A Parallel Approximation Algorithm for the Max Cut Problem on Cubic Graphs

T. Calamoneri[1]*, I. Finocchi[1], Y. Manoussakis[2]**, and R. Petreschi[1]

[1] Dept. of Computer Science - Univ. of Rome "La Sapienza" - Italy
{calamo,finocchi,petreschi}@dsi.uniroma1.it
[2] Dept. of Computer Science - Univ. Paris Sud - Orsay - France
yannis@lri.lri.fr

Abstract. We deal with the *maximum cut problem* on cubic graphs and we present a simple $O(\log n)$ time parallel algorithm, running on a CRCW PRAM with $O(n)$ processors. The approximation ratio of our algorithm is $1.\bar{3}$ and improves the best known parallel approximation ratio, i.e. 2, in the special case of cubic graphs.

Keywords: Cubic graphs, Max Cut, NP-completeness, PRAM Model.

1 Introduction and Notation

In this paper we deal with the *maximum cut problem*, that can be formally stated as follows:

INSTANCE: An undirected n-vertex graph $G(V, E)$.

SOLUTION: A partition of V into two disjoint sets V_r (right side class) and V_l (left side class).

MEASURE: The number of edges with one endpoint in V_l and one endpoint in V_r, i.e. the cardinality of $E_s = \{(u, v)$ such that either $u \in V_l$ and $v \in V_r$ or $u \in V_r$ and $v \in V_l\}$.

The problem of finding a maximum cut of a given graph is, in general, NP-complete [6, 7] and has been deeply studied (see, for example, [3, 4, 5, 11, 12, 14, 16, 17]).

In this paper we point out our attention on the special class of cubic graphs [1, 9]. Even restricted to this class, the maximum cut problem does not become easier, since it has been proved to be NP-complete if the graph is triangle-free and is at most cubic [18] and to be APX-complete, even if the degree of G is bounded by a constant [15]. Here we present new theoretical results characterizing the cardinality of the cut in cubic graphs with respect to the degree of vertices in

* This author was partially supported by the Italian National Research Council (CNR).
** Part of this research has been done while visiting Dept. of Computer Science, University of Rome "La Sapienza".

the graph (V, E_s). These results make it possible to design a simple $O(\log n)$ time parallel algorithm, running on a CRCW PRAM with $O(n)$ processors.

The best known approximation ratio for the maximum cut problem in parallel is 2 and follows from [13]. Our results improve it in the special case of cubic graphs, since our algorithm achieves an approximation ratio $1.\overline{3}$. So, this parallel algorithm approaches the best known sequential approximation ratio, that is 1.138 proved in [8]. It is worthwhile to note that the sequential algorithm for Max Cut presented in [8] is based on the use of the primal-dual technique and its parallelization seems not to be easy.

The remainder of this paper is organized as follows. Section 2 considers the problem from a theoretical point of view, while Section 3 addresses the design of the parallel approximation algorithm. Conclusions and open problems are presented in Section 4.

In the sequel, we denote with B the bipartite graph $B(V_l \cup V_r, E_s)$ and with E_d the set $E - E_s$, i.e. $E_d = \{(u, v)$ s.t. either $u, v \in V_l$ or $u, v \in V_r\}$. From now on we call edges in E_s and E_d *solid* and *dotted* edges, respectively.

We partition the vertices of V_l (V_r) according to their *solid degree*, that is their degree in B. In particular, $V_l = L_0 \cup L_1 \cup L_2 \cup L_3$ and $V_r = R_0 \cup R_1 \cup R_2 \cup R_3$, where L_i and R_i are the sets of vertices of solid degree $i = 0, 1, 2, 3$ in V_l and V_r, respectively. We also denote with l_i (r_i) the cardinality of L_i (R_i).

2 Some Theoretical Results

The aim of this section is to give sufficient conditions on V_l and V_r in order to guarantee the approximation ratio obtained in this work.

In the following we state some results referring to V_l, but – for symmetry properties – it will be always possible to exchange the roles of V_l and V_r.

First of all, observe that it is always possible to modify the partition so that no vertex has solid degree 0; indeed, if such a vertex exists, it is enough to move it from its class to the other one. Therefore, it is not restrictive to suppose $l_0 = r_0 = 0$.

Moreover, under the condition $l_1 = l_2 = 0$, we can trivially deduce the following equations which will be useful for the rest of the section:

1. $3l_3 = 3r_3 + 2r_2 + r_1$, derived by counting the number of solid edges as sum of solid degrees of vertices in V_l and V_r.
2. $n = l_3 + r_1 + r_2 + r_3$, derived by counting the total number of vertices.

Lemma 1. *If V_l contains only vertices of solid degree* 3*, i.e. $V_l = L_3$, then* $\frac{n}{4} \leq l_3 \leq \frac{n}{2}$.

Proof. The condition is easily deduced from Equations 1 and 2. In particular, in order to obtain the lower bound we isolate r_1 from Equation 2 and we substitute it in Equation 1. For what concerns the upper bound, we isolate r_3 from Equation 2 and we substitute it in Equation 1.

It is to notice that lower and upper bounds on l_3 hold when $r_2 = r_3 = 0$ and $r_1 = r_2 = 0$, respectively.

Lemma 2. *If V_l contains only vertices of solid degree 3, i.e. $V_l = L_3$, then $r_1 \geq 2n - 5l_3$.*

Proof. From Equation 1 and Equation 2 it follows that $r_2 + r_3 = \frac{3l_3 + r_2 - r_1}{3}$ and that $r_1 = n - l_3 - (r_2 + r_3)$, respectively.

By substituting the first equality in the second one, simple calculations lead to $r_1 = \frac{3}{2}n - 3l_3 - \frac{r_2}{2}$. In this latter equality we substitute $r_2 \leq n - l_3 - r_1$, obtained by Equation 2 and by the fact that $r_3 \geq 0$.

The inequality in the statement follows immediately.

Let c be the maximum number of pairwise vertex-disjoint odd length cycles in G and let c_r be the number of odd length cycles in the subgraph of G induced by V_r. Note that, due to the structure of the partition of vertices, all cycles in V_r are vertex-disjoint. Trivially, $c \geq c_r$. Moreover, let us remind that the *odd girth* g of a graph G is defined as the length a shortest odd cycle (if any) in G.

Lemma 3. *For any partition (V_l, V_r) of the vertices of G it holds: $0 \leq c_r \leq \frac{r_1}{g}$.*

Proof. It is easy to see that only vertices of R_1 can be involved in cycles in the subgraph induced by V_r. Moreover, this subgraph is a collection of isolated vertices, simple paths and cycles. The upper bound for c_r holds when all the connected components of the subgraph induced by V_r are odd length cycles, each having length g.

3 A Parallel Algorithm for Finding a “Large” Cut

In this section we present a parallel algorithm for finding a “large” cut of a cubic graph.

The idea behind our algorithm is to find any bipartition of the vertices and to increase the number of edges in the cut by appropriately moving vertices from one side of the bipartition to the other. In particular, we move to the opposite side both vertices of solid degree 0 and vertices of solid degree 1; indeed, as shown in Figure 1, each time we transfer a 0-degree vertex it becomes a 3-degree vertex and the number of solid edges increases by 3 and each time we transfer a 1-degree vertex it becomes a 2-degree vertex and the number of solid edges increases by 1.

Before detailing the algorithm for approximating the maximum cut, we present some preliminary procedures for coloring vertices that will be used in the following.

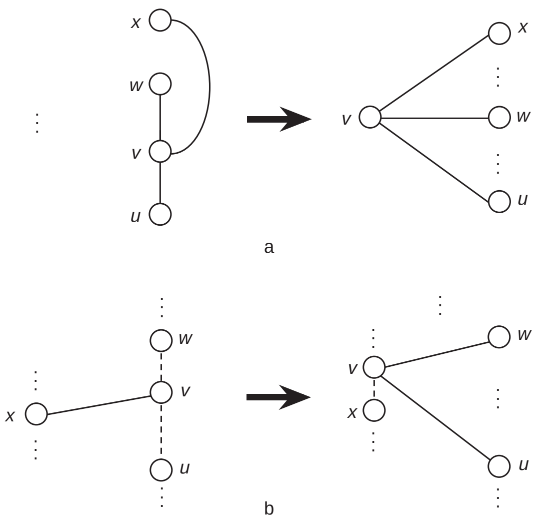

Fig. 1. (a) Vertex v is moved from R_0 to V_l; (b) Vertex v is moved from R_1 to V_l.

Lemma 4. *Let $G(V,E)$ be a not necessarily connected graph of maximum degree 2. Then $O(\log n)$ parallel steps, using $O(n)$ processors on a EREW PRAM model, are sufficient to find a 3-coloring of G, where the number of vertices having color 3 is as small as possible (possibly 0).*

Proof. Observe that $G(V,E)$ is a collection of isolated vertices, paths and cycles.

It is not difficult to recognize all the connected components and to decide whether they are vertices, paths or cycles by using the pointer jumping technique [10].

Then we work on each connected component as follows:

- If the connected component is an isolated vertex, we give it color 1.
- If the connected component is a path, we find a 2-coloring using the pointer jumping technique. Namely, we start from any vertex v, we assign to v color 1 and to its neighbors color 2; in the general step i of the pointer jumping loop, we assign the color of vertex j to not yet colored vertices at distance 2^i from j. First observe that after $\lceil \log n \rceil - 1$ iterations each vertex has a color, as guaranteed by the pointer jumping technique. The found coloring is trivially a 2-coloring and it is valid. Indeed, vertex v assigns its color (that is, 1) to all vertices with even distance from it, while v's adjacent vertices assign their color (that is, 2) to all vertices with even distance from them, and so with odd distance from v. It follows that adjacent vertices cannot have the same color.

- If the connected component is a cycle, we find a 3-coloring such that color 3 is assigned to at most one vertex. We choose at random an edge (u, w) of the cycle and we remove it; what remains is obviously a path and the previous procedure can be run. If in the output u and w have the same color, then we set $c(w) = 3$. The proof of correctness is very similar to the previous one.

For what concerns the parallel complexity, the pointer jumping technique guarantees a $O(\log n)$ time using $O(n)$ processors on a EREW PRAM model.

We want to underline that although there exist more efficient algorithms to find a 3-coloring of a cycle [2], the previous algorithm guarantees that at most one vertex receives color 3.

Before stating the next lemma, we need to introduce the concept of *independent dominating set* of a graph. An independent dominating set of a graph $G(V, E)$ is a subset $V' \subseteq V$ such that for any vertex $u \in V - V'$ there is a vertex $v \in V'$ for which $(u, v) \in E$, and such that no two vertices in V' are joined by an edge in E.

Lemma 5. *Let $G(V, E)$ be a graph of maximum degree 3. Then $O(\log n)$ parallel steps, using $O(n)$ processors on a CRCW PRAM model, are sufficient to find an independent dominating set S for G.*

Proof. Assume w.l.o.g. that G is connected (otherwise we apply the same argument for each connected component of G).

We build an independent dominating set S as follows. First, we find a rooted spanning tree T of G and assign to each vertex its level in T. Then, we consider the subgraph induced by the vertices at odd level; as it has maximum degree 2, we can 3-color it according to the algorithm in the proof of Lemma 4. We put in S all vertices colored 1 and we delete from G both them and their neighbors. The remaining vertices are a subset of vertices at even level in T and have maximum degree 2. We 3-color the graph induced by these vertices and we add to S all vertices colored 1.

We now prove the correctness of the algorithm.

S is an independent set. First, a 3-coloring of the subgraph induced by all the vertices at odd level is found and color 1 is an independent set S' for it. Then, the subgraph induced by all the vertices at even level that are not adjacent to any vertex in S' is considered. By 3-coloring its vertices and considering color 1, we construct an independent set S'' for this subgraph. $S = S' \cup S''$ is an independent set because, by construction, no vertex in S'' is adjacent to any vertex in S'.

S is a dominating set. The coloring algorithms ensure that each vertex colored 2 is adjacent to at least one vertex colored 1, and that each vertex colored 3 is adjacent to exactly one vertex colored 1 and one vertex colored 2. Moreover, each vertex at even level deleted after the first coloring is adjacent to at least a vertex colored 1. This guarantees that each vertex in $V - S$ is adjacent to at least one vertex colored 1.

For what concerns the complexity and the PRAM model, let us consider each step separately. Finding a spanning tree requires $O(\log n)$ time using $O(n)$ processors on a CRCW PRAM [10]. Rooting the tree and leveling its vertices can be done through the Euler Tour technique [10] in $O(\log n)$ time using $O(n)$ processors on a EREW PRAM. Finding connected components and coloring them requires $O(\log n)$ time with $O(n)$ processors on a EREW PRAM in view of Lemma 4. All the other tests and operations are performed in constant time.

Now we are ready to describe the parallel algorithm for finding a "large" cut.

As already stated, we start by any bipartition of the vertices. The first step consists in eliminating vertices of solid degree 0, if they exist. Furthermore, in order to satisfy the hypotheses of Lemma 1 and Lemma 2, we move some vertices from V_l to V_r in order to obtain $V_l = L_3$. Observe that $|E_s|$ may decrease during this step, but we need it for obtaining a stronger structure of the bipartite graph B.

At this point we eliminate 1-degree vertices from V_r; unfortunately, we are not able to ensure the extinction of 1-degree vertices from the whole graph, because they can be generated in V_l, although they completely disappear from V_r. We can however guarantee that the performance ratio of our algorithm is $1.\bar{3}$.

In the following we give the headlines of our algorithm and then we detail it step by step.

```
ALGORITHM Parallel-Approx-Max-Cut($G$);
Input: a cubic graph $G(V,E)$ with $V = \{0, 1, \dots, n-1\}$
Output: A bipartition ($V_l$,$V_r$) of the vertices of $G$ such that the Max Cut is
approximated by a ratio of $1.\bar{3}$
Step 1:
    $V_l \leftarrow \{v \in V$ such that $v$ is even$\}$
    $V_r \leftarrow \{v \in V$ such that $v$ is odd$\}$
    Eliminate-0-Degree($V_l$,$V_r$)
Step 2:
    Make-Left-Side-Of-Degree-3($V_l$,$V_r$)
    Eliminate-0-Degree($V_l$,$V_r$)
Step 3:
    Eliminate-1-Degree-From-Right-Side($V_l$,$V_r$)
    Eliminate-0-Degree($V_l$,$V_r$)
    RETURN($V_l$,$V_r$)
```

Recall that moving a node of solid degree 0 or 1 to the opposite side improves the number of solid edges since all its incident dotted edges become solid and vice versa. As our algorithm works in parallel, we must pay attention in avoiding that adjacent vertices are moved in the same parallel step, because this fact could make useless the local improvement. Hence, our algorithm makes strong use of coloring procedures to check this independence property. In the following, we detail its main steps.

- **Eliminate-0-Degree(V_l,V_r)**
  The aim of this procedure is to eliminate 0-degree vertices from G by moving some of them to the opposite side. Observe that transferring an independent dominating set of $L_0 \cup R_0$ makes $L_0 = R_0 = \emptyset$ and the solid degree of each moved vertex becomes 3. Thus Lemma 5 can be applied to find an independent dominating set whose vertices can be moved to the opposite side in a single parallel step. The procedure returns the updated sets V_l and V_r.
  This procedure can be run on a CRCW PRAM model in $O(\log n)$ time using $O(n)$ processors.
- **Make-Left-Side-Of-Degree-3(V_l,V_r)**
  In order to eliminate all vertices of solid degree 1 or 2 in the left side, let us consider the subgraph induced by $L_1 \cup L_2$ (by the previous algorithm step we can assume $L_0 = \emptyset$). If we consider a 3-coloring of such graph and move to V_r vertices of any two colors, we guarantee remaining vertices have degree 3. Since we are interested in making l_3 as large as possible (remind $|E_s| = 3l_3$), among all sets of vertices of any couple of colors we choose to move the less numerous one. A convenient 3 coloring can be found by means of the algorithm described in the proof of Lemma 4.
  This procedure can be run on a EREW PRAM model in $O(\log n)$ time using $O(n)$ processors.
- **Eliminate-1-Degree-From-Right-Side(V_l,V_r)**
  Let us consider the subgraph induced by R_1. After 3 coloring its vertices, we move those with color 1 to V_l in order to eliminate 1-degree vertices from the right side. Unlike the previous procedure, here we move only one color because we want to minimize the possible new vertices of degree 1 created in V_l (note that trying to move both a vertex colored 3 and its adjacent vertex colored 1 should imply that both of them are added to L_1). Unfortunately, not moving vertices colored 3 does not guarantee to have L_1 empty at the end of the procedure: indeed, a vertex in L_3 can become of degree 1 each time it is adjacent to two vertices in V_r colored 1.
  The running time of this procedure is the same as the previous one.

Lemma 6. *The execution of the procedure Eliminate-1-Degree-From-Right-Side (V_l, V_r) increases the cardinality of E_S by k, if k vertices are moved from R_1 to V_l.*

Proof. The assertion follows from the fact that moved vertices are independent since all of them have color 1 and from the observation of Figure 1(b).

Lemma 7. *During the execution of the procedure Eliminate-1-Degree-From--Right-Side(V_l, V_r) at least $\frac{r_1}{2} - \frac{c_r}{2}$ vertices are moved from V_r to V_l.*

Proof. Let us denote by p_o and p_e the number of vertices in R_1 involved in odd- and even-length paths in the subgraph induced by V_r, where the *length* of a path is defined to be the number of its edges. Analogously, c_o and c_e denote the number of vertices in R_1 involved in odd- and even-length cycles in the same subgraph.

As the procedure Eliminate-1-Degree-From-Right-Side(V_l, V_r) is concerned, we move from the right side to the left one the following number of vertices:

– for each path of even length l, $\frac{l}{2}$ vertices; therefore, totally at least $\frac{p_e}{2}$. Indeed, let $l_1, l_2, \ldots, l_k$ be the lengths of even-lengths paths in the subgraph induced by V_r. Then, $l_1 - 1, l_2 - 1, \ldots, l_k - 1$ are the numbers of vertices of degree 1 in such paths. Therefore, $\sum_{i=1}^{k} l_i = p_e + k$. The number of moved vertices is $\sum_{i=1}^{k} \frac{l_i}{2} = \frac{1}{2}p_e + \frac{1}{2}k \geq \frac{1}{2}p_e$;
– for each odd-length path of length l, $\frac{l-1}{2}$ vertices; therefore – with reasonings similar to the previous ones – totally exactly $\frac{p_o}{2}$;
– for each even-length cycle, exactly half of its vertices; therefore, totally exactly $\frac{c_e}{2}$;
– for each cycle of odd length l, $\frac{l-1}{2}$ vertices; therefore, totally exactly $\sum_{i=1}^{c_r} \frac{l_i - 1}{2}$, where l_i is the length of the i-th odd-length cycle. This sum is equal to $\frac{c_o}{2} - \frac{c_r}{2}$.

Hence, by summing up all these contributions, the number of vertices moved from right to left is at least $\frac{p_e}{2} + \frac{p_o}{2} + \frac{c_e}{2} + \frac{c_o}{2} - \frac{c_r}{2} = \frac{r_1}{2} - \frac{c_r}{2}$.

The algorithm outputs a partition of the vertices of the graph into two sets V_l and V_r and the solution is the set E_s, i.e. the set of edges connecting V_l with V_r. The next theorem guarantees the approximation ratio $1.\bar{3}$ for our algorithm.

Theorem 8. *The performance ratio of* `Parallel-Approx-Max-Cut(`G`)` *is* $1.\bar{3}$.

Proof. Observe that the size of the optimal maximum cut cannot exceed the difference between the number of edges and the maximum number of vertex disjoint odd-length cycles, i.e. $\frac{3}{2}n - c \leq \frac{3}{2}n - c_r$.

From the idea behind the algorithm and Lemma 7 it follows that the size of our approximate solution is at least $3l_3 + \frac{r_1}{2} - \frac{c_r}{2}$.

Consequently, the approximation ratio of our algorithm is

$$R \leq \frac{\frac{3}{2}n - c_r}{3l_3 + \frac{r_1}{2} - \frac{c_r}{2}}$$

that is a function of c_r, bounded by its maximum over the definition interval of c_r.

This function always decreases since its derivative

$$\frac{-3l_3 - \frac{r_1}{2} + \frac{3}{4}n}{(3l_3 + \frac{r_1}{2} - \frac{c_r}{2})^2}$$

is always negative in view of Lemma 1 and Lemma 2:

$$-3l_3 - \frac{r_1}{2} + \frac{3}{4}n \leq -3l_3 - \frac{1}{2}(2n - 5l_3) + \frac{3}{4}n \leq -\frac{n}{8}\frac{n}{4} < 0$$

Considering the definition interval for c_r given in Lemma 3, it is easy to prove that the maximum of our function holds for $c_r = 0$. Furthermore, from Lemma 2 and Lemma 1 we have the following chain of inequalities:

$$R \leq \frac{\frac{3}{2}n}{3l_3 + \frac{r_1}{2}} \leq \frac{\frac{3}{2}n}{3l_3 + \frac{1}{2}(2n - 5l_3)} \leq \frac{\frac{3}{2}n}{n + \frac{n}{8}} = \frac{4}{3} = 1.\bar{3}$$

4 Conclusions

We presented a parallel approximation algorithm for Max Cut on cubic graphs that achieves a performance ratio equal to $1.\bar{3}$, substantially improving the best known parallel approximation ratio, i.e. 2, in the special case of cubic graphs.

Starting from any bipartition of the vertices, the general strategy of our algorithm consists of increasing the number of edges in the cut by appropriately moving vertices from one side of the bipartition to the other.

The algorithm manages with simple coloring procedures and can be efficiently implemented in $O(\log n)$ parallel time on a CRCW PRAM with $O(n)$ processors.

We consider to be interesting to test the experimental behavior of this algorithm and to compare the quality of its solutions with the quality of the solutions found by the best known sequential algorithm on cubic graphs. Moreover, a generalization of the approach to d-regular graphs should also represent a valuable contribution.

References

[1] Calamoneri, T.: *Does Cubicity Help to Solve Problems?* Ph.D. Thesis, University of Rome "La Sapienza",XI-2-97, 1997.

[2] Cole, R. – Vishkin, U.: Deterministic Coin Tossing with applications to optimal parallel list ranking, *Information and Control*, 70(1), pp. 32-53, 1986.

[3] Delorme, C. – Poljak, S.: Combinatorial properties and the complexity of a max-cut approximation, *European Journal of Combinatorics*, 14, pp. 313-333, 1993.

[4] Fernandez de la Vega, W. – Kenyon, C.: A Randomized Approximation Scheme for Metric MAX-CUT, *IEEE Symposium on Foundations of Computer Science (FOCS98)*, 1998.

[5] Frieze, A.M. – Jerrum, M.: Improved Approximation Algorithms for MAX k-CUT and MAX BISECTION, *Algorithmica*, 18(1), pp. 67-81, 1997.

[6] Garey, M.R. – Johnson, D.S.: *Computers and Intractability: a Guide to Theory of NP-completeness*, W.H.Freeman, 1979.

[7] Garey, M.R. – Johnson, D.S. – Stockmayer, L.: Some Simplified NP Complete Graph Problems, *Theor. Comput. Sci.*, 1, pp. 237-267, 1976.

[8] Goemans, M.X. – Williamson, D.P.: .878-Approximation Algorithms for MAX CUT and MAX 2SAT, *Proceedings of the Twenty-Sixth Annual ACM Symposium on the Theory of Computing (STOC94)*, pp. 422-431, 1994.

[9] Greenlaw, R. – Petreschi, R.: Cubic graphs, *ACM Computing Surveys*, 27(4), pp. 471-495, 1995.

[10] Jájá, J.: *An Introduction to Parallel Algorithms*. Addison Wesley, 1992.

[11] Kann, V. – Khanna, S. – Lagergren, J. – Panconesi, A.: On the Hardness of Approximating Max k-Cut and its Dual, *Chicago Journal of Theoretical Computer Science*, 1997.

[12] Karloff, H.: How Good is the Goemans-Williamson MAX CUT Algorithm?, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing (STOC96)*, pp. 427-434, 1996.

[13] Luby, M.: A Simple Parallel Algorithm for the Maximal Independent Set, *SIAM J. on Computing*, 15, pp. 1036-1053, 1986.

[14] Nesetril, J. – Turzik, D.: Solving and Approximating Combinatorial Optimization Problems (Towards MAX CUT and TSP), *Proc. of SOFSEM97: Theory and Practics of Informatics*, LNCS 1338, pp. 70-81, 1997.

[15] Papadimitriou, C.H. and Yannakakis, M.: Optimization, Approximation, and Complexity Classes, *J. Comput. System Sci.*, 43, pp. 425-440, 1991.

[16] Poljak, S.: Integer Linear Programs and Local Search for Max-Cut, *SIAM Journal on Computing*, 24(4), pp. 822-839, 1995.

[17] Poljak, S. – Tuza, Z.: The max-cut problem- a survey Special Year on Combinatorial Optimization, *DIMACS series in Discrete Mathematics and Theoretical Computer Science*, 1995.

[18] Yannakakis, M.: Node- and Edge-Deletion NP-Complete Problems. *Proc. 10th Annual ACM Symp. on the Theory of Comp. (STOC78)*, ACM New York, 1978, pp 253–264.