

# MAX CUT in cubic graphs

Eran Halperin \*

Dror Livnat \*

Uri Zwick \*

## Abstract

We present an improved semidefinite programming based approximation algorithm for the MAX CUT problem in graphs of maximum degree at most 3. The approximation ratio of the new algorithm is at least 0.9326. This improves, and also somewhat simplifies, a result of Feige, Karpinski and Langberg. We also observe that results of Hopkins and Staton and of Bondy and Locke yield a simple combinatorial  $\frac{4}{5}$ -approximation algorithm for the problem. Slightly improved results would appear in the full version of the paper.

## 1 Introduction

MAX CUT is one of the most basic, and most studied, combinatorial optimization problems. In a seminal paper, Goemans and Williamson [GW95] used semidefinite programming to obtain a 0.87856-approximation algorithm for the problem. Feige, Karpinski and Langberg [FKL00] considered the MAX CUT problem in bounded degree graphs and obtained, among other results, a 0.921-approximation algorithm for MAX CUT problem in graphs of maximum degree 3. (They also obtained an improved 0.924-approximation algorithm for cubic graphs, i.e., graphs in which the degree of all vertices is 3.) Berman and Karpinski [BK99] showed that no approximation algorithm may achieve an approximation ratio of 0.997 for the MAX CUT problem in cubic graphs, unless  $P=NP$ . Approximation algorithms for bounded occurrence constraint satisfaction problems were also considered by Håstad [Hås00].

In this paper, we obtain an improved approximation algorithm for the MAX CUT problem in graphs of maximum degree at most 3. Our algorithm is, in fact, an approximation algorithm for the more general MAX 2-XOR problem in graphs of maximum degree at most 3. Håstad [Hås97] (see also Trevisan *et al.* [TSSW00]) show that obtaining an approximation ratio of  $\frac{16}{17} + \epsilon$  for MAX CUT, and a ratio of  $\frac{11}{12} + \epsilon$  for MAX 2-XOR,

for any  $\epsilon > 0$ , are both NP-hard tasks. As  $\frac{11}{12} \simeq 0.9167$ , our approximation algorithm shows that approximating general instances of MAX 2-XOR is provably more difficult than approximating instances of MAX 2-XOR with maximum degree at most 3.

An instance of the MAX CUT problem is an unweighted undirected graph  $G = (V, E)$ . The goal is to find a subset  $S \subseteq V$  of the vertices such that the size of the cut  $\delta(S) = \{(u, v) \in E \mid u \in S \text{ and } v \notin S\}$  defined by  $S$  is maximized. We consider a slightly more general problem called MAX 2-XOR. An instance of MAX 2-XOR is an undirected graph  $G = (V, E_{\neq}, E_{=})$  with two types of edges. Edges of  $E_{\neq}$  are called *inequality* edges, while edges of  $E_{=}$  are called *equality* edges. The goal is to find a set  $S \subseteq V$  such that the size of the ‘cut’  $\delta(S) = \{(u, v) \in E_{\neq} \mid u \in S \text{ and } v \notin S\} \cup \{(u, v) \in E_{=} \mid u, v \in S \text{ or } u, v \notin S\}$  is maximized. Clearly an instance  $G = (V, E)$  of MAX CUT is equivalent to the instance  $(V, E, \phi)$  of MAX 2-XOR. The consideration of this more general problem allows us to *simplify* the description and analysis of our algorithm.

Our algorithm follows the general framework set out by Feige *et al.* [FKL00]. It starts by solving the semidefinite programming relaxation of the MAX 2-XOR, enhanced by the triangle constraints. It rounds a solution of the relaxation using a random hyperplane. Finally, it uses a local improvement step in which vertices that were ‘misplaced’ by the random hyperplane are moved to the other side of the cut.

Our algorithm and its analysis differ from that of Feige *et al.* [FKL00] in the following respects: (i) Our algorithm is somewhat simpler, even though it works on a more general problem. (ii) Our analysis is tighter as it considers a vertex together with *all* its neighbors; (iii) The presence of vertices of degree 2 increases, rather than decreases, the approximation ratio obtained by our algorithm.

Our algorithm is composed of a simple preprocessing step that converts every input graph of maximum degree 3 into a *reduced* cubic graph. (See definition in the next section.) Our approximation algorithm itself works on reduced cubic graphs. It is then easy to convert the cut obtained in the reduced cubic graph into a cut of the original graph without decreasing the

—\*School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel. E-mail: {heran, dror, zwick}@tau.ac.il. This research was supported by the ISRAEL SCIENCE FOUNDATION (grant no. 246/01).

approximation ratio.

The design of a local improvement step that combines well with the semidefinite programming approach leads us also to the question of obtaining purely combinatorial approximation algorithms for the MAX CUT problem in graphs of maximum degree at most 3. We observe that results of Hopkins and Staton [HS82] and of Bondy and Locke [BL86] yield a simple combinatorial  $\frac{4}{5}$ -approximation algorithm for the problem.

The rest of this paper is organized as follows. In the next section we describe the simple reduction to reduced cubic graphs. In Section 3 we describe the approximation algorithm for reduced cubic graphs. In Section 4 we discuss combinatorial approximation algorithms for the problem. In Section 5 we mention some improved results that would appear in the full version of the paper. We end in Section 6 with some concluding remarks and open problems.

## 2 The reduction

In this section we describe a simple way of reducing the problem of approximating MAX 2-XOR in graphs of maximum degree 3 into the problem of approximating MAX 2-XOR in *reduced cubic* graphs:

**DEFINITION 2.1.** (REDUCED CUBIC GRAPHS) *A connected graph  $G = (V, E_{\neq}, E_{=})$  is a reduced cubic graph if and only if (i)  $\deg(v) = 3$ , for every  $v \in V$ ; (ii)  $\deg_{=}(v) \leq 1$ , for every  $v \in V$ ; (iii) Every vertex  $v \in V$  is contained in at most one triangle.*

In the above definition, we let  $\deg(v)$  be the degree of a vertex  $v \in V$  in the graph  $(V, E_{\neq} \cup E_{=})$ , and  $\deg_{=}(v)$  be its degree in the graph  $(V, E_{=})$ . In the sequel, we use  $u - v \in G$  to denote the fact that  $(u, v) \in E_{\neq} \cup E_{=}$ . This notation does not specify whether the edge is in  $E_{\neq}$  or in  $E_{=}$ . Also, when we draw a graph  $G = (V, E_{\neq}, E_{=})$ , a line connecting vertex  $u$  to vertex  $v$  indicates that  $u - v \in G$ , again without specifying the type of the edge.

We make use of the following simple lemma whose proof is obvious:

**LEMMA 2.1.** *Let  $\mathcal{F}$  and  $\mathcal{F}'$  be two families of graphs. If we can efficiently convert any  $G \in \mathcal{F}$  into a graph  $G' \in \mathcal{F}'$  such that  $OPT(G) = OPT(G') + c$ , where  $c \geq 0$ , and such that we can efficiently convert any cut of  $G'$  of value  $Z$  into a cut of  $G$  of value  $Z + c$ , then an  $\alpha$ -approximation algorithm for  $\mathcal{F}'$  immediately extends to an  $\alpha$ -approximation algorithm for  $\mathcal{F}$ .*

The algorithm for converting an arbitrary graph of maximum degree 3 into a reduced cubic graph is given

in Figure 1. The algorithm is extremely simple and intuitive. Steps 1-4 of the algorithm are repeatedly performed until none of them can be applied.

Step 1 of the algorithm gets rid of vertices of degree 1. When a vertex of degree 1 is removed, the optimum clearly decreases by exactly 1. Step 2 of the algorithm gets rid of paths of degree 2 vertices. When a path of length  $k + 1$  is replaced by an appropriate equality or inequality edge, the size of the optimum decreases by exactly  $k$ . The most complicated step is step 3 that gets rid of vertices that participate in two triangles. It is not difficult to check that if a vertex  $u$  participates in two triangles, there must also be an edge  $u - v \in G$  that participates in two triangles. Let  $x$  and  $y$  be the other vertices in these two triangles. If  $x - y \in G$ , then  $x, y, u, v$  form a connected component of the graph, and the graph is either not connected, or composed of only these four vertices. We assume that this is not the case. Let  $x'$  and  $y'$  be the other neighbors of  $x$  and  $y$ . (Note that in this stage the graph is already cubic.) If  $x' = y'$ , we can simply remove  $u, v, x, y, x' = y'$  from the graph. Otherwise, we remove  $u, v, x, y$  from the graph and replace them by an appropriate edge between  $x'$  and  $y'$ . It is not difficult to check that this decreases the optimum by a small predictable constant. Step 4 is similar to step 3 and gets rid of parallel edges. Finally, step 5, makes sure that each vertex has at most one equality edge adjacent to it.

Given a cut of the graph  $G'$  produced by this process, it is easy to undo the changes, in reversed order, and increase the size of the cut by the required amount.

## 3 The approximation algorithm

We now present our semidefinite programming based approximation algorithm for the MAX 2-XOR problem in reduced cubic graphs. The approximation algorithm is composed of three main steps:

1. Solving the SDP relaxation given in Figure 2.
2. Rounding the solution using a random hyperplane.
3. Performing a local improvement step.

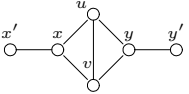
The SDP relaxation given in Figure 2 is a natural extension of the relaxation of MAX CUT in graphs of degree at most 3 used by Feige *et al.* [FKL00]. (We assume there, without loss of generality, that  $V = \{1, 2, \dots, n\}$ .) The rounding step is identical to the rounding step of Goemans and Williamson [GW95]. Our local improvement step, and its analysis, are different from those used by Feige *et al.* [FKL00]. A full description of the algorithm is given in Figure 3.

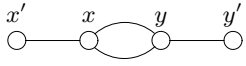
### ALGORITHM REDUCE

**Input:** A graph  $G = (V, E_{\neq}, E_{=})$  of maximum degree at most 3.

**Output:** A reduced cubic graph  $G' = (V, E'_{\neq}, E'_{=})$ .

0. Let  $G' \leftarrow (V, E_{\neq}, E_{=})$ .
1. If  $v \in G'$  is a vertex of degree 1, then remove it from  $G'$ .
2. If  $x - v_1 - v_2 - \dots - v_k - y \in G'$ , and  $v_1, v_2, \dots, v_k$  are of degree 2 while  $x, y$  are of degree 3, then remove  $v_1, v_2, \dots, v_k$  from  $G'$  and add an inequality edge  $(x, y)$  to  $E'_{\neq}$ , if the number of inequality edges on the path from  $x$  to  $y$  is odd, and an equality edge  $(x, y)$  to  $E'_{=}$ , otherwise.

3. If   $\in G'$ , then remove  $u, v, x, y$  from  $G'$ . If  $x' = y'$ , then remove  $x'$  from  $G'$ . If in *any* optimal solution of the small subproblem induced by  $u, v, x, y, x', y'$ , the vertices  $x'$  and  $y'$  are on opposite sides of the cut, add an inequality edge  $(x', y')$  to  $E'_{\neq}$ . If in *any* optimal solution of the small subproblem induced by  $u, v, x, y, x', y'$ , the vertices  $x'$  and  $y'$  are on the same side of the cut, add an equality edge  $(x', y')$  to  $E'_{=}$ . If there are optimal solutions of both types, do not add any edge.

4. If   $\in G'$ , then remove  $x, y$  from  $G'$ . If  $x' = y'$ , then remove  $x'$  from  $G'$ . Add an appropriate edge connecting  $x'$  and  $y'$ , if necessary, as above.

5. If  $v \in G'$  has more than one equality edge incident on it, make the equality edges incident on  $v$  inequality edges, and vice versa.

Figure 1: Converting a graph of maximum degree 3 into a reduced cubic graph.

The local improvement step used by the algorithm is very simple. We let  $V_i$  be the set of vertices such that  $i$  of their edges are unsatisfied by the current cut  $\delta(S)$ . Clearly, if a vertex  $v \in V_3$  is moved to the other side of the cut, the number of edges in the cut increases by 3, while if a vertex  $v \in V_2$  is moved, the number of edges cut increases by 1. Each such move is clearly beneficial. In fortunate situations, we can increase the cut using such local improvement steps by  $3|V_3| + |V_2|$ . Usually, however, we have to settle for a smaller improvement, as each move may destroy several other potential moves. For example, if  $v \in V_3$  has a neighbor  $u \in V_3$ , then when  $v$  is moved to the other side of the cut,  $v$  is moved from  $V_3$  to  $V_0$ , while  $u$  is moved from  $V_3$  to  $V_2$ . To maximize the improvement obtained, we should choose the sequence of improvement steps with some care.

Our algorithm performs the improvement steps in the following order. Whenever there is a vertex with three unsatisfied edges incident to it, i.e.,  $V_3 \neq \phi$ , the algorithm chooses such a vertex with the minimum number of neighbors in  $V_3$  and flips it, i.e., moves it

to the other side of the cut. When there are no such vertices left, the algorithm looks for unsatisfied paths or cycles of  $V_2$  vertices. The algorithm then flips every second vertex on such a path or cycle. These steps are illustrated in Figure 4.

We now turn to the analysis of the algorithm. To illustrate the ideas used in the analysis in the simplest setting, we first consider the performance of the algorithm on triangle-free graphs. We then show how to extend the analysis to graphs that may contain triangles. The performance of the algorithm actually improves in the presence of triangles, but the analysis is a bit more complicated. (The presence of triangles also enhances the performance of MAX CUT algorithms in general graphs, see Zwick [Zwi99].)

### 3.1 Analysis of the algorithm on triangle-free cubic graphs

We begin with the following combinatorial lemma:

LEMMA 3.1. *Let  $G = (V, E_{\neq}, E_{=})$  be a triangle-free*

$$\begin{array}{l}
\text{Maximize} \quad \sum_{(i,j) \in E_{\neq}} \frac{1 - v_i \cdot v_j}{2} + \sum_{(i,j) \in E_{=}} \frac{1 + v_i \cdot v_j}{2} \\
\quad v_i \cdot v_j + v_i \cdot v_k + v_j \cdot v_k \geq -1 \quad , \quad 1 \leq i, j, k \leq n \\
\quad v_i \cdot v_j - v_i \cdot v_k - v_j \cdot v_k \geq -1 \quad , \quad 1 \leq i, j, k \leq n \\
\quad v_i \cdot v_j + v_i \cdot v_k + v_j \cdot v_k = -1 \quad , \quad (i, j), (j, k) \in E_{\neq} \\
\quad -v_i \cdot v_j - v_i \cdot v_k + v_j \cdot v_k = -1 \quad , \quad (i, j), (j, k) \in E_{=} \\
\quad v_i \cdot v_j - v_i \cdot v_k - v_j \cdot v_k = -1 \quad , \quad (i, j) \in E_{\neq}, (i, k) \in E_{=} \\
\quad v_i \in \mathbb{R}^n, \|v_i\| = 1 \quad , \quad 1 \leq i \leq n
\end{array}$$

Figure 2: A semidefinite programming relaxation of MAX 2-XOR in a graph of maximum degree 3.

*cubic graph.* Let  $S \subseteq V$  be a cut of  $G$ , and let  $V_2$  and  $V_3$  be the corresponding sets of vertices with two and three unsatisfied adjacent edges. Then, the local improvement step of algorithm CUBIC-MAX-2-XOR increases the size of the cut by at least  $\frac{2}{5}|V_2| + \frac{17}{15}|V_3|$ .

*Proof:* We prove the lemma by induction on the size of the cut  $\delta(S)$ . If  $S$  is an optimal cut, then  $V_2 = V_3 = \phi$ , and the claim holds. We suppose, therefore, that the claim holds for all cuts that are larger than  $S$ , and show that the claim also holds for  $S$ . We assume, at first, that  $S \neq \phi$  and  $S \neq V$ . We show later how to remove this assumption.

If  $V_3 \neq \phi$ , then the algorithm chooses, in Step 3(a), a vertex  $v \in V_3$  that has the smallest number of neighbors in  $V_3$ , and moves it to the other side of the cut. As  $S \neq \phi$  and  $S \neq V$ , the vertex  $v \in V$  has at most two neighbors in  $V_3$ . Assume, therefore, that  $u_1, u_2$  and  $u_3$  are the neighbors of  $v$  and that  $u_1, u_2 \in V_3$  and  $u_3 \in V_2$ . The other cases are easier. When  $v$  is moved to the other side of the cut, the size of the cut increases by 3. The vertex  $v$  is removed from  $V_3$ . The vertices  $u_1$  and  $u_2$  are moved from  $V_3$  to  $V_2$ , and the vertex  $u_3$  is removed from  $V_2$ . By the induction hypothesis, the local improvement step could still improve the size of the cut by at least  $\frac{2}{5}(|V_2| + 1) + \frac{17}{15}(|V_3| - 3)$ . All together, we get an improvement of at least

$$3 + \frac{2}{5}(|V_2| + 1) + \frac{17}{15}(|V_3| - 3) \geq \frac{2}{5}|V_2| + \frac{17}{15}|V_3| ,$$

as required.

If  $V_3 = \phi$ , the algorithm chooses a path or a cycle of  $V_2$  vertices. We consider the case of a cycle of an odd size. The other cases are easier. Let  $v_1 - v_2 - \dots - v_{2k+1} - v_1 \in G$  be a cycle, where  $v_1, v_2, \dots, v_{2k+1} \in V_2$ , all whose edges are unsatisfied by  $S$ . As the graph contains no triangles, we have  $k \geq 2$ . Let  $u_i$ , for  $1 \leq i \leq 2k+1$ , be the third neighbor of  $v_i$ . The edges  $v_i - u_i \in G$  are satisfied by  $S$ .

The algorithm moves  $v_2, v_4, \dots, v_{2k}$  to the other side of the cut. (Consult Figure 4(c) that depicts the case  $k = 2$ .) The size of the cut increases by  $k$ . The vertices  $v_1, v_2, \dots, v_{2k+1}$  are removed from  $V_2$ . The number of unsatisfied edges incident on  $u_i$ , where  $1 \leq i \leq 2k+1$ , does not decrease. Thus, none of the  $u_i$ 's is removed from  $V_2$ . (Some may join  $V_2$  or  $V_3$ , which is only better for us.) By the induction hypothesis, the local improvement step would further increase the size of the cut by at least  $\frac{2}{5}(|V_2| - (2k+1))$ . All together, we get an increase of at least

$$k + \frac{2}{5}(|V_2| - (2k+1)) \geq \frac{2}{5}|V_2| ,$$

as  $k \geq 2$ .

Finally, we have to handle the case  $S = \phi$  or  $S = V$ . In this case,  $V_3 = V$ , and the *first*  $V_3$  vertex moved to the other side would have three  $V_3$  neighbors. This, however, is compensated by the fact that the *last*  $V_3$  vertex moved has at most one  $V_3$  neighbor.  $\square$

We note that the coefficient  $\frac{2}{5}$  of  $V_2$  vertices in the above lemma cannot be improved. The coefficient  $\frac{17}{15}$  of  $V_3$  vertices can be improved a bit, by making the algorithm choose the  $V_3$  vertices to be moved more carefully. However, the algorithm and its analysis become more complicated. Details would appear in the full version of the paper. Furthermore, this has only a negligible effect on the performance ratio of the whole algorithm, as most of the improvement comes, as we shall see, from  $V_2$  rather than  $V_3$  vertices.

Let  $v_1, v_2, \dots, v_n$  be an (almost) optimal solution of the SDP relaxation of the MAX 2-XOR instance  $G = (V, E_{\neq}, E_{=})$ . Let  $\theta_{ij} = \arccos(v_i \cdot v_j)$  be the angle between  $v_i$  and  $v_j$ . Clearly, the probability that an edge  $(i, j) \in E_{\neq}$  is satisfied by the cut generated by the random hyperplane is  $\theta_{ij}/\pi$ , and the probability that an edge  $(i, j) \in E_{=}$  is satisfied by this cut is  $(\pi - \theta_{ij})/\pi$ . Let  $p_2(i)$  be the probability that  $i \in V_2$ , and let  $p_3(i)$  be

**ALGORITHM** CUBIC-MAX-2-XOR

**Input:** A reduced cubic graph  $G = (V, E_{\neq}, E_{=})$ .

**Output:** A large cut of  $G$ .

1. Solve the SDP relaxation of Figure 2.

2. Rounding:

(a) Choose a random  $n$ -dimensional vector  $r$ . Let  $S = \{i \mid v_i \cdot r \geq 0\}$  and  $\bar{S} = V - S$ .

(b) For each vertex  $v \in V$ , let  $n(v)$  be the number of unsatisfied edges adjacent to  $v$ , i.e., the number of edges adjacent to  $v$  that are *not* in  $\delta(S)$ . Let  $V_i = \{v \in V \mid n(v) = i\}$ . (In the next step, vertices are moved from one side of the cut to the other. It is assumed, then, that the sets  $V_i$  are automatically updated.)

3. Local improvement:

Repeat the following steps until none is applicable.

Give precedence to step (a).

(a) If  $V_3 \neq \emptyset$ , choose a vertex  $v \in V_3$  with the smallest number of neighbors in  $V_3$  and move  $v$  to the other side of the cut. (I.e.,  $S \leftarrow S \oplus \{v\}$ .)

(b) If  $u - v_1 - v_2 - \dots - v_k - w \in G$  is a path of unsatisfied edges, where  $v_1, v_2, \dots, v_k \in V_2$  while  $u, w \notin V_2$ , then move  $v_i$ , for  $i$  odd, to the other side of the cut.

(c) If  $v_1 - v_2 - \dots - v_k - v_1 \in G$  is a cycle of unsatisfied edges, where  $v_1, v_2, \dots, v_k \in V_2$ , then move  $v_i$ , for  $i$  even, to the other side of the cut.

Figure 3: Algorithm CUBIC-MAX-2-XOR.

the probability that  $i \in V_3$ , with respect to the cut  $S$  generated by the random hyperplane. If the neighbors of  $i$  are  $j, k$  and  $\ell$ , and if  $(i, j), (i, k), (i, \ell) \in E_{\neq}$ , then

$$\begin{aligned} p_2(i) &= \text{prob}(v_i, v_j, v_k, -v_\ell) + \text{prob}(v_i, v_j, -v_k, v_\ell) \\ &\quad + \text{prob}(v_i, -v_j, v_k, v_\ell), \\ p_3(i) &= \text{prob}(v_i, v_j, v_k, v_\ell), \end{aligned}$$

where  $\text{prob}(v_i, v_j, v_k, v_\ell)$  is the probability that the vectors  $v_i, v_j, v_k$  and  $v_\ell$  all lie on the same side of a random hyperplane. If  $(v_i, v_\ell) \in E_{=}$ , then  $v_\ell$  in these expressions should be replaced by  $-v_\ell$ . Unfortunately, there is no simple way of expressing  $\text{prob}(v_i, v_j, v_k, v_\ell)$  in terms of the angles between the vectors  $v_i, v_j, v_k$  and  $v_\ell$ . The computation of  $\text{prob}(v_i, v_j, v_k, v_\ell)$  is equivalent to the computation of the volume of a tetrahedron defined by the vectors  $v_i, v_j, v_k, v_\ell$ . For more details, see Karloff and Zwick [KZ97] and Zwick [Zwi01].

Combining these facts with Lemma 3.1, and letting  $\beta_2 = \frac{2}{5}$  and  $\beta_3 = \frac{17}{15}$ , we get that the expected size of the cut produced by the algorithm is at least:

$$\begin{aligned} \text{EXP}(G) &= \sum_{(i,j) \in E_{\neq}} \frac{\theta_{ij}}{\pi} + \sum_{(i,j) \in E_{=}} \frac{\pi - \theta_{ij}}{\pi} \\ &\quad + \beta_2 \sum_{i \in V} p_2(i) + \beta_3 \sum_{i \in V} p_3(i). \end{aligned}$$

The upper bound on the size of the maximum cut supplied by the SDP relaxation is:

$$\text{SDP}(G) = \sum_{(i,j) \in E_{\neq}} \frac{1 - v_i \cdot v_j}{2} + \sum_{(i,j) \in E_{=}} \frac{1 + v_i \cdot v_j}{2}.$$

Clearly,  $\text{EXP}(G)/\text{SDP}(G)$  is a lower bound on the performance ratio of the algorithm on the instance  $G = (V, E_{\neq}, E_{=})$ . For every vertex  $i \in V$ , we now define:

$$\begin{aligned} \text{exp}(i) &= \sum_{j:(i,j) \in E_{\neq}} \frac{\theta_{ij}}{2\pi} + \sum_{j:(i,j) \in E_{=}} \frac{\pi - \theta_{ij}}{2\pi} \\ &\quad + \beta_2 p_2(i) + \beta_3 p_3(i), \end{aligned}$$

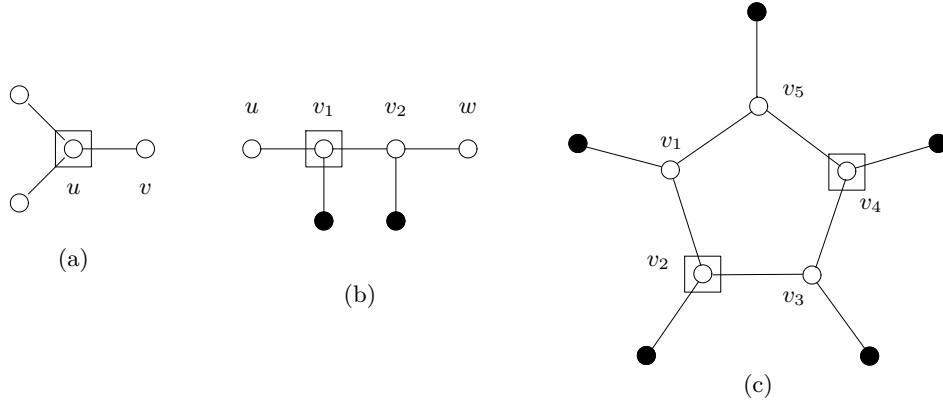


Figure 4: The local improvement steps used by algorithm CUBIC-MAX-2-XOR.

$$\text{sdp}(i) = \sum_{j:(i,j) \in E_{\neq}} \frac{1 - v_i \cdot v_j}{4} + \sum_{j:(i,j) \in E_{=}} \frac{1 + v_i \cdot v_j}{4}.$$

It is easy to check that

$$\text{EXP}(G) = \sum_{i \in V} \text{exp}(i) \quad , \quad \text{SDP}(G) = \sum_{i \in V} \text{sdp}(i).$$

To see this, note that half the contribution of each edge  $(i, j)$  appears in  $\text{exp}(i)$  and  $\text{sdp}(i)$ , and the other half in  $\text{exp}(j)$  and  $\text{sdp}(j)$ . Thus:

$$\frac{\text{EXP}(G)}{\text{SDP}(G)} = \frac{\sum_{i \in V} \text{exp}(i)}{\sum_{i \in V} \text{sdp}(i)} \geq \min_{i \in V} \frac{\text{exp}(i)}{\text{sdp}(i)}.$$

Finally, note that if the neighbors of  $i \in V$  are  $j, k$  and  $\ell$ , then the expressions  $\text{exp}(i)$  and  $\text{sdp}(i)$  depend only on the vectors  $v_i, v_j, v_k$  and  $v_\ell$ . We have numerically computed the minimum possible ratio  $\text{exp}(i)/\text{sdp}(i)$ , when  $v_i, v_j, v_k$  and  $v_\ell$  satisfy the constraints of the SDP relaxation, and found that it is at least 0.9326. The worst configuration is  $(\theta_{ij}, \theta_{ik}, \theta_{jk}, \theta_{i\ell}, \theta_{j\ell}, \theta_{k\ell}) \simeq (2.5682, 2.5682, 0.8229, 2.5682, 0.8229, 0.8229)$ . In this worst configuration we have  $p_2(i) = \Pr(i \in V_2) \simeq 0.1075$  and  $p_3(i) = \Pr(i \in V_3) \simeq 0.0157$ . The computation of the minimum ratio was carried out in Matlab, using simple adaptations of code that was used in Karloff and Zwick [KZ97] and Halperin and Zwick [HZ01].

### 3.2 Analysis of the algorithm on reduced cubic graphs

We now extend the analysis given above to reduced cubic graphs. Reduced cubic graphs may contain triangles, but each triangle in a reduced cubic graph is *isolated*. In other words, each vertex and each edge participate in at most one triangle. Furthermore, note

that due to condition (ii) of the definition of reduced cubic graphs (Definition 2.1), each triangle in a reduced cubic graph contains at most one equality edge. A triangle is said to be an *inequality triangle* if its three edges are inequality edges. We start with the following extension of Lemma 3.1:

**LEMMA 3.2.** *Let  $G = (V, E_{\neq}, E_{=})$  be a reduced cubic graph. Let  $T \subseteq V$  be the vertices of  $G$  that are contained in inequality triangles, and let  $\bar{T} = V - T$ . Let  $S \subseteq V$  be a cut of  $G$ , and let  $V_2$  and  $V_3$  be the corresponding sets of vertices with two and three unsatisfied adjacent edges. Then, the local improvement step of algorithm CUBIC-MAX-2-XOR increases the size of the cut by at least  $\frac{2}{5}|V_2 \cap \bar{T}| + \frac{17}{15}|V_3 \cap \bar{T}| + \frac{1}{3}|V_2 \cap T| + \frac{16}{17}|V_3 \cap T|$ .*

*Proof:* The proof of the lemma is a simple extension of the proof of Lemma 3.1 and it is omitted from this extended abstract.  $\square$

The definitions of  $\text{EXP}(G)$  and  $\text{SDP}(G)$  remain as in the previous sub-section. Also, if  $i \in \bar{T}$ , i.e., if  $i$  is not contained in a triangle, then  $\text{exp}(i)$  and  $\text{sdp}(i)$  also remain unchanged. But, if  $i$  is part of the triangle  $i - j - k - i \in G$ , and all the edges of this triangle are *inequality* edges, and if  $\ell$  is the other neighbor of  $i$ , we let:

$$\begin{aligned} \text{exp}(i) &= \frac{(\theta_{ij} + \theta_{jk} + \theta_{ki})}{3\pi} + \frac{\theta_{i\ell}}{2\pi} + \frac{1}{3}p_2(i) + \frac{16}{15}p_3(i), \\ \text{sdp}(i) &= \frac{3 - v_i \cdot v_j - v_j \cdot v_k - v_k \cdot v_i}{6} + \frac{1 - v_i \cdot v_\ell}{4} \\ &= \frac{2}{3} + \frac{1 - v_i \cdot v_\ell}{4}. \end{aligned}$$

The last equality follows from the fact that  $v_i \cdot v_j + v_j \cdot v_k + v_k \cdot v_i = -1$  is a constraint of the SDP relaxation.

**ALGORITHM COMB-CUBIC-MAX-CUT****Input:** A graph  $G = (V, E)$  of maximum degree at most 3.**Output:** A large cut.

1. Throw away all the triangles of  $G$  and their adjacent edges.
2. In the remaining triangle-free graph, find a cut that cuts at least  $4/5$  of the edges.
3. Add back the triangles and the pairs of glued triangles, one by one, and place their vertices optimally in the two sides of the cut.

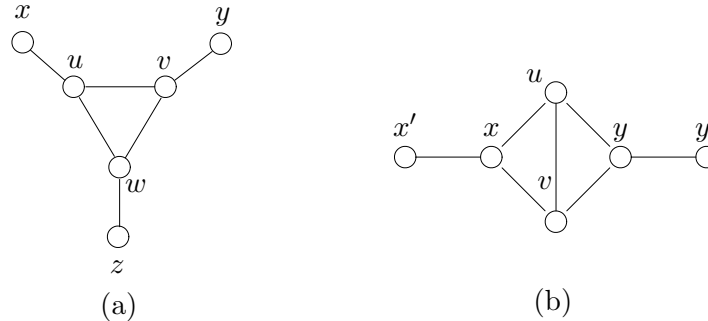
Figure 5: A combinatorial  $\frac{4}{5}$ -approximation algorithm for MAX CUT in graph of degree at most 3.

Figure 6: Triangles and glued pairs of triangles.

It is again easy to check that  $\text{EXP}(G) = \sum_{i \in V} \exp(i)$  and  $\text{SDP}(G) = \sum_{i \in V} \text{sdp}(i)$ . This time, the three edges of a triangle are split evenly among the three vertices of the triangle.

Numerically, we find that the minimum value of the ratio  $\exp(i)/\text{sdp}(i)$ , where  $i$  is in an inequality triangle, is at least 0.9541. The worst configuration is  $(\theta_{ij}, \theta_{ik}, \theta_{jk}, \theta_{il}, \theta_{jl}, \theta_{kl}) \simeq (2.5535, 0.5881, \pi, \pi, 0.5881, 2.5535)$ . Note that in this setting  $v_\ell = -v_i$ ,  $v_k = -v_j$  and  $p_2(i) = p_3(i) = 0$ . The worst ratio obtained for triangles is, therefore, much better than the worst ratio obtained for vertices that are not in triangles, and the performance ratio of the algorithm remains at least 0.9326 even in the presence of (isolated) triangles. Non-isolated triangles, and vertices of degree less than three were removed in the pre-processing step, so the approximation algorithm can be used to obtain an approximation ratio of at least 0.9326 on *any* MAX 2-XOR instance, and thus any MAX CUT instance of degree at most 3.

**4 Combinatorial approximation algorithms**

It is easy to see that any cubic graph with  $m$  edges has a cut of size at least  $2m/3$ . This result is tight for  $K_4$ , a clique on 4 vertices. Lemma 3.1 implies that any *triangle-free* cubic graph with  $m$  edges has a cut of size at least  $34m/45$ . Indeed, start with the cut  $S = \phi$  for which  $|V_3| = n$  and  $|V_2| = 0$ . The lemma then says that the local improvement step would produce a cut of size at least  $17n/15 = 34m/45$ . (In a cubic graph  $m = 3n/2$ .) This result can be further improved, as shown by Hopkins and Staton [HS82] and Bondy and Locke [BL86]:

LEMMA 4.1. ([HS82],[BL86]) *Let  $G = (V, E)$  be a triangle-free graph of maximum degree at most 3 with  $|E| = m$ . Then,  $G$  has a cut of size at least  $4m/5$ . Furthermore, such a cut can be found in polynomial time.*

Lemma 4.1 is tight for  $C_5$ , a cycle of length 5. Bondy and Locke [BL86] show that the *only* cubic graphs for which the lemma is tight are the *Petersen* graph, and the *dodecahedron* graph.

We remark that Lemma 4.1 cannot replace Lemma 3.1 (and Lemma 3.2) in the analysis of the semidefinite programming based approximation algorithm, as it is important there to be able to bound the improvement obtained by the local improvement step when it is applied on *any* cut that may be produced by the random hyperplane rounding step.

However, we can use Lemma 4.1 to obtain a very simple combinatorial  $\frac{4}{5}$ -approximation algorithm for the MAX CUT problem in graphs of degree at most 3. (We remark that unlike the previous section we consider here the MAX CUT problem, and not the more general MAX 2-XOR problem.)

**THEOREM 4.1.** *Algorithm COMB-CUBIC-MAX-CUT is  $\frac{4}{5}$ -approximation algorithm for the MAX CUT problem in graphs of degree at most 3.*

*Proof:* Triangles in a simple graph of degree at most 3 are isolated, as in Figure 6(a), or glued in pairs, as in Figure 6(b). The optimum cut can cut at most 5 of the edges of Figure 6(a). No matter on which side of the cut the vertices  $x, y$  and  $z$  were placed in step 2 of the algorithm, we can always place  $u, v$  and  $w$  such that 4 of these edges are cut. The case of glued triangles is even better. The optimum cut cuts at most 6 edges, and the cut produced would cut at least 5 edges.  $\square$

## 5 Further improvements

A slightly improved approximation ratio of 0.9328 for the MAX 2-XOR problem in graphs of maximum degree at most 3 can be obtained by dealing a bit differently with vertices of  $V_2$  that belong to odd cycles. It is also possible to slightly strengthen the semidefinite programming relaxation of the problem by adding *pentagon* constraints. More details would appear in the full version of the paper.

Using similar ideas it is also possible to obtain an improved ‘stand alone’ combinatorial approximation algorithm for the MAX CUT problem in *regular* cubic graphs. This algorithm is *not* based on the results of [HS82] and [BL86]. The approximation ratio achieved by this algorithm is  $\frac{22}{27} = 0.8148$ . More details would again appear in the full version of the paper.

## 6 Concluding remarks

We obtained an improved semidefinite programming based approximation algorithm for the MAX CUT problem in graphs of maximum degree at most 3. We also obtained simple combinatorial approximation algorithms for the MAX CUT problem in such graphs.

## References

- [BK99] P. Berman and M. Karpinski. On some tighter in-approximability results (extended abstract). In *Proceedings of the 26th International Colloquium on Automata, Languages and Programming, Prague, Czech Republic*, pages 200–209, 1999.
- [BL86] J.A. Bondy and S.C. Locke. Largest bipartite subgraphs in triangle-free graphs with maximum degree three. *Journal of Graph Theory*, 10:477–504, 1986.
- [FKL00] U. Feige, M. Karpinski, and M. Langberg. Improved approximation of Max-Cut on graphs of bounded degree. Technical report, E-CCC Report number TR00-043, 2000.
- [GW95] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42:1115–1145, 1995.
- [Hås97] J. Håstad. Some optimal inapproximability results. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El Paso, Texas*, pages 1–10, 1997. Full version available as E-CCC Report number TR97-037.
- [Hås00] J. Håstad. On bounded occurrence constraint satisfaction. *Information Processing Letters*, 74(1-2):1–6, 2000.
- [HS82] G. Hopkins and W. Staton. Extremal bipartite subgraphs of cubic triangle-free graphs. *Journal of Graph Theory*, 6:115–121, 1982.
- [HZ01] E. Halperin and U. Zwick. Approximation algorithms for MAX 4-SAT and rounding procedures for semidefinite programs. *Journal of Algorithms*, 40:184–211, 2001.
- [KZ97] H. Karloff and U. Zwick. A  $7/8$ -approximation algorithm for MAX 3SAT? In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, Miami Beach, Florida*, pages 406–415, 1997.
- [TSSW00] L. Trevisan, G.B. Sorkin, M. Sudan, and D.P. Williamson. Gadgets, approximation, and linear programming. *SIAM Journal on Computing*, 29:2074–2097, 2000.
- [Zwi99] U. Zwick. Outward rotations: a tool for rounding solutions of semidefinite programming relaxations, with applications to max cut and other problems. In *Proceedings of the 31th Annual ACM Symposium on Theory of Computing, Atlanta, Georgia*, pages 679–687, 1999.
- [Zwi01] U. Zwick. Computer assisted proof of optimal approximability results. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, California*, 2001.