Università degli Studi di Roma Tre
Dipartimento di Informatica e Automazione
Via della Vasca Navale, 79 – 00146 Roma, Italy

The Complexity of the Matching-Cut Problem

Maurizio Patrignani[1] and Maurizio Pizzonia[2]

RT-DIA-35-98 **Luglio 1998**

(1) patrigna@dia.uniroma3.it

(2) pizzonia@dia.uniroma3.it

Research supported in part by the ESPRIT LTR Project no. 20244 - ALCOM-IT and by the CNR Project "Geometria Computazionale Robusta con Applicazioni alla Grafica ed al CAD."

ABSTRACT

Finding a cut or finding a matching in a graph are so simple problems that hardly are considered problems at all. Finding a cut whose split edges are a matching is expected to be a more difficult task. In this paper, by means of a reduction from the NAE3SAT problem, we prove, in fact, that this is an NP-complete problem. It remains intractable even if we impose the graph to be simple (no multiple edges or self-loops allowed) or its maximum degree to be k, with $k \geq 4$. On the contrary, we give a polynomial time algorithm that computes a matching-cut of an SP-graph. It's open whether the problem is tractable or not for planar graphs.

1 Introduction

Divide and conquer is a widely adopted algorithmic technique consisting of efficiently solving problems by recursively splitting them into subproblems and combining the so obtained solutions together. When the problem instance can be represented by a graph, such an approach may require to perform a cut on the graph according to some constraints or properties.

In this paper we address the problem of finding a cut in a graph that does not involve two adjacent edges. We propose this problem as an interesting combinatorial problem *per se*, although it firstly arose in a practical application, when studying a new approach to three-dimensional orthogonal graph drawing, based on generating the final drawing through a sequence of steps [4, 1]. Starting from a "degenerate" drawing, in which all vertices and edges are overlapped on the same point, at each step the drawing "splits" into two pieces and finds a structure more similar to its final version. The observation that a cut through non adjacent edges yields a more efficient (in terms of time of computation) and effective (in terms of volume occupation, number of bends introduced, and average edge length) performance, stimulated the investigation of the complexity of finding such a cut.

The paper is organized as follows: in Section 2 basic definitions are given; in Section 3 the problem of finding a matching-cut is demonstrated to be NP-complete. Section 4 shows that the problem retains its complexity even if we impose the graph to be simple or its maximum degree to be k, with $k \geq 4$. In Section 5 we produce a polynomial time algorithm to find a matching-cut in an SP-graph.

2 Preliminaries

A *graph* $G(V, E)$ consists of a set V of vertices and a set E of edges (u, v), such that u and v are vertices in V. In a *simple* graph every edge concerns two different vertices and two edges share at most a vertex.

An edge (u, v) is said to be *incident* to vertices u and v. Two edges are *adjacent* if they are incident to the same vertex v. The *degree* of a vertex v is the cardinality of the set of its incident edges. A *path* from u to v is a sequence of edges $e_1, \ldots, e_k$ such that, e_1 is incident to u, e_k is incident to v, and for every $i = 1, \ldots, k-1$, e_i is adjacent to e_{i+1}. A graph is *connected* if a path exists between each pair of vertices. In the following we consider only connected graphs.

A *cut* of a graph is a partition of its vertices set V in two non-empty sets. Note that a cut of a graph with n vertices can be associated with an array of n elements with values in $\{0, 1\}$, where all the elements with the same value correspond to the vertices of the same block of the partition. Since $(0, 0, \ldots, 0)$ and $(1, 1, \ldots, 1)$ are forbidden configurations (they don't correspond to partitions), and since there are two configurations corresponding to the same partition, it follows that there are $2^{n-1} - 1$ distinct cuts.

A cut of a connected graph can be alternatively characterized by specifying the set of edges connecting two vertices that do not belong to the same block of the partition. Observe that one description of the cut can be easily obtained from the other in linear time.

A *matching* in a graph G is a set $E' \subseteq E$ such that each pair of distinct edges e_i and

e_j of E' are not adjacent.

The problem of finding a cut in a graph can be solved in linear time: it suffices a random labeling of its vertices with the two labels 0 and 1. Imposing the cut to be maximum the problem becomes NP-complete ([2]), but approximable within 1.1383 ([3]).

Finding a matching can be solved in $O(1)$ time: each single edge is a matching.

Although finding a cut and finding a matching are simple problems, we expect the problem of finding a cut that is a matching to be a more difficult task. In the following section we will demonstrate that this problem is NP-complete.

3 NP-Completeness of The Matching-Cut Problem

Formally, the Matching-Cut problem is defined as follows:

> *Problem:* **Matching-Cut**
> *Instance:* A connected graph G(V,E).
> *Question:* Does a set $E' \subseteq E$, such that E' is a cut and a matching, exist?

We will demonstrate that this problem is NP-complete by demonstrating separately that it is in NP and that it is NP-hard.

3.1 The Matching-Cut Problem is in NP

Note that, given a cut, it's easy to check in linear time if it is a matching. In fact, supposing to have a label associated with each vertex to mark it's belonging to one set or to the other, we can proceed as follows:

1. consider one edge at a time. Let u and v be its incident vertices
2. if u and v have different colors
   (a) if both u and v are unmarked mark them
   (b) else return FALSE;
3. return TRUE;

By generating all possible cuts and by checking in linear time if a cut is a matching-cut, it follows that the matching cut problem is in NP.

3.2 The Matching-Cut Problem is NP-Hard

We will demonstrate that the Matching-Cut problem is NP-hard by means of a reduction from the NAE3SAT problem:

> *Problem:* **Not-All-Equal-3-Sat (NAE3SAT)**
> *Instance:* A set C of clauses, each containing 3 literals from a set of boolean variables.
> *Question:* Can truth values be assigned to the variables so that each clause contains at least one true literal and at least one false literal?

l_i	l_j	l_k	cut
0	0	0	no cut
0	0	1	A
0	1	0	B
0	1	1	C
1	0	0	D
1	0	1	E
1	1	0	F
1	1	1	no cut

Table 1: Truth assignment corresponding to the six cuts showed in Figure 4.

Given a formula ϕ in conjunctive normal form with variables $x_1, \ldots, x_v$ and clauses $C_1, \ldots, C_m$, each with three literals, we shall construct a graph $G(\phi)$ that admits a matching-cut if and only if ϕ is satisfiable in the NAE3SAT acceptation.

To begin with, we build two chains of $2c+v$ double linked vertices as shown in Figure 1.

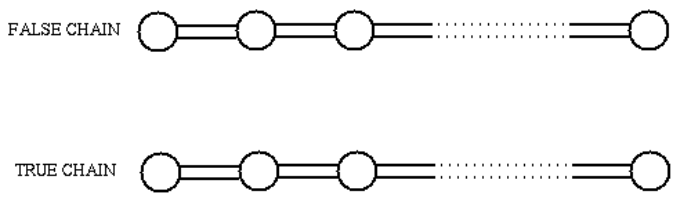


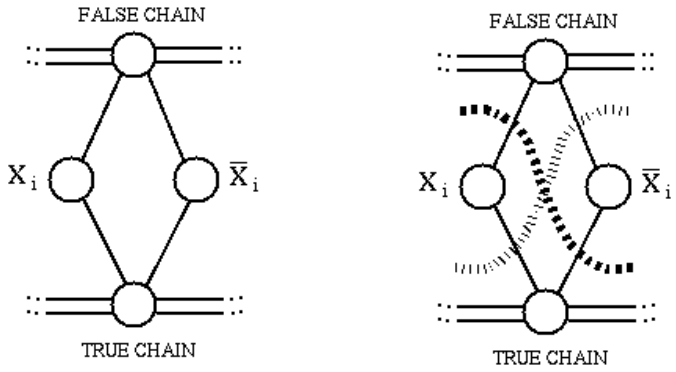
Figure 1: First step of the construction of $G(\phi)$: we create two chains of $2c + v$ double linked vertices.

The upper chain will be called in the following FALSE-chain, and the lower TRUE-chain. Since the vertices of each chain are linked with multiple edges, no matching-cut can separate two vertices of the same chain. So, each chain will belong to the same block of the partition that constitutes a matching-cut. We will build the remaining part of the graph $G(\phi)$ adding subgraphs connected both to the upper and to the lower chains in such a way that, if a matching-cut exists, it will necessarily separate the two chains cutting through all the inserted subgraphs. We call *TRUE-set* the block of the partition which the TRUE-chain belongs to, and *FALSE-set* the other.

The subgraphs to be introduced are of two types: the variable-gadget and the clause-gadget. Figure 2-a shows the variable-gadget. We introduce a variable-gadget for each boolean variable x_i. The only way to cut the variable-gadget with a matching-cut is along the lines of Figure 2-b. Each cut leaves the x_i vertex and the $\bar{x}_i$ vertex on the two opposite sets. Any other cut is illegal, since it isn't a matching.

Figure 3 shows the clause-gadget. We will introduce a clause-gadget for each clause of the ϕ formula. The three black vertices of Figure 3 correspond to the literals l_i, l_j, and l_k of the clause $(l_i \vee l_j \vee l_k)$. There are six ways to cut through the clause-gadget with a matching-cut, as Figure 4 shows. Each cut corresponds to the truth assignment for the literals illustrated in Table 1.

Note that there isn't a matching-cut for the clause-gadget that leaves all the three vertices corresponding to the literals on the same side, and that there is no other matching-



(a) (b)

Figure 2: The variable-gadget for variable x_i (a) and its two matching-cuts (b).

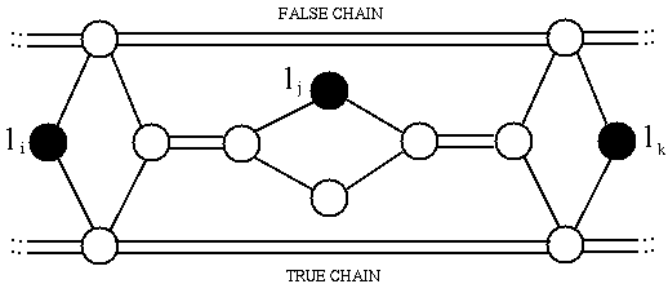


Figure 3: The clause-gadget for clause $(l_i \vee l_j \vee l_k)$.

cut that could separate a subgraph of the clause-gadget. To end the construction of $G(\phi)$, each vertex representing a literal l_i, l_j, or l_k of the clause-gadget is connected to the vertex representing the corresponding literal of the variable-gadget by means of two edges, so to impose that they will belong to the same block of the partition determined by the matching-cut. As an example, Figure 5 shows a construction for a formula with three boolean variables and a single clause.

We claim that a matching-cut can be found in the graph $G(\phi)$ if and only if the corresponding instance ϕ of the NAE3SAT problem has a solution.

Suppose that such a cut exists. Since the chains can not be cut and since no lesser part of the gadgets can be separated from the rest of the graph, the matching-cut has to separate the two chains, cutting through all the gadgets that tie them together. All the variable-gadgets are attached to the FALSE-chain and to the TRUE-chain, so the matching-cut must cut through their edges in one of the two ways shown in Figure 2-b, determining a truth assignment of the boolean variables. Putting a variable-vertex on a set implies that the corresponding literals in the clauses belong to the same set, and the negate literals belong to the other set. The clause-gadget too are cut in two by the

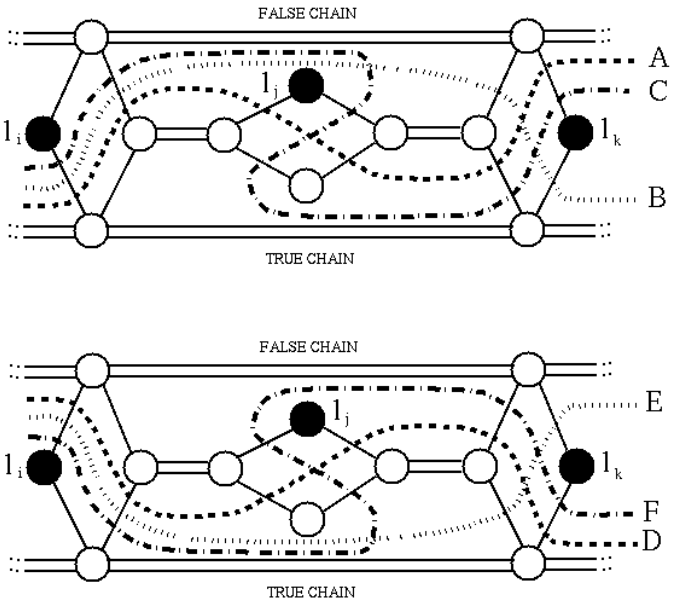


Figure 4: The clause-gadget for clause $(l_i \vee l_j \vee l_k)$.

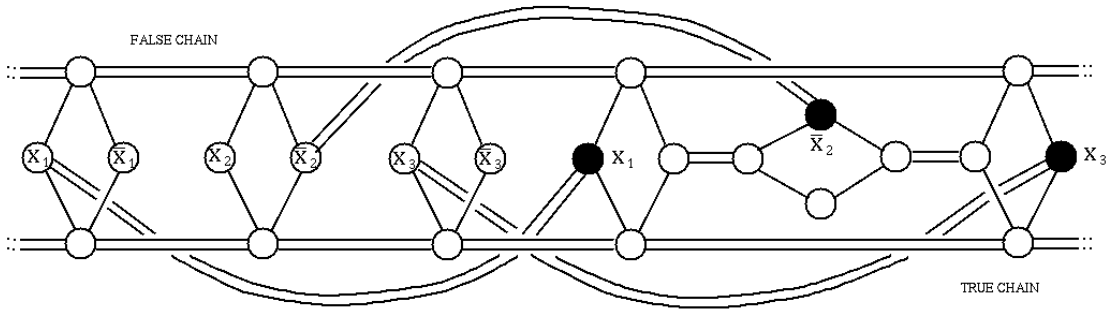


Figure 5: Construction for the first three variables and a clause $(X_1 \vee \overline{X}_2 \vee X_3)$.

matching-cut, and necessarily in one of the six ways illustrated in Figure 4, implying that at least one of the literal-vertices belongs to the TRUE-set and the other to the FALSE-set. So each clause, and the ϕ formula, is satisfied as requested by the NAE3SAT problem.

Conversely, if a truth assignment exists such that each clause has a true literal and a false one, a matching-cut can be found cutting through the variable-gadgets so to leave the variable-vertex in the set corresponding to its boolean value, the negate literal will be in the other set, and cutting through the clause-gadget by using necessarily one of the six cuts of Table 1 corresponding to the truth assignment of the literals. It's easy to verify that the cut so obtained is a matching-cut. The following theorem is therefore demonstrated:

Theorem 1 *The Matching-Cut problem for a connected graph G is NP-complete.*

4 Simple graphs and graph with maximum degree four

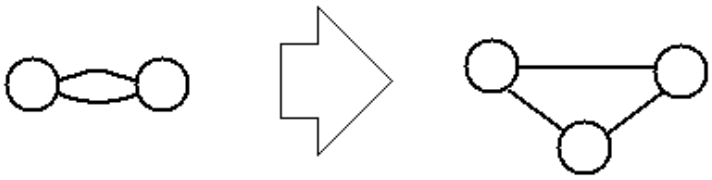


Figure 6: Replacing an edge with a path of two edges to eliminate double edges.

We easily modify the previous construction to demonstrate that if we impose the graph to be simple, the problem retains its complexity. Namely, we note that between a pair of vertices of the graph $G(\phi)$ there can be a single edge or two, and in the latter case we can replace one of the two edges with a two edges long path, as shown in Figure 4. It's easy to see that the three vertices of the so formed triangle are bounded to be in the same set of the bipartition imposed by a matching-cut. So, the following corollary is demonstrated:

Corollary 1 *The Matching-Cut problem for a simple connected graph G is NP-complete.*

We modify the same construction to demonstrate that the problem remains NP-complete even imposing a maximum degree k, with $k \leq 4$ for the vertices of the graph. Observe that the vertices of the proposed construction have even degree. Also, edges are introduced two-by-two in order to impose that the double linked vertices will belong to the same set of the bipartition determined by a matching-cut. So, each vertex with degree greater than 4 can be replaced with the star construction of Figure 7, in which every vertex has degree four. It's easy to see that a matching-cut can not separate the vertices of the star-shaped subgraph of Figure 7.

Corollary 2 *The Matching-Cut problem for a connected graph G (simple or not) of degree k, with $k \geq 4$ is NP-complete.*

5 SP-graphs

In this section we describe a polynomial time algorithm that finds a matching-cut in an SP-graph, if such a cut exists.

An *SP-graph* can be built with the following recursive rules:

base step the graph consisting of a single edge between two vertices (s and t) is an SP-graph;

parallel composition given two SP-graphs $G_{top}(s_1, t_1)$ and $G_{bottom}(s_2, t_2)$ a new SP-graph can be obtained by means of two "vertex-fusions" after which $s = s_1 = s_2$ will be the source and $t = t_1 = t_2$ will be the sink of the new SP-graph.

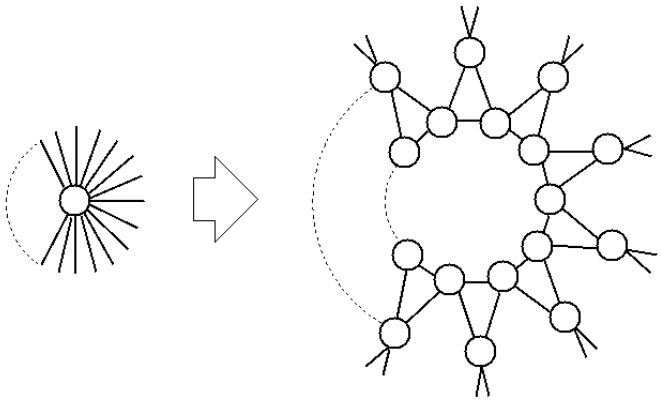


Figure 7: Vertices with degree greater that 4 can be replaced with the star-shaped graph of this figure to demonstrate that the problem remains NP-complete even for graph of degree greater or equal that four.

serial composition given two SP-graphs $G_{left}(s_1, t_1)$ and $G_{right}(s_2, t_2)$, we obtain a new SP-graph $G(s_1,t_2)$ by means of one vertex-fusion $t_2 = s_1$.

Each SP-graph can be represented with a binary SPQ-tree in which nodes are labeled as follows:

S nodes represent serial compositions, the left subtree is associated with G_{top} and the right subtree is associated with G_{bottom}

P nodes represent parallel compositions, the left subtree is associated with G_{left} and the right subtree is associated with G_{right}.

Q nodes represent graphs composed by a single edge.

Many different SPQ-trees may represent the same SP-graph because parallel or serial compositions may be grouped in different ways. We represent SP-graph with right binary trees, so implicitly we state that bottom serial composition groups before the top one and that right parallel composition groups before the left one. Given such constraints the right binary representation tree is uniquely determined for each SP-graph. An algorithm to recognize in linear time if a graph is an SP-graph and to build its SPQ-tree can be found in [5].

5.1 The algorithm for SP-graphs

Our algorithm to test if a matching-cut exists in a given SP-graph relays on a simple post order traversal of the binary SPQ-tree for such graph. Before describing the details of the algorithm we must state some definitions.

We say that a matching-cut is *running-through* the SP-graph if it leaves s and t in two distinct sets. If a running through matching-cut exists in a component the vertices that constitute the separation pair may remain *engaged*, i.e., no other edge of the matching-cut can be incident to those vertices because there already exists one in the matching-cut.

We associate two labels $l_1(G)$ and $l_2(G)$ to each SP-graph G, that is a subgraph of the input graph. Label $l_1(G)$ signals under which conditions graph G has at least one running-through matching-cut, and has one of the following values:

0	there does not exist a running-through matching-cut for such component;
s	there exists at least one running-through matching-cut but s is necessarily engaged
t	there exists at least one running-through matching-cut but t is necessarily engaged
$s \cdot t$	there exists at least one running-through matching-cut but both s and t result necessarily engaged
$s \oplus t$	there exists more than one running-through matching-cut and either s or t result engaged (i.e. a choice is possible)
1	there exists at least one running-through matching-cut and neither s nor t result engaged

Label $l_2(G)$ signals if G admits at least one non running-through matching-cut. It may have value 1 (for true) or 0 (for false).

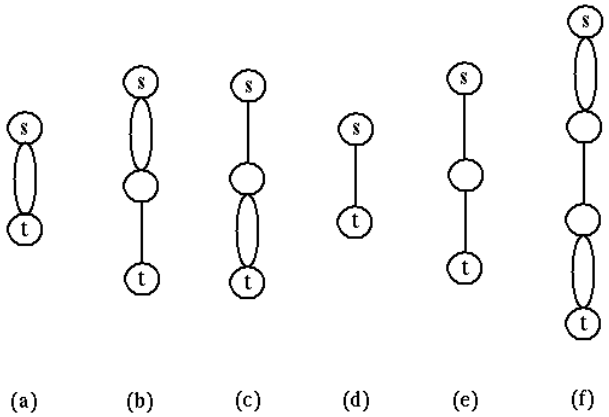


Figure 8: Examples of graphs for which no matching-cut exists (a), only a t-engaging matching-cut exists (b), only an s-engaging matching-cut exists (c), only an s and t engaging matching-cut exists (d), a matching-cut engages s or t (e), and a matching-cut not engaging s nor t exists (f).

The labels l_1 and l_2 may be computed recursively for each node of the SPQ-tree as follows:

Q nodes have $l_1 = s \cdot t$ and $l_2 = 0$.

S nodes have labels depending on the values of the labels of their children G_{up} and G_{down}. The following table defines $l_1(G)$:

$l_1(G)$ / $l_1(G_{up})$ / $l_1(G_{down})$	0	$s \cdot t$	s	t	$s \oplus t$	1
0	0	s	s	1	1	1
$s \cdot t$	t	$s \oplus t$	$s \oplus t$	1	1	1
s	1	1	1	1	1	1
t	t	$s \oplus t$	$s \oplus t$	1	1	1
$s \oplus t$	1	1	1	1	1	1
1	1	1	1	1	1	1

The label $l_2(G)$ is 1 if $l_2(G_{up}) = 1$ or $l_2(G_{down}) = 1$, otherwise its value is given by the following table:

$l_2(G)$ / $l_1(G_{up})$ / $l_1(G_{down})$	0	$s \cdot t$	s	t	$s \oplus t$	1
0	0	0	0	0	0	0
$s \cdot t$	0	0	1	0	1	1
s	0	0	1	0	1	1
t	0	1	1	1	1	1
$s \oplus t$	0	1	1	1	1	1
1	0	1	1	1	1	1

P nodes have labels depending on the values of the labels of their children G_{left} and G_{right}. The value of $l_1(G)$ is given in the following table:

$l_1(G)$ / $l_1(G_{right})$ / $l_1(G_{left})$	0	$s \cdot t$	s	t	$s \oplus t$	1
0	0	0	0	0	0	0
$s \cdot t$	0	0	0	0	0	$s \cdot t$
s	0	0	0	$s \cdot t$	$s \cdot t$	s
t	0	0	$s \cdot t$	0	$s \cdot t$	t
$s \oplus t$	0	0	$s \cdot t$	$s \cdot t$	$s \cdot t$	$s \oplus t$
1	0	$s \cdot t$	s	t	$s \oplus t$	1

The label $l_2(G)$ is 1 only if $l_2(G_{left}) = 1$ or $l_2(G_{right}) = 1$. Otherwise its value is 0.

Lemma 1 *A matching-cut in an SP-graph G exists iff (i) $l_1(G)$ is different from 0 or (ii) $l_2(G) = 1$, where $l_1(G)$ and $l_2(G)$ are recursively computed as described above.*

Proof: The sufficiency of the condition follows immediately from the definition of $l_1(G)$ and $l_2(G)$, and from the correctedness of the tables given above.

Conversely, suppose a matching-cut exists. It may be a running-through matching-cut or not. In the first case $l_1(G)$ must be necessarily different from zero. In the second case consider the partition of the vertices of the graph G determined by the cut. One of the two sets, say V_0, does not contain s nor t. Two are the cases: the graph $G_0(V_0)$

induced by V_0 is connected or not. If not, consider one of its connected components. It defines necessarily a new matching-cut that is not running-through and that results in a partition in two connected blocks. So without loss of generality we can consider G_0 connected. Now, lets call G_1 the the smallest node of the SPQ-tree of G containing G_0. Obviously, node G_1 can't be a Q node. Observe that G_1 can't be neither a P node, because of the connection of G_0 and the minimality of G_1. So G_1 is an S node. Consider the two children G_{up} and G_{down} of G_0 in the SPQ-tree. For the connection of G_0 and the minimality of G_1 the cut must pass through G_{up} and G_{down}, which imply $l_2(G_1) = 1$. It follows that all ancestors of G_1 will have label $l_2(\cdot) = 1$, and so $l_2(G) = 1$.

Theorem 2 *There exists a polynomial time algorithm to verify if an SP-graph admits a matching-cut.*

Proof: It suffices observing that the number of the nodes of the SPQ-tree associated to the graph is bounded by $2m$, and therefore a post order traversal can be performed in linear time. The values of $l_1(G_i)$ and $l_2(G_i)$ for each node G_i of the SPQ-tree can be computed in constant time, depending only on the values of the children of G_i.

To actually compute a matching-cut (if any) for a given SP-graph we need some more structures: we associate to each node G_i of the SPQ-tree three sets $S_{i,1}$, $S'_{i,1}$, and $S_{i,2}$ of edges of the graph G. The set $S_{i,1}$ represents an existing matching-cut according to the constraints expressed by the label $l_1(G_i)$. We need $S'_{i,1}$ to represent an alternative matching-cut when the label $l_1(G_i)$ is $s \oplus t$, with the convenction that, in such a case, $S_{i,1}$ represents a running-through matching cut engaging s and $S'_{i,1}$ represents a running-through matching cut engaging t. The set $S_{i,2}$ represents an existing non running through matching-cut (if any). In calculating the values of the labels $l_1(G_i)$ and $l_2(G_i)$ we update the three sets above from the respective sets of the children, by taking one of the two, or by making their union as the case is. Observe that it's easy to devise an implementation so that the set union and the copying operations take constant time (provided that old sets are no more needed).

It follows that the problem of finding a matching cut in an SP-graph can be solved in polynomial (in fact, linear) time.

6 Conclusions and Open Problems

We studied the problem of finding a matching-cut in a graph, and showed that in the general case the problem is in NP, and that retains its complexity even if the graph is simple or its maximum degree is bounded by a constant $k \geq 4$.

As for other classes of graphs, we produced an algorithm to find a matching cut (if any) in an SP-graph in linear time, so demonstrating that the problem is in P for this particular class of graphs. It's an open problem demonstrating if the case of planar graph si tractable or not.

Acknowledgements

We are grateful to Giuseppe Di Battista and to Walter Didimo for their encuragement and support.

References

[1] G. Di Battista, M. Patrignani, and F. Vargiu. A split&push approach to 3D orthogonal drawing. In S. Whitesides, editor, *Graph Drawing (Proc. GD '98)*, Lecture Notes Comput. Sci. Springer-Verlag. to appear.

[2] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, 1979.

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

[4] Maurizio Patrignani and Francesco Vargiu. 3DCube: A tool for three dimensional graph drawing. In G. Di Battista, editor, *Graph Drawing (Proc. GD '97)*, volume 1353 of *Lecture Notes Comput. Sci.*, pages 284–290. Springer-Verlag, 1998.

[5] J. Valdes, R. E. Tarjan, and E. L. Lawler. The recognition of series-parallel digraphs. *SIAM J. Comput.*, 11(2):298–313, 1982.