (in Figure 4 we assume the above for j). In such a case moving j results in a ratio $\geq 3/8$. In the last case *(c)* both i, j, and k have at least one neighbor on an opposite side. In this case moving i we have a ratio $\geq 1/3$. □

Corollary 3.3 *Let W be the weight of the partition (X, Y) obtained by step (2) of algorithm A_{Cut}, and let S_l $(l = 4, \ldots, 8)$ be the number of good triplets in (X, Y) according to the above definition. The expected weight of the partition received by algorithm A_{Cut} is:*

$$\begin{aligned} E[w(A_{Cut})] &= E\left[W + \frac{2}{3}S_4 + \frac{1}{2}S_5 + \frac{2}{5}S_6 + \frac{3}{8}S_7 + \frac{1}{3}S_8\right] = E[W] + \sum_{l=4}^{8} \alpha_l E[S_l] \\ &= \sum_{e_{ij}} \Pr(e_{ij} \text{ is cut}) + \sum_{l=4}^{8} \sum_{(i,j,k)\in T_l} \alpha_l \Pr(v_i, v_j, v_k \text{ are not separated}) \end{aligned}$$

$$= \sum_{l=4}^{8} \sum_{(i,j,k)\in T_l} \frac{\Pr(e_{ij} \text{ is cut})}{d_{ij}} + \frac{\Pr(e_{ik} \text{ is cut})}{d_{ik}} + \alpha_l \Pr(v_i, v_j, v_k \text{ are not separated}),$$

where $(\alpha_4, \alpha_5, \ldots, \alpha_8) = (2/3, 1/2, 2/5, 3/8, 1/3)$.

Proof: All probabilities above are taken over random hyperplanes cutting the unit sphere of R^n. The second and third equalities are due to linearity of expectation, and the fourth is due to the fact that each edge e_{ij} is in exactly d_{ij} triplets in T. □

Lemma 3.4 *Let* $l \in \{4, \ldots, 8\}$. *For every triplet* $(i,j,k) \in T_l$ *we have that:*

$$\frac{\Pr(e_{ij} \text{ is cut})}{d_{ij}} + \frac{\Pr(e_{ik} \text{ is cut})}{d_{ik}} + \alpha_l \Pr(v_i, v_j, v_k \text{ are not separated}) \geq r\left(\frac{1-\langle v_i, v_j\rangle}{2d_{ij}} + \frac{1-\langle v_i, v_k\rangle}{2d_{ik}}\right)$$

for $r = 0.921$.

Proof: Let $l = d_{ij} + d_{ik}$. Define the following functions:

$$\begin{aligned}
f(\theta_{ij}, \theta_{ik}) &= \frac{\theta_{ij}}{d_{ij}\pi} + \frac{\theta_{ik}}{d_{ik}\pi} + \alpha_l\left(1 - \frac{1}{2\pi}(\theta_{ij} + \theta_{ik} + \arccos(-1 - \cos(\theta_{ij}) - \cos(\theta_{ik})))\right), \\
g(\theta_{ij}, \theta_{ik}) &= \frac{1-\cos(\theta_{ij})}{2d_{ij}} + \frac{1-\cos(\theta_{ik})}{2d_{ik}}, \\
h(\theta_{ij}, \theta_{ik}) &= \frac{f(\theta_{ij}, \theta_{ik})}{g(\theta_{ij}, \theta_{ik})},
\end{aligned}$$

where θ_{ij} (θ_{ik}) is the angle between v_i and v_j (v_i and v_k).

In [GW95] it is shown that the probability that three unit vectors v_i, v_j, and v_k are not separated by the random hyperplane rounding technique is $1 - \frac{1}{2\pi}(\theta_{ij} + \theta_{ik} + \theta_{jk})$. Furthermore recall that for every triplet $(i,j,k) \in T$ it is the case that $\cos(\theta_{ij}) + \cos(\theta_{ik}) + \cos(\theta_{jk}) = -1$ (constraint *(3)* of *(SDP-Cut)*). We conclude that the function $f(\theta_{ij}, \theta_{ik})$ represents the expected contribution of the triplet (i,j,k) to the cut. Similarly, the function $g(\theta_{ij}, \theta_{ik})$ represents the contribution of each triplet (i,j,k) to the objective function of *(SDP-Cut)*.

In order to prove the above claim it is suffice to show that $h(\theta_{ij}, \theta_{ik}) > r$ in the range $\theta_{ij} \in [0, \pi]$, $\theta_{ik} \in [\pi - \theta_{ij}, \pi]$ (the triangle constraints added to *(SDP-Cut)* imply that $\cos(\theta_{ij}) + \cos(\theta_{ik}) \leq 0$). By evaluating the value of h over a number of points in the above range and bounding the derivative of h over the whole range our lemma is proven (we use the facts that $d_{ij}, d_{ik} \in \{2,3,4\}$ and $l = d_{ij} + d_{ik} \in \{4, \ldots, 8\}$). □

By Corollary 3.3 and Lemma 3.4 we have that the expected weight of the partition obtained by algorithm A_{Cut} is

$$E[w(A_{Cut})] \geq r \sum_{l=4}^{8} \sum_{(i,j,k)\in T_l} \left(\frac{1-\langle v_i, v_j\rangle}{2d_{ij}} + \frac{1-\langle v_i, v_k\rangle}{2d_{ik}}\right) = rZ \geq rOpt(G),$$

for $r = 0.921$.

This completes the proof of Theorem 3.1. The same line of analysis yields a 0.924 approximation ratio on 3 regular graphs. □

4 Max-Cut on graphs of maximal degree Δ

In the following section we consider the Max-Cut problem on graphs of maximal degree Δ. We present a slightly different algorithm and analysis than the ones presented in the case where $\Delta = 3$, and achieve an improved approximation ratio of $\alpha + \Omega(1/\Delta^4)$.

As in the previous section, the starting point of our enhanced algorithm is the following semidefinite relaxation of the Max-Cut problem on a given graph $G = (V, E)$.

(SDP-Cut)	Maximize	$\sum_{e_{ij} \in E} \frac{1-\langle v_i, v_j \rangle}{2}$	
	subject to:		
	(1)	$v_i \in S_n$	for $i \in [n]$
	(2)	$\langle v_i, v_j \rangle + \langle v_i, v_k \rangle + \langle v_j, v_k \rangle \geq -1$	
		$\langle v_i, v_j \rangle - \langle v_i, v_k \rangle - \langle v_j, v_k \rangle \geq -1$	for all $i, j, k \in [n]$

Note that constraint *(3)* used in the previous section is no longer valid. As earlier, solving *(SDP-Cut)* on a given graph G, we obtain a set of n vectors corresponding to the vertices of G. We will concentrate on the case in which for all edges e_{ij} in G the angle between the corresponding vectors v_i and v_j is equal to $\theta_0 \simeq 2.3311$, for in this case the random hyperplane technique will yield a partition (X, Y) of expected weight no better than α times the weight of the optimal partition in G. [2]

In this *worst* case, we will show that there is some probability (depending on Δ alone) that a vertex and more than half its neighbors lie on the same side of the partition obtained by the random hyperplane rounding technique. As earlier, we denote such vertices as misplaced, and by adding an additional step which moves misplaced vertices from one side of the partition to the other, we are able to achieve an improved approximation ratio. In Theorem 4.3 we show that improving the approximation ratio for this worst case scenario suffices to obtain an improved approximation ratio when the angles θ_{ij} corresponding to edges e_{ij} are not necessarily all of value θ_0.

Consider the following algorithm for the Max-Cut problem on graphs with bounded degree Δ.

Algorithm A_{Cut}**:**

1. Solve the semidefinite relaxation *(SDP-Cut)* and round the resulting optimal vector configuration using the random hyperplane rounding technique. Denote the partition obtained by (X, Y).

2. Move misplaced vertices in (X, Y) from one side of the partition to the other until no such vertices are left.

Let $\{v_1, \ldots, v_n\}$ be the vector configuration obtained after solving the semidefinite relaxation *(SDP-Cut)* on a given graph G. Let (X, Y) be the partition obtained after rounding this vector configuration using the random hyperplane rounding technique ([GW95]). Define

[2] As noted earlier, there are other worst cases for the [GW95] algorithm, namely when for some edges e_{ij} the angles between the corresponding vectors is zero. In [Zwi99] it is shown how to deal with these cases by the use of an improved rounding technique which uses *outward rotations*. We deal with such cases in a different manner which will be explained in the proof of Theorem 4.3.

an edge e_{ij} to be *bad* if the inner product $\langle v_i, v_j \rangle$ is equal to $\cos(\theta_0) \simeq -0.688$ (*i.e.* $\theta_{ij} = \theta_0$). Define a vertex i to be *all bad* if for all neighbors j of i we have that the edge e_{ij} is bad. Denote by $N(i)$ the set of vertices adjacent to i. Following we show that an 'all bad' vertex will be misplaced in (X, Y) with probability $\Omega(1/\Delta^2)$. We will need the following lemma.

Lemma 4.1 *Let v be some vector in R^n of norm $\|v\|$, and let $r = (r_1, \ldots, r_n) \in R^n$ be the normal of a random hyperplane in R^n (i.e. each r_i has standard normal distribution $N(0,1)$). The size of the projection of v on r can be bounded as follows:*

$$\Pr[|\langle v, r \rangle| \geq \varepsilon \|v\|] = \Pr[|r_1| \geq \varepsilon] \geq 1 - \varepsilon,$$

$$\Pr[\langle v, r \rangle \in [0, \varepsilon] \|v\|] = \Pr[r_1 \in [0, \varepsilon]] \in [\varepsilon/8, \varepsilon/2],$$

where $\varepsilon \in [0, 1]$.

Proof: As the distribution of r is spherically symmetric ([Ren70] Theorem **IV**.16.3), we may assume that v is the vector $(\|v\|, 0, \ldots, 0)$, thus $\langle v, r \rangle$ is exactly $r_1 \|v\|$. □

Lemma 4.2 *Let (X, Y) be the partition obtained after step (1) of algorithm A_{Cut}. Let i be an* **all bad** *vertex of degree d. With probability at least $\Omega(1/d^2)$ it is the case that* **more** *than half of the vertices in $N(i)$ lie on the same side of the partition (X, Y) as the vertex i.*

Proof: Let $N(i) = \{u_1, \ldots, u_d\}$ and denote the vector v_i corresponding to the vertex i as v. As the vectors $v, \{u_1, \ldots, u_d\}$ lie in a common $d+1$ dimensional space, the following representation of $v, \{u_1, \ldots, u_d\}$ may be assumed without loss of generality. Let $v = (1, 0, \ldots, 0) \in R^{d+1}$ and $u_j = (\alpha_j, \beta_j, \gamma_j)$ for $j \in [d]$. Where $\alpha_j \in R$, $\beta_j \in R$, $\gamma_j \in R^{d-1}$, and α_j, β_j, γ_j are vectors in mutually orthogonal subspaces. Furthermore, we may assume for the vector u_1 that $\gamma_1 = 0$. Let $r = (r_1, \ldots, r_{d+1})$ be a $d+1$ dimensional random variable representing the normal to a random hyperplane, *i.e* every r_i is an independent standard normal random variable. Let r be defined as $(\alpha_r, \beta_r, \gamma_r)$ where $\alpha_r = r_1$, $\beta_r = r_2$, and $\gamma_r = (r_3, \ldots, r_{d+1})$.

In general we intend to prove that with probability $\delta = \Omega(1/d^2)$ (over r), the vector v and more than half its neighbors lie on the same side of the hyperplane corresponding to r. Let δ_1 be the probability that the vector v lies very *close* but *above* the random hyperplane corresponding to r, *i.e.* the inner product $\langle v, r \rangle$ is small but positive. Note that this probability is dependent on r_1 alone, and corresponds to a positive r_1 of low magnitude. Conditioning on such r (*i.e* r_1), we are interested in the probability that more than half of the vectors in $N(i)$ also lie above the hyperplane corresponding to r. As r_1 is small, this is roughly the probability that more than half of the projected vectors $(\beta_j, \gamma_j) \in R^d$ for $j \in [d]$ will be above the random hyperplane corresponding to the random vector (r_2, r_3).

If the number of vectors in $N(i)$ is odd (*i.e.* d is odd), then the above will happen with probability 1/2, and we may conclude that δ is roughly $\delta_1/2$. Otherwise we condition on an additional projection. Let δ_2 be the probability that the vector u_1 is also very close but above a random hyperplane corresponding to a random vector r. As before this probability is dependent on r_1 and r_2 alone (recall that $u_1 = (\alpha_1, \beta_1, 0)$). Now conditioning the random vector r on r_1 and r_2 (in order to assure that both v and u_1 lie close to and above the random hyperplane corresponding to r), we are interested in the probability that more than half of the remaining vectors in $N(i)$ also lie above the hyperplane corresponding to r. As above, due to the fact that r_1 and r_2 are of small magnitude this probability is roughly the probability that more than half of the projected vectors $\gamma_j \in R^{d-1}$ for $j \in \{2, \ldots, d\}$ will be above the random hyperplane corresponding to the random vector r_3. We are now in a case similar to the case in which d is odd, and conclude that δ is roughly $\delta_1\delta_2/2$. Detailed proof follows.

The vertex i is **all bad**, thus for all $j \in [d]$ we have that the inner product $\langle v, u_j\rangle = \alpha_j$ equals $\cos(\theta_0) \simeq -0.688$, and that the vector $(\beta_j, \gamma_j) \in R^d$ satisfies $\|(\beta_j, \gamma_j)\| \simeq 0.725$ (for u_1 this implies that $\beta_1 \simeq 0.725$). For $i, j \in [d]$ the triangle constraints ($\langle v, u_i\rangle + \langle v, u_j\rangle + \langle u_i, u_j\rangle \geq -1$) of *(SDP-Cut)* yield $\langle u_i, u_j\rangle = \alpha_i\alpha_j + \beta_i\beta_j + \langle\gamma_i, \gamma_j\rangle \geq 0.376$. Hence $\beta_i\beta_j + \langle\gamma_i, \gamma_j\rangle \geq -0.097$ meaning that the angle between the vectors $(\beta_i, \gamma_i), (\beta_j, \gamma_j) \in R^d$ is at most 1.763 (bounded away from π).

Let $\varepsilon > 0$ be some small value which will be fixed later in the proof. By Lemma 4.1 we have that with probability at least $\varepsilon/16$ (over $\alpha_r \in R$) it is the case that $\langle v, r\rangle = \alpha_r \in [\varepsilon/8, \varepsilon]$, and with probability at least $\varepsilon/2$ (over $\beta_r \in R$) we have that $\beta_r \in [\varepsilon, 8\varepsilon]$ (we assume that $\varepsilon < 1/8$). We conclude that with probability at least $\frac{\varepsilon^2}{2^5}$ over the random vector r we have that both v and u_1 lie above the hyperplane corresponding to r.

Furthermore, considering a vector γ_j, we conclude using Lemma 4.1 that with probability at least $1 - \frac{1}{4d}$ (over γ_r) the value of $|\langle\gamma_j, \gamma_r\rangle|$ is at least $\frac{1}{4d}\|\gamma_j\|$. Hence with probability at least 3/4 (over γ_j) we have for all vectors γ_j that $|\langle\gamma_j, \gamma_r\rangle| \geq \frac{1}{4d}\|\gamma_j\|$.

Finally, consider the probability over γ_r that **more** than half the inner products $\langle\gamma_j, \gamma_r\rangle$ are non-negative ($j \in [d]$). As described above, this probability is at least half, due to the fact that for any γ_r we have $\langle\gamma_1, \gamma_r\rangle = 0$. We conclude, that with probability 1/4 over γ_r, we are in the case in which $\langle\gamma_r, \gamma_j\rangle \geq \frac{1}{4d}\|\gamma_j\|$ for **more** than half of the vectors γ_j. Denote the set of these vectors as $\hat{N}(i)$. Note that $u_1 \in \hat{N}(i)$.

Let r be a random vector $r = (\alpha_r, \beta_r, \gamma_r)$ with $\alpha_r,\ \beta_r,\ \gamma_r$ as above (this happens with probability at least $\frac{\varepsilon^2}{2^7}$). We now claim that choosing an appropriate ε, we obtain for all vectors u_j in $\hat{N}(i)$ that the inner product $\langle u_j, r\rangle > 0$, as $|\hat{N}(i)| > d/2$ this completes the proof of our lemma.

Let $u_j = (\alpha_j, \beta_j, \gamma_j)$ be some vector in $\hat{N}(i)$. If $\beta_j \geq 0.7$ then we have that

$$\langle u_j, r\rangle = \alpha_j\alpha_r + \beta_j\beta_r + \langle\gamma_j, \gamma_r\rangle \geq (\beta_j - 0.688)\varepsilon + \frac{1}{4d}\|\gamma_j\| > 0.$$

Otherwise assume that for $u_j \in \hat{N}(i)$ we have $\beta_j < 0.7$. As we have shown earlier, $\|(\beta_j, \gamma_j)\| \simeq 0.725$ for all vectors in $N(i)$, thus we conclude that in this case $\|\gamma_j\|$ must be at least 1/8. Furthermore, for any vector $u_j \in N(i)$ the inner product $\langle u_1, u_j\rangle = \alpha_1\alpha_j + \beta_1\beta_j$ is at least 0.376, thus implying that β_j must be at least -0.14 (recall that $u_1 = (\alpha_1, \beta_1, 0)$). We conclude that

$$\langle u_j, r\rangle = \alpha_j\alpha_r + \beta_j\beta_r + \langle\gamma_j, \gamma_r\rangle \geq (-1.12 - 0.688)\varepsilon + \frac{1}{4d}\|\gamma_j\| > \frac{1}{2^5 d} - 2\varepsilon.$$

Setting ε to be less than $\frac{1}{2^6 d}$ the above inner product is positive.

All in all, the proof of our lemma is complete with $\delta = \frac{\varepsilon^2}{2^7} = \Omega(1/d^2)$. $\square$

Theorem 4.3 *There exists a semidefinite based algorithm that for every $\Delta > 0$ approximates the Max-Cut problem on graphs with bounded degree Δ within an expected approximation ratio of $\alpha + \Omega(1/\Delta^4)$.*

Proof: Consider the optimal vector configuration $\{v_1, \ldots, v_n\}$, and the partition (X, Y) achieved after step *(1)* of algorithm A_{Cut}. Let Z be the value of the above optimal vector configuration, and $w(X)$ the weight of the partition (X, Y). Using the analysis of [GW95] it can be seen that the expected value of $w(X)$ is at least αZ. Define an edge e_{ij} to be *bad* if the inner product $\langle v_i, v_j\rangle$ is *close* to $\cos(\theta_0) \simeq -0.688$, *i.e.* $\langle v_i, v_j\rangle \in [-0.688-0.01, -0.688+0.01]$.

Let Z_{good} be the contribution of the edges in G which are good (*i.e.* not bad) to the value Z defined above, and let $\varepsilon > 0$ be some small constant that will be determined later. We consider two cases. If Z_{good} is greater than εZ, using the standard analysis of [GW95], we have that

$$E[w(X)] \geq \alpha(Z - Z_{good}) + \left(\alpha + \frac{1}{2^{12}}\right) Z_{good} \geq \left(\alpha + \frac{\varepsilon}{2^{12}}\right) Z.$$

Otherwise, the contribution of the good edges, Z_{good}, is less that εZ. In such a case consider the graph $\hat{G}$ obtained by removing these good edges from G. In $\hat{G}$ all vertices are all bad and have degree at most Δ. Furthermore, $\hat{G}$ has at least $\frac{(1-\varepsilon)Z}{\Delta}$ vertices of degree ≥ 1. Using Lemma 4.2, we have that each such vertex in $\hat{G}$ is misplaced in (X, Y) (with respect to $\hat{G}$) with probability at least $\delta = \Omega(1/\Delta^2)$. Note that in Lemma 4.2 we dealt with the case in which e_{ij} was assumed to be bad if $\langle v_i, v_j\rangle$ was exactly $\cos(\theta_0) \simeq -0.688$. Our analysis in Lemma 4.2 was slack, thus it can be seen that we obtain the same results for the above definition of a bad edge as well. As each vertex moved may effect at most Δ other vertices, we conclude that running the greedy step of our algorithm A_{Cut} on $\hat{G}$ has an expected contribution of $\frac{(1-\varepsilon)Z}{2^{19}\Delta^3(\Delta+1)}$ edges. Therefore in such a case, we obtain a final cut of $\hat{G}$ (which is also a cut of G) of expected value at least $\left(\alpha + \frac{1}{2^{19}\Delta^3(\Delta+1)}\right)(1-\varepsilon)Z$. Setting ε to be $\frac{1}{2^{20}\Delta^3(\Delta+1)}$ our proof is complete. $\square$

5 Concluding remarks

Throughout our work, we consider the unweighted version of the Max-Cut problem. Our analysis can be extended to graphs with edge weights as well, although the results we obtain are weaker. In Section 3, when considering graphs without edge weights, a vertex i is said to be misplaced if any triplet of vertices (i, j, k) (where j and k are adjacent to i) lay on the same side of the cut. In the case in which we allow edge weights, a vertex i and two of its neighbors may be on the same side of the cut without i being misplaced, thus in this case we consider only the triplet (i, j, k) in which the edges e_{ij} and e_{ik} are the heaviest edges adjacent to i. In Section 4, for every vertex v regardless of its degree d, use the part of Lemma 4.2 addressing the case in which the degree d is even, and choose the neighbor u_1 of v to be such that the edge (v, u_1) is the heaviest edge adjacent to v.

In general, when considering graphs with edge weights, we stop the greedy placement step after a polynomial number of steps and not necessarily at a local optimum. Our extended analysis holds under this additional restriction.

In a recent paper Halperin, Livnat, and Zwick [HLZ01] improve over the approximation ratio of 0.921 that we present for graphs of degree at most three (without edge weights), and obtain a ratio of 0.9326. Their algorithm follows the general framework we present (the addition of a local improvement step to the GW algorithm) and differs from ours in the greedy procedure analyzed.

Acknowledgments

We thank Monique Laurent for commenting on an early version of our paper. The first author is the Incumbent of the Joseph and Celia Reskin Career Development Chair. This research was supported in part by a Minerva grant, and by DFG grant 673/4-1, Esprit BR grants 7079, 21726, and EC-US 030, and by the Max-Planck Research Prize.

References

[BK98] P. Berman and M. Karpinski. On some tighter inapproximability results, further improvements. *ECCC*, TR98-065, 1998. Extended abstract appears in *ICALP* 1999, pages 200-209.

[BM86] F. Barahona and A.R. Mahjoub. On the cut polytope. *Mathematical Programming*, 36:157–173, 1986.

[FG95] U. Feige and M.X. Goemans. Approximating the value of two prover proof systems with applications to Max-2-Sat and Max-Dicut. *Proc. of the 3rd Israel Symposium on Theory of Computing and Systems*, pages 182–189, 1995.

[FS01] U. Feige and G. Schechtman. On the optimality of the random hyperplane rounding technique for MAX CUT. *Proc. 33rd Annual ACM Symposium on the Theory of Computing*, pages 433–442, 2001.

[GW95] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of ACM*, 42:1115–1145, 1995.

[HLZ01] E. Halperin, D. Livnat, and U. Zwick. Max-Cut in cubic graphs. *Manuscript*, 2001.

[Kar96] H. J. Karloff. How good is the Goemans-Williamson MAX CUT algorithm? *In Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 427–434, 1996.

[KZ97] H. Karloff and U. Zwick. A 7/8-approximation algorithm for Max-3-Sat? *In Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science*, pages 406–415, 1997.

[Nes98] Y. E. Nesterov. Semidefinite relaxation and nonconvex quadratic optimization. *Optimization Methods and Software*, 9:141–160, 1998.

[Pol95] S. Poljak. Local search for Max-Cut. *SIAM Journal on Computing*, 24:822–839, 1995.

[Ren70] A. Renyi. Probability theory. *Elsevier, New York*, 1970.

[TFMP99] T.Calamoneri, I. Finocchi, Y. Manoussakis, and R. Petreschi. Approximation algorithm for the Max Cut problem on cubic graphs. *Advances in Computing Science - ASIAN'99, Lect. Notes in Comp. Science 1742, Springer Verlag Ed.*, pages 27–36, 1999.

[Zwi99] U. Zwick. Outward rotations: a new tool for rounding solutions of semidefinite programming relaxations, with application to Max-Cut and other problems. *In Proceedings of the 31th ACM Symposium on Theory of Computing*, pages 679–687, 1999.