On Max Cut in Cubic Graphs

Tiziana Calamoneri Irene Finocchi Rossella Petreschi
Department of Computer Science - University of Rome "La Sapienza" - Italy
{ *calamo, finocchi, petreschi* }*@dsi.uniroma1.it*

Yannis Manoussakis*
Department of Computer Science - University of Paris Sud, Orsay - France
yannis@lri.lri.fr

Abstract

This paper is concerned with the *maximum cut problem* in parallel on cubic graphs. New theoretical results characterizing the cardinality of the cut are presented. These results make it possible to design a simple combinatorial $O(\log n)$ time parallel algorithm, running on a CRCW P-RAM with $O(n)$ processors. The approximation ratio achieved by the algorithm is $1.\bar{3}$ and improves the best known parallel approximation ratio, i.e., 2, in the special class of cubic graphs. The algorithm also guarantees that the size of the returned cut is at least $\frac{9g-3}{8g}n$, where g is the odd girth of the input graph. Experimental results round off the paper, showing that the solutions obtained in practice are likely to be much better than the theoretical lower bound.

Keywords: Parallel Algorithms, Approximation Algorithms, Max Cut, Bipartite Graphs, Cubic Graphs, P-RAM Model.

1 Introduction

This paper is concerned with the *maximum cut problem*, a well-known combinatorial optimization problem that is formally stated as follows:

INSTANCE: an undirected n-vertex graph $G(V, E)$.
SOLUTION: a partition of the vertex set V into two disjoint sets V_r and V_l. These sets identify a bipartite subgraph $B = (V_l, V_r, E_s)$, where $E_s = \{(u, v)$ s.t. $u \in V_l$ and $v \in V_r\}$.
MEASURE: the cardinality of E_s, i.e., the size of the cut.

The problem of finding a maximum cut of a given graph is, in general, NP-complete [7, 8]. Due to its relevance in several applications, it has been deeply studied, with respect to both the definition above [3, 5, 6, 9, 15, 16, 19, 22, 23] and the formulation as the maximum

*Part of this research has been done while visiting the Department of Computer Science, University of Rome "La Sapienza", Italy.

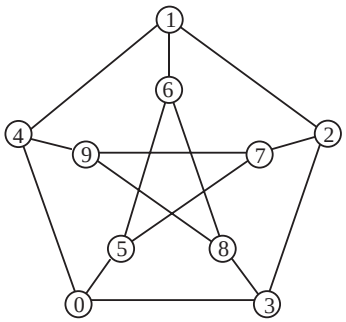

Figure 1: Petersen graph.

bipartite subgraph problem [4, 12, 17]. Max cut does not become easier even on the special class of cubic graphs, i.e., graphs whose vertices have constant degree 3 [1, 10]. Indeed, it has been proved to be NP-complete if the graph is triangle-free and at most cubic [24] and to be APX-complete even if the degree of G is bounded by a constant [21]. Moreover, tight lower bounds for an optimal solution have been proved. Namely, a lower bound on $|E_s|$ in cubic tetrahedron-free graphs is $\frac{7}{6}n$ [17] and it is the best possible. For cubic triangle-free graphs this bound has been improved to $\frac{6}{5}n$ [12], that is also tight due to the Petersen and the Dodecahedron graphs.

In this paper we present new theoretical results characterizing the size of the maximum cut in cubic graphs with respect to the degree of the vertices in the corresponding bipartite subgraph B and to the odd girth of G, i.e., the length of a shortest odd cycle of G (if any). These results make it possible to design a simple combinatorial parallel approximation algorithm, running on a CRCW P-RAM with $O(n)$ processors in $O(\log n)$ time.

We remark that the best known approximation ratio for the maximum cut problem in parallel is 2 and follows from [18]. Our results improve it in the special case of cubic graphs: our parallel algorithm achieves an approximation ratio $1.\bar{3}$, thus approaching the best known sequential approximation ratio 1.138 [9]. As far as the cut is built, we are also able to guarantee that $|E_s| \geq \frac{9g-3}{8g}n$, where g is the odd girth of G. Note that if G contains no odd cycle, then it is bipartite and $E_s = E$; in this case we consider $g \to \infty$, because it is well-known that graphs without short odd cycles are nearly bipartite [11]. In view of these considerations, the odd girth seems a natural measure for the bipartiteness of a graph.

Our lower bound on the size of the cut becomes $|E_s| \geq \frac{21}{20}n$ on triangle-free cubic graphs. Although this value is worse than the known lower bound $\frac{6}{5}n$, on some graphs for which the $\frac{6}{5}n$ bound is tight the algorithm is able to gain it (e.g., Petersen graph in Figure 1). For this reason we perform an experimental analysis aimed at determining the percentage of graphs for which the $\frac{6}{5}n$ bound is not achieved by the algorithm. We consider more than 11000 triangle-free cubic graphs, showing that only in the 0.02% of the instances our algorithm does not obtain the $\frac{6}{5}n$ bound and in these cases the size of the solution is never more than a constant far from $\frac{6}{5}n$.

The remainder of this paper is organized as follows. Section 2 presents preliminary definitions and theoretical results useful for the analysis of the algorithm. In Section 3

the algorithm is described and its performances are discussed. Section 4 is devoted to the experimental results, and Section 5 addresses concluding remarks and highlights directions for further research.

2 Preliminaries

Let $G = (V, E)$ be a n-vertex cubic graph and let V_l and V_r denote any partition of V. Let $E_s = \{(u, v) \text{ s.t. } u \in V_l \text{ and } v \in V_r\}$ and $E_d = E - E_s$, i.e., $E_d = \{(u, v) \text{ s.t. } u, v \in V_l \text{ or } u, v \in V_r\}$. In the rest of this paper we call edges in E_s and E_d *solid* and *dotted* edges, respectively. Let $B = (V_l, V_r, E_s)$ be the bipartite subgraph of G identified by V_l and V_r.

We can 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 denote by l_i (r_i) the cardinality of L_i (R_i). It is worth observing 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 sufficient to move it from its side to the other one. Therefore, it is not restrictive to assume

$$l_0 = r_0 = 0$$

We will detail later how to achieve this configuration in parallel from an algorithmic point of view. Under this hypothesis, subgraph G_l (G_r) induced by V_l (V_r) has maximum degree 2 and consists of isolated vertices, paths, and simple cycles. We will call c_l (c_r) the number of odd length cycles in G_l (G_r).

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

By counting the number of solid edges and the total number of vertices, respectively, we trivially deduce the following equations which will be useful in the remainder of this section:

$$3l_3 + 2l_2 + l_1 = 3r_3 + 2r_2 + r_1 \tag{1}$$

$$n = l_3 + l_2 + l_1 + r_1 + r_2 + r_3 \tag{2}$$

Lemmas 1 to 4 hold when V_l contains only vertices of solid degree 3, i.e., $V_l = L_3$, $L_2 = \emptyset$, and $L_1 = \emptyset$.

Lemma 1 *If $V_l = L_3$, then $\frac{n}{4} \leq l_3 \leq \frac{n}{2}$.*

Proof. The condition is easily deduced from Equation 1 and Equation 2. In particular, in order to obtain the lower bound, we isolate r_1 from Equation 2 and we substitute it in Equation 1. To get 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 $V_l = L_3$, 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 $r_1 = \frac{3}{2}n - 3l_3 - \frac{r_2}{2}$. The inequality in the statement is obtained observing that $r_2 \leq n - l_3 - r_1$, due to Equation 2 and to the fact that $r_3 \geq 0$. □

Lemma 3 *Assume $V_l = L_3$. For each l_3 and for each couple of feasible triples (r_1, r_2, r_3) and (r'_1, r'_2, r'_3), it holds: $r_1 - r'_1 = r_3 - r'_3$ and $r_2 - r'_2 = -2(r_1 - r'_1)$.*

Proof. Fixing l_3 and given two feasible triples (r_1, r_2, r_3) and (r'_1, r'_2, r'_3), it is possible to write the values r'_i with respect to the values r_i, $i = 1, 2, 3$, by counting the variation of the number of vertices with solid degree i; namely, $r'_i = r_i + \Delta_i$, $i = 1, 2, 3$ and $\Delta_i \in \mathbf{Z}$.

Due to its feasibility, each triple satisfies Equation 1 and Equation 2. In view of the fact that l_3 is fixed, the right side of Equation 1 can be made equal to $3r'_3 + 2r'_2 + r'_1$. The same holds for Equation 2. Hence we obtain:

$$\begin{cases} 3r_3 + 2r_2 + r_1 = 3(r_3 + \Delta_3) + 2(r_2 + \Delta_2) + (r_1 + \Delta_1) \\ l_3 + r_3 + r_2 + r_1 = l_3 + (r_3 + \Delta_3) + (r_2 + \Delta_2) + (r_1 + \Delta_1) \end{cases}$$

from which $\Delta_1 = \Delta_3$ and $\Delta_2 = -2\Delta_1$ immediately follow. □

Lemma 4 *Let $V_l = L_3$. If $l_3 \geq \frac{2}{5}n$ or $r_3 \geq r_1$, then $|E_s| \geq \frac{6}{5}n$.*

Proof. $|E_s| = 3l_3$ due to the hypothesis $V_l = L_3$. If $l_3 \geq \frac{2}{5}n$ the statement trivially follows. Otherwise we isolate r_2 in Equation 2 and then we get from the Equation 1 $|E_s| = 3l_3 = \frac{6}{5}n + \frac{3}{5}r_3 - \frac{3}{5}r_1 \geq \frac{6}{5}n$ if and only if $r_3 \geq r_1$. □

Since V_l and V_r partition V, at least one among them has cardinality at least $\frac{n}{2}$. For symmetry reasons it is not restrictive to assume $|V_l| \geq \frac{n}{2}$.

Lemma 5 *If both V_l and V_r contain only vertices of solid degree 2 and 3, then a sufficient condition to gain the bound $\frac{6}{5}n$ is that $l_2 \leq \frac{3}{10}n$.*

Proof. $|E_s| = 3l_3 + 2l_2 = 3(l_3 + l_2) - l_2$. The $\frac{6}{5}n$ lower bound follows from the inequality $l_2 + l_3 \geq \frac{n}{2}$ and from the hypothesis $l_2 \leq \frac{3}{10}n$. □

Let c be the maximum number of vertex disjoint odd cycles in G and let c_r be the number of odd cycles in G_r. Note that all cycles in V_r are vertex disjoint due to the structure of the partition, that does not contain vertices having solid degree 0. Trivially, $c \geq c_r$.

Lemma 6 *For any partition V_l, V_r of the vertices of G it holds: $0 \leq c_r \leq \frac{r_1}{g}$.*

Proof. First observe 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 An approximation algorithm for max cut

Before detailing the algorithm for approximating a maximum cut, we present some procedures for coloring vertices and for finding a maximal independent dominating set in parallel. We recall that a vertex coloring of a graph $G(V, E)$ is an assignment of colors to the vertices of G such that adjacent vertices have different colors. An independent dominating set of G is a subset of vertices $S \subseteq V$ such that if $u, v \in S$ then $(u, v) \notin E$ (independence) and $\forall w \in V$ either w or a neighbor of w belongs to S (dominance).

3.1 Two useful subroutines

Lemma 7 *Let $G(V, E)$ be a graph of maximum degree 2. Then $O(\log n)$ parallel steps, using $O(n)$ processors on a EREW P-RAM 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, simple 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 [13]. Then work on each connected component as follows:

- If the connected component is an isolated vertex, give it color 1.
- If the connected component is a path, find a 2-coloring using the pointer jumping technique. Namely, in the first step start from any vertex v, assign to v color 1 and to its adjacent vertices color 2; in the general step i of the pointer jumping loop, assign the color of each colored 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. Exactly 2 colors are used and the coloring is valid. Indeed, vertex v assigns its color (that is, 1) to all vertices having even distance from it, while v's neighbors assign their color (that is, 2) to all vertices having even distance from them, and so odd distance from v. It follows that adjacent vertices cannot have the same color.
- If the connected component is a simple cycle, find a 3-coloring such that color 3 is assigned to at most one vertex as follows: choose any edge (u, w) of the cycle and remove it; what remains is obviously a path and the previous procedure can be run. If u and w have the same color in the returned coloring, then set the color of w to 3. The proof of correctness is very similar to the previous one.

The pointer jumping technique guarantees that the algorithm can be run on a EREW P-RAM model using $O(n)$ processors in $O(\log n)$ time. □

We want to remark that although more efficient algorithms to find a 3-coloring of a cycle are known [2], the algorithm here presented guarantees that at most one vertex receives color 3.

Lemma 8 *Let $G(V, E)$ be a graph of maximum degree 3. Then $O(\log n)$ parallel steps, using $O(n)$ processors on a CRCW P-RAM model, are sufficient to find a maximal independent dominating set S for G.*

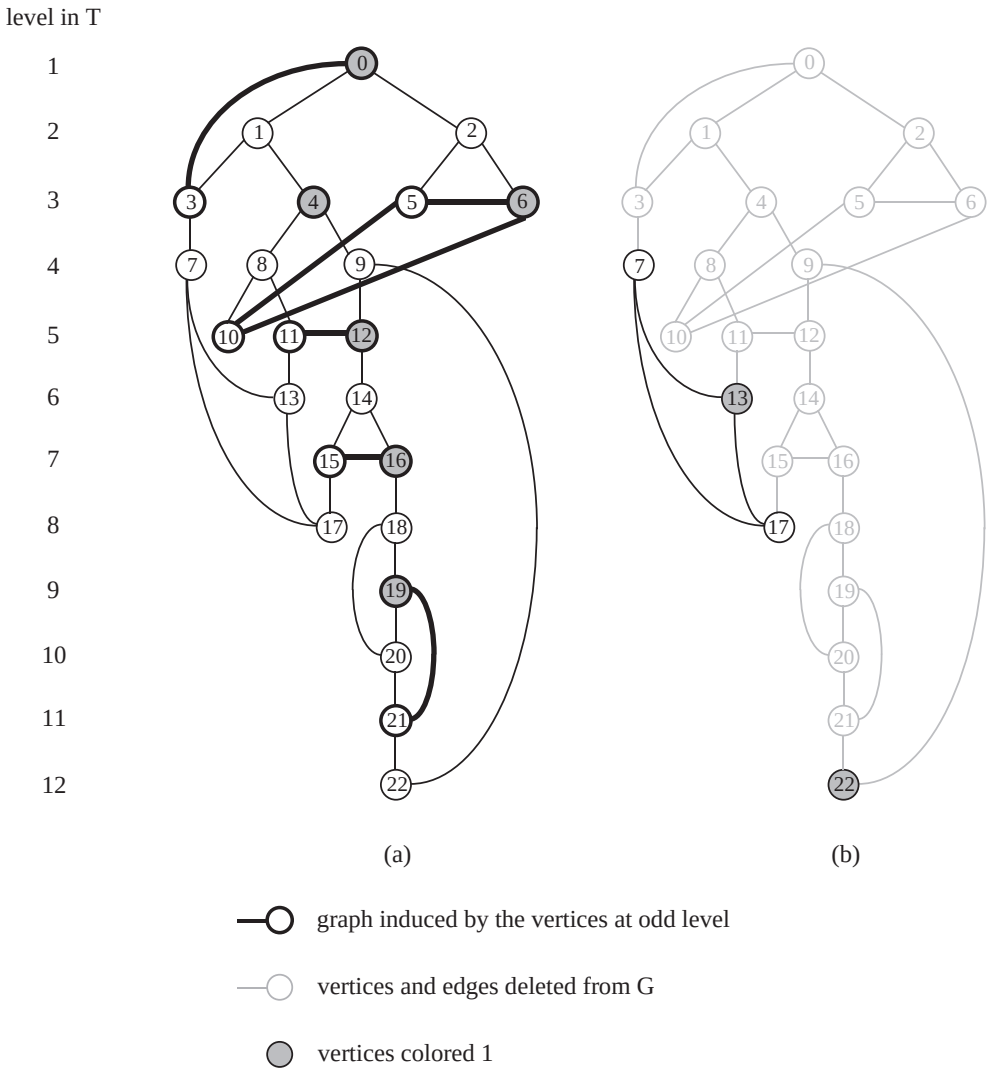

Figure 2: Finding a maximal independent dominating set as in the proof of Lemma 8: (a) vertices at odd levels are considered; (b) remaining vertices at even levels are considered.

Proof. Without loss of generality, assume that G is connected; if this is not the case, apply the same argument on each connected component. Build the independent dominating set S as follows. First, find a rooted spanning tree T of G and assign each vertex with its level in T. Then, consider the subgraph induced by the vertices at odd level; as it has maximum degree 2, 3-color it according to the algorithm in the proof of Lemma 7. Put in S all the vertices colored 1 and delete from G all these vertices, all their neighbors, and all the

edges incident to at least one of the removed vertices. The remaining vertices are a subset of the vertices at even levels in T and have maximum degree 2. 3-color them and add to S all the vertices colored 1. Figure 2 illustrates an example of the execution of this procedure.

Now we 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 vertices colored 1 are 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 build an independent set S'' for this subgraph. $S = S' \cup S''$ is still an independent set because, by construction, no vertex in S'' is adjacent to any vertex in S'.

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

To compute the complexity and determine the P-RAM model, let us consider each step separately. Finding a spanning tree requires $O(\log n)$ time using $O(n)$ processors on a CRCW P-RAM [13]. Rooting the tree and leveling its vertices can be done using the Euler Tour technique [13] in $O(\log n)$ time with $O(n)$ processors on a EREW P-RAM. Finding connected components and coloring them requires $O(\log n)$ time with $O(n)$ processors on a EREW P-RAM in view of Lemma 7. All the other tests and operations can be executed in constant parallel time. □

As a consequence of Lemma 8, removing from G vertices in S and edges incident to any of them we obtain a graph $\tilde{G}$ whose vertices have their degree decreased of at least 1.

3.2 Finding a large cut

We are now ready to describe the parallel algorithm for finding a large cut. The idea behind our algorithm consists of starting from any bipartition of the vertices and of increasing the number of edges in the cut by appropriately moving vertices from one side to the other one. In particular, we always move to the opposite side vertices of solid degree 0; indeed, each time we transfer a solid 0-degree vertex it becomes a solid 3-degree vertex and the number of solid edges increases by 3. 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; nevertheless, this is useful in order to provide the bipartite graph B with a stronger structure. At this point we eliminate 1-degree vertices from V_r, yet we are not able to ensure that all 1-degree vertices are removed from the whole graph: 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 $1.\bar{3}$ and that $|E_s| \geq \frac{9g-3}{8g}n$.

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 $V = \{0,1,\ldots,n-1\}$

Output: a bipartition (V_l,V_r) of the vertices of G such that the Max Cut is approximated by a ratio $1.\bar{3}$ and $|E_s| \geq \frac{9g-3}{8g}n$

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)

Let us first observe that moving a vertex 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 (see Figure 3). However, as our algorithm works in parallel, we must pay attention to avoid that adjacent vertices are moved in the same parallel step, because this fact could make useless the local improvement. Hence, the algorithm makes strong use of coloring procedures to check this independence property. In the following we detail the three main subroutines.

Eliminate-0-Degree(V_l,V_r): this procedure aims at eliminating 0-degree vertices from B by moving some of them to the opposite side. Observe that transferring an independent dominating set of the subgraph induced by $L_0 \cup R_0$ makes $L_0 = R_0 = \emptyset$ and the solid degree of each moved vertex becomes 3. Thus, Lemma 8 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.

`Eliminate-0-Degree` can be run on a CRCW P-RAM model in $O(\log n)$ time using $O(n)$ processors.

Make-Left-Side-Of-Degree-3(V_l,V_r): in order to eliminate from the left side all the vertices of solid degree 1 or 2, let us consider the subgraph induced by $L_1 \cup L_2$ (due to Step 1 $L_0 = \emptyset$). If we consider a 3-coloring of such graph and move to V_r vertices of any two colors, we guarantee the remaining vertices to 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 7.

`Make-Left-Side-Of-Degree-3` can be run on a EREW P-RAM model in $O(\log n)$ time using $O(n)$ processors.

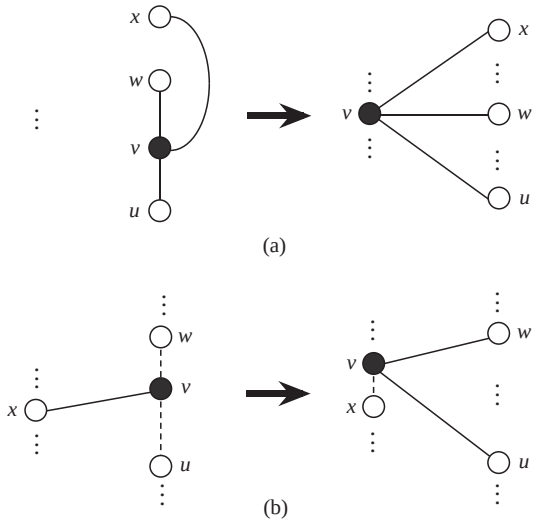

Figure 3: (a) Vertex v is moved from R_0 to V_l; (b) vertex v is moved from R_1 to V_l.

Eliminate-1-Degree-From-Right-Side(V_l,V_r): let us consider the subgraph induced by R_1. In order to eliminate 1-degree vertices from the right side, we 3-color its vertices and then we move those with color 1 to V_l. Unlike the previous procedure, here we move only one color because we want to minimize the possible new vertices of degree 1 in V_l (both a vertex colored 3 and its adjacent vertex colored 1 would be added to L_1 when moved together). This expedient does not guarantee L_1 to be empty at the end of this procedure: a vertex in L_3 becomes of degree 1 each time it is adjacent to two vertices colored 1 in V_r.

`Eliminate-1-Degree-From-Right-Side` can be run on a EREW P-RAM model in $O(\log n)$ time using $O(n)$ processors.

3.3 Analysis of the algorithm

In this section we analyze the performance of algorithm `Parallel-Approx-Max-Cut`.

Lemma 9 *If during the execution of procedure* `Eliminate-1-Degree-From-Right-Side`*(V_l, V_r) k vertices are moved from R_1 to V_l, then the cardinality of E_s increases by k.*

Proof. Since we move only vertices with color 1, the moved vertices are guaranteed to be independent. Then, we can consider vertices separately. The movement of a single vertex transforms a solid edge into a dotted one and two dotted edges into two solid ones (see Figure 3(b)). □

Lemma 10 *During the execution of procedure* `Eliminate-1-Degree-From-Right-Side`*(V_l, V_r) at least $\lceil \frac{r_1}{2} \rceil - \lceil \frac{c_r}{2} \rceil$ vertices are moved from V_r to V_l.*

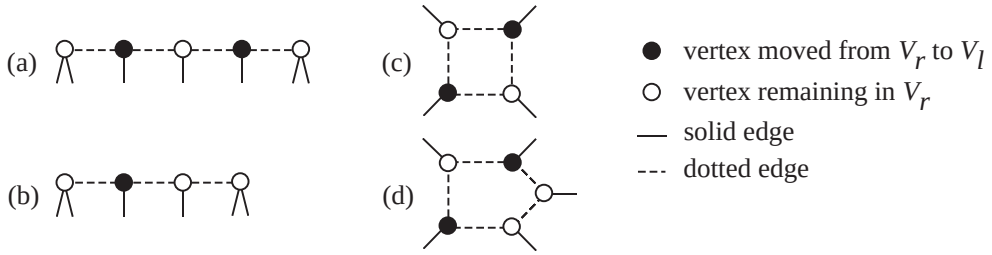

Figure 4: Eliminating 1-degree vertices.

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 G_r, where the *length* of a path is defined as the number of its edges. Similarly, 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 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 numbers of vertices:

- For each path of even length l, $\frac{l}{2}$ vertices (black vertices in Figure 4(a)); therefore, all the even-length paths contribute with at least $\lceil \frac{p_e}{2} \rceil$ vertices. Indeed, let $l_1, l_2, \ldots, l_k$ be the lengths of even-lengths paths in G_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 $\sum_{i=1}^{k} \frac{l_i}{2} = \lceil \frac{1}{2} p_e \rceil + \frac{1}{2} k \geq \lceil \frac{1}{2} p_e \rceil$.

- For each odd-length path of length l, $\frac{l-1}{2}$ vertices (black vertices in Figure 4(b)); therefore – with reasonings similar to the previous ones – the odd-length paths contribute in total with exactly $\frac{p_o}{2}$ vertices.

- For each even-length cycle, exactly half of its vertices (black vertices in Figure 4(c)); therefore, the even-length cycles contribute in total with exactly $\frac{c_e}{2}$ vertices.

- For each cycle of odd length l, $\frac{l-1}{2}$ vertices (black vertices in Figure 4(d)); therefore, the totality of odd-length cycles contributes with exactly $\sum_{i=1}^{c_r} \frac{l_i - 1}{2}$ vertices, where l_i is the length of the i-th odd-length cycle. This sum is equal to $\lceil \frac{c_o}{2} \rceil - \lceil \frac{c_r}{2} \rceil$.

Summing up all the contributions, the number of vertices moved from the right side to the left one is at least $\lceil \frac{p_e}{2} \rceil + \frac{p_o}{2} + \frac{c_e}{2} + \lceil \frac{c_o}{2} \rceil - \lceil \frac{c_r}{2} \rceil \geq \lceil \frac{r_1}{2} \rceil - \lceil \frac{c_r}{2} \rceil$. □

The algorithm returns 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 two theorems guarantee for our algorithm the approximation ratio $1.\bar{3}$ and the lower bound on the size of the solution $\frac{9g-3}{8g} n$.

Theorem 1 *Given a cubic graph G, algorithm* `Parallel-Approx-Max-Cut`(G) *achieves a performance ratio* $1.\bar{3}$.

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., $\frac{3}{2}n - c \leq \frac{3}{2}n - c_r$.

As the algorithm is concerned and in view of Lemma 10, the size of the solution is at least $3l_3 + \lceil \frac{r_1}{2} \rceil - \lceil \frac{c_r}{2} \rceil$. Consequently, the approximation ratio of our algorithm is

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

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

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

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

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

Considering the definition interval for c_r given in Lemma 6, it is easy to prove that the maximum of the function holds for $c_r = 0$. Furthermore, from Lemma 2 and Lemma 1 we have the following chain of inequalities:

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

□

Theorem 2 *Given a cubic graph G, algorithm* `Parallel-Approx-Max-Cut`(G) *returns a cut having size $|E_s| \geq \frac{9g-3}{8g}n$.*

Proof. After the execution of Step 2, the number of solid edges in B is equal to $3l_3$. The elimination of 1-degree vertices from the right side in Step 3 increases this number by a value that is at least $\lceil \frac{r_1}{2} \rceil - \lceil \frac{c_r}{2} \rceil$, according to Lemma 10. Then:

$$\begin{array}{ll}
|E_s| \geq 3l_3 + \lceil \frac{r_1}{2} \rceil - \lceil \frac{c_r}{2} \rceil \geq & \text{for } c_r \leq \frac{r_1}{g} \\
3l_3 + \frac{g-1}{2g}r_1 \geq & \text{for } r_1 \geq 2n - 5l_3 \\
3l_3 + \frac{g-1}{2g}(2n - 5l_3) = \frac{g-1}{g}n + \frac{g+5}{2g}l_3 \geq & \text{for } l_3 \geq \frac{n}{4} \\
\frac{g-1}{g}n + \frac{g+5}{2g}\frac{n}{4} = \frac{9g-3}{8g}n &
\end{array}$$

□

In conclusion, the following theorem holds:

Theorem 3 *Given a cubic graph G with odd girth g, algorithm* `Parallel-Approx-Max-Cut` (G) *is $1.\bar{3}$ approximating and finds a cut of G having size at least $\frac{9g-3}{8g}n$ in $O(\log n)$ time on a CRCW P-RAM model with $O(n)$ processors.*

An example of the execution of the algorithm on Petersen graph concludes this section. Figure 5(a) shows the bipartition of the vertices of the graph after Step 1. Figure 5(b) is obtained after the execution of procedure `Make-Left-Side-Of-Degree-3`, which moves from V_l to V_r vertices 0 and 6. At this point vertex 5 has solid degree 0 and it is moved to the left side by procedure `Eliminate-0-Degree` (see Figure 5(c)). On this instance Step 3 has no effect, because after Step 2 all vertices already have solid degree 2 or 3.

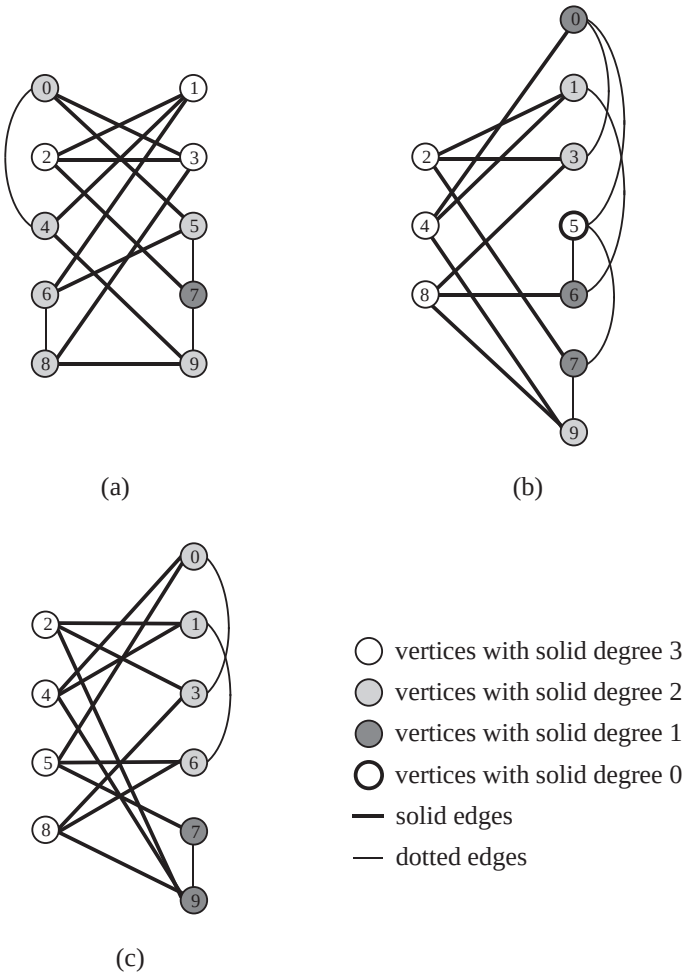

Figure 5: (a) The initial bipartition; (b) the bipartition after `Make-Left-Side-Of-Degree-3`; (c) the final bipartition.

4 Experimental results

The lower bound on the size of the solution proved in Theorem 2 holds for any cubic graph and is a function of the odd girth of the graph. Since this function is strictly increasing, the bigger is the odd girth, the better is the solution generated by the algorithm. In view of

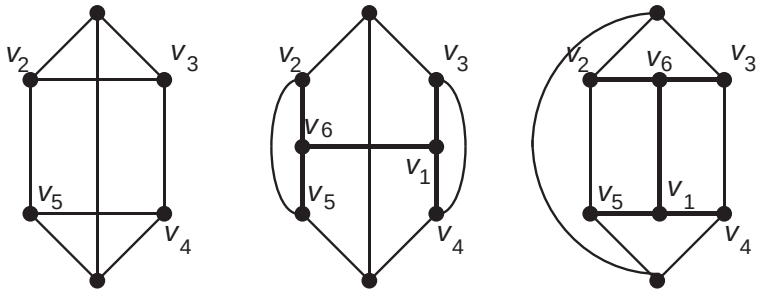

Figure 6: H-expansion operation for generating cubic graphs.

the fact that $\lim_{g\to\infty} \frac{9g-3}{8g} n = \frac{9}{8}n$, both the theoretical bound $\frac{7}{6}n$ for tetrahedron-free cubic graphs and the theoretical bound $\frac{6}{5}n$ for triangle-free cubic graphs might not be gained even on graphs with big odd girth. This means that our algorithm may not recognize a bipartite graph. However, we observed that on some relevant instances whose biggest cut has exactly size $\frac{6}{5}n$ (e.g., Petersen graph) the algorithm is able to achieve it. For this reason we have conducted an experimental analysis aimed at determining the percentage of graphs for which the algorithm does not obtain the $\frac{6}{5}n$ bound. The results of the experimentation support the consideration that the algorithm obtains in practice results much better than the $\frac{9g-3}{8g}n$ lower bound.

We have tested the algorithm on 11380 triangle-free cubic graphs with up to 100 vertices. All the instances have been generated using the H-expansion method introduced by E. L. Johnson in 1963 [14]. An H-expansion (see Figure 6) is an operation that, given a cubic graph G on n vertices and two arbitrary not-incident edges $e_1 = (v_2, v_3)$ and $e_2 = (v_4, v_5)$ in G, eliminates e_1 and e_2 and adds two vertices v_1 and v_6 with edges either

(a) (v_6, v_1), (v_6, v_2), (v_6, v_5), (v_1, v_3), (v_1, v_4) or

(b) (v_6, v_1), (v_6, v_2), (v_6, v_3), (v_1, v_5), (v_1, v_4).

Then, the following theorem holds:

Theorem 4 *[20] For $n \geq 6$, every connected cubic graph having $n + 2$ vertices is an H-expansion of a connected cubic graph having n vertices.*

As our experimentation concerns triangle-free cubic graphs, we need to use the H-expansion operation paying attention not to introduce triangles, when the starting graph is triangle-free. The (b) choice of edges in the H-expansion method guarantees this property.

We summarize the results of our experimentation, grouped according to the number of vertices, in Figure 7 and in Table 1.

The histogram in Figure 7 highlights that the algorithm does not reach the $\frac{6}{5}n$ lower bound on very few graphs. For each fixed number of vertices, grey and white bars represent the percentage of graphs with bipartition size $\geq \frac{6}{5}n$ and $< \frac{6}{5}n$, respectively. The histogram also shows that the percentage of graphs for which the algorithm does not gain the $\frac{6}{5}n$ bound decreases as the instance size gets bigger.

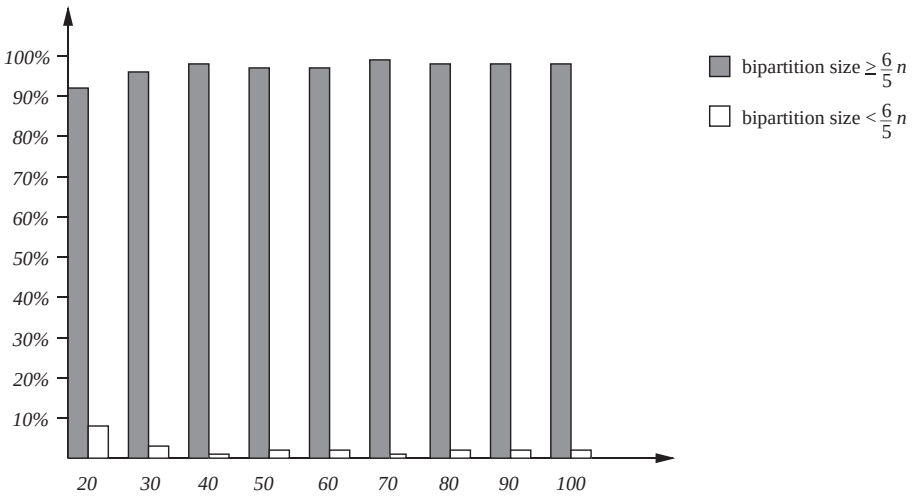

Figure 7: Percentage of graphs with bipartition size $\geq \frac{6}{5}n$ and $< \frac{6}{5}n$ (y axis) versus number of vertices (x axis).

The experimental results are detailed in Table 1, that is divided into nine columns reporting:

1. **n**: number of vertices in the graphs.
2. **Graphs**: number of graphs with fixed number of vertices **n**.
3. **More-$\frac{6}{5}$**: number of graphs among **Graphs** for which $|E_s| > \frac{6}{5}n$.
4. **$\frac{6}{5}$**: number of graphs among **Graphs** for which $|E_s|$ equals the theoretical bound $\frac{6}{5}n$.
5. **Less-$\frac{6}{5}$**: number of graphs among **Graphs** for which $|E_s| < \frac{6}{5}n$.
6. **$\frac{21}{20}$**: number of graphs among **Graphs** for which $|E_s|$ equals our theoretical bound $\frac{21}{20}n$.
7. **Distance**: maximum value of $\frac{6}{5}n - |E_s|$ computed on all graphs counted in columns **Less-$\frac{6}{5}$** and **$\frac{21}{20}$**.
8. **Average**: average value of $|E_s|$ for all the instances having number of vertices **n**.
9. **$\frac{6}{5}$n**: $\frac{6}{5}$ of the number of vertices **n**, used to compare with the previous column.

Observe that values in column **$\frac{21}{20}$** are always equal to 0: this means that the algorithm finds that for all the tested graphs the number of solid edges is greater than the theoretical bound proved in Theorem 2. Moreover, only for few graphs the algorithm do not gain the $\frac{6}{5}n$ bound (see column **Less-$\frac{6}{5}$**) and, even in this case, $|E_s|$ generated by the algorithm differs at most for 3 edges from this value, as shown in the **Distance** column. In total, only 248 cuts contain less than $\frac{6}{5}n$ edges, while 10336 bipartitions have size $> \frac{6}{5}n$.

n	Graphs	More-$\frac{6}{5}$	$\frac{6}{5}$	Less-$\frac{6}{5}$	$\frac{21}{20}$	Distance	Average	$\frac{6}{5}n$
20	300	151	126	23	0	1	24.660	24
30	340	266	63	11	0	1	37.447	36
40	940	849	72	19	0	1	50.497	48
50	1000	874	97	29	0	2	62.509	60
60	1000	888	87	25	0	2	74.864	72
70	1600	1478	105	17	0	2	87.387	84
80	1600	1482	89	29	0	3	99.592	96
90	1600	1506	64	30	0	3	111.915	108
100	3000	2842	93	65	0	2	124.200	120

Table 1: Experimental results on triangle free cubic graphs.

5 Concluding remarks

In this paper we have presented a parallel approximation algorithm for the *maximum cut problem* on cubic graphs. Due to its combinatorial nature, the algorithm appears to be very simple and efficient. It runs on a CRCW P-RAM with $O(n)$ processors in $O(\log n)$ time and achieves an approximation ratio $1.\bar{3}$, improving the best known parallel approximation ratio, i.e., 2, in the special class of cubic graphs. Moreover, the size of the cut returned by the algorithm is at least $\frac{9g-3}{8g}n$, where g is the odd girth of the input graph. This lower bound is valid for every cubic graph, even if for triangle-free cubic graphs is not as good as the best known theoretical bound.

We conclude this paper presenting possible directions for further research. The study of the very few instances for which the algorithm does not gain the $\frac{6}{5}n$ bound might provide useful insights for improving the algorithm itself and hopefully for achieving the $\frac{6}{5}n$ lower bound on each triangle-free instance. Furthermore, a generalization of the idea behind the algorithm to d-regular graphs, for $d > 3$, might improve the approximation ratio of the max cut problem on a larger class of graphs.

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] Erdös, P.: On some of my favourite problems in graph theory and block designs. *Le Mathematiche*, XLV, 1990, pp 61–74.

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

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

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

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

[9] 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.

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

[11] Gyori, E.–Kostochka, A.V.–Luckzak, T.: Graphs without short odd cycles are nearly bipartite. *Discrete Mathematics*, 163, 1997, pp 279–284.

[12] Hopkins, G.–Staton, W.: Extremal Bipartite Subgraphs of Cubic Triangle-Free Graphs. *J. of Graph Theory*, 6, 1982, pp 115–121.

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

[14] Johnson, E.L.: A proof of the four-coloring of the edges of a regular three-degree graph. *O.R.C., 63-28 (R.R.) Mimeographed Report*, Operation Research Center, University of California, 1963.

[15] 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.

[16] 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.

[17] Locke, S.C.: Maximum k-Colorable Subgraphs. *Journal of Graph Theory*, 6, 1982, pp 123–132.

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

[19] 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.

[20] Ore, O.: The four-color problem. *Academic Press*, 1967.

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

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

[23] 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.

[24] 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.