The Complexity of the Matching-Cut Problem[*]

Maurizio Patrignani and Maurizio Pizzonia

Dipartimento di Informatica e Automazione,
Università di Roma Tre, via della Vasca Navale 79,
00146 Roma, Italy.
{patrigna,pizzonia}@dia.uniroma3.it

Abstract

Finding a cut or finding a matching in a graph are so simple problems that they are hardly considered problems at all. In this paper, by means of a reduction from the NAE3SAT problem, we prove that combining these two problems together, i.e., finding a cut whose split edges are a matching is an NP-complete problem. It remains intractable even if we impose the graph to be simple (no multiple edges 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 and for graphs with maximum degree 3.

1 Introduction

We consider two elementary graph-theoretic problems: finding a matching in a graph, i.e., a set of disjoint edges; and finding a cut in a graph, i.e., a set of edges whose removal disconnects the graph. These two problems can be separately solved in so a straightforward way that they are hardly considered problems at all: an arbitrary edge is a matching, and the set of edges incident to an arbitrary node is a cut. However, we show that combining these two problems together, that is, finding a cut that at the same time is a matching, yields an NP-complete problem.

The problem remains intractable even if we impose the graph to be simple (no multiple edges 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 and for graphs with maximum degree 3.

Although the problem we address is an interesting combinatorial problem *per se*, it firstly arose in a practical application.

In fact, in [4, 1] is described a new approach to three-dimensional orthogonal graph drawing, based on generating the final drawing through a sequence of steps. The method starts 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 involving 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 on the complexity of finding such cut.

[*]Research supported in part by the Murst Project: "Algorithms for Large Data Sets".

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.

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. In a *simple* graph every edge incides on two distinct vertices and two edges share at most a vertex. A *path* from u to v is a sequence of edges $e_1, \dots, e_k$ such that, e_1 is incident to u, e_k is incident to v, and for every $i = 1, \dots, k-1$, 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 vertex set V into 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 $(0, 0, \dots, 0)$ and $(1, 1, \dots, 1)$ 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.

If the graph is connected, a cut can be alternatively characterized by specifying the set of edges connecting 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?

Given a cut, it's easy to check in linear time if it is a matching by marking the extremes of each edge belonging to the cut. If no vertex is marked twice, the cut is also a matching. A non-deterministic Turing machine may generate all possible cuts and check in linear time if the cuts are matching-cuts. It follows that the matching cut problem is in NP.

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?

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

We start by building two chains of $2m + v$ double linked vertices as shown in Figure 1.

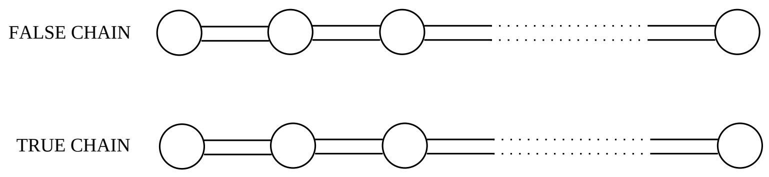

Figure 1: First step of the construction of $G(\phi)$: we create two chains of $2c + v$ double linked vertices.

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, and 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 not allowed, since it is not a matching.

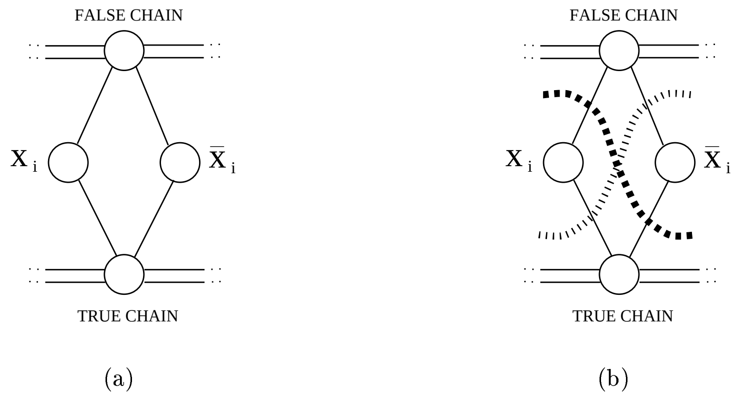

Figure 2: The variable-gadget for variable x_i (a) and the only two possible matching-cuts (b).

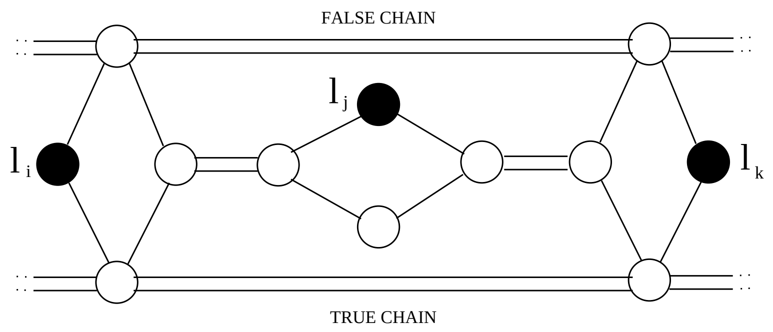

Figure 3: The clause-gadget for clause $(l_i \vee l_j \vee l_k)$.

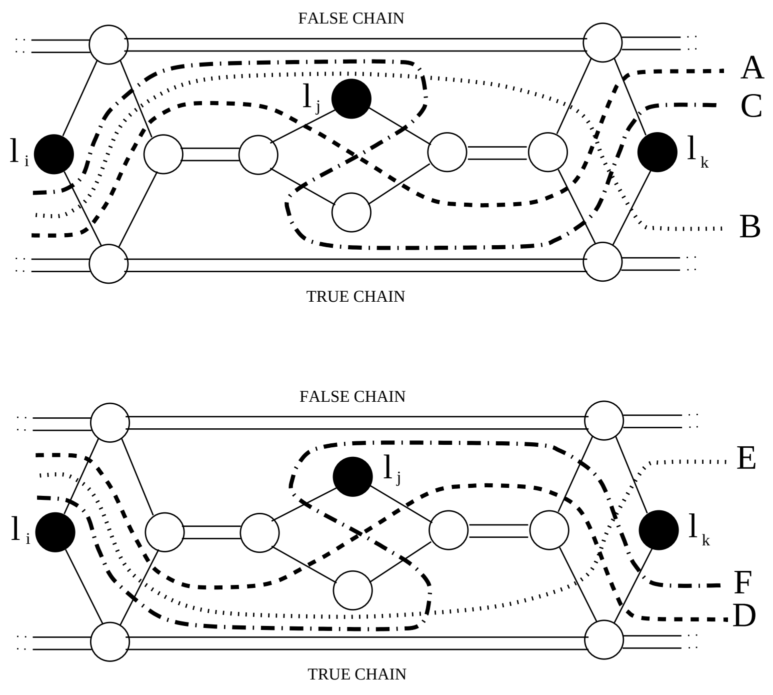

Figure 4: The clause-gadget for clause $(l_i \vee l_j \vee l_k)$.

l_i	l_j	l_k	cut
false	false	false	no cut
false	false	true	A
false	true	false	B
false	true	true	C
true	false	false	D
true	false	true	E
true	true	false	F
true	true	true	no cut

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

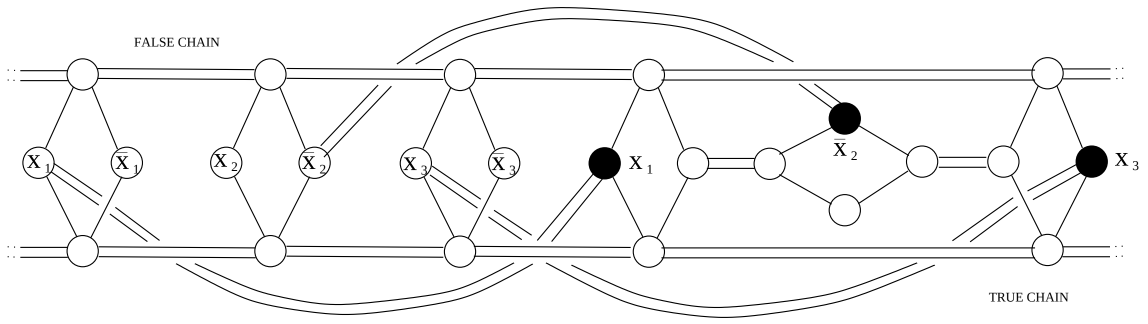

Figure 5: Construction for the first three variables and a clause $(X_1 \vee \overline{X}_2 \vee X_3)$.

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 is no 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-cut that could separate a subgraph of the clause-gadget.

Finally, 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. Figure 5 shows the whole 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 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. Leaving 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 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

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 6.

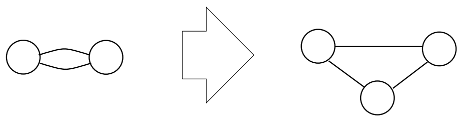

Figure 6: Replacing an edge with a path of two edges to eliminate double edges.

It's easy to see that the three vertices of the so formed triangle are forced to be in the same set of the bipartition induced 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 Matching-cuts of 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.

A graph G with no self-loop is *series-parallel* (or *SP-graph*) if two of its vertices s and t are selected to be respectively the *source* and the *sink* of G (denoted $G(s,t)$), and G can be built by recursively applying the following rules:

basic step graph $G(s,t)$ consists of a single edge between s and t;

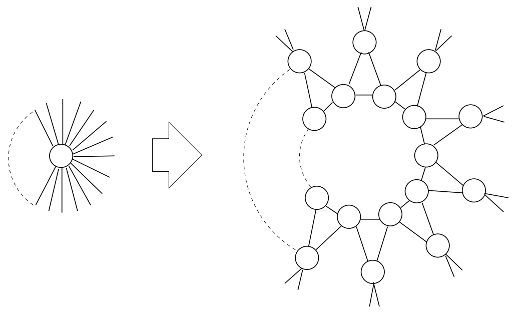

Figure 7: Vertices with degree greater that 4 can be replaced with the star-shaped graph of this figure to demonstrate that the problem remains NP-complete even for graph of degree greater or equal that four.

serial composition graph $G(s,t)$ is the union of two SP-graphs $G_u(s_u,t_u)$ and $G_d(s_d,t_d)$ such that $s = s_u$, $t_u = s_d$, and $t = t_d$.

parallel composition graph $G(s,t)$ is the union of two SP-graphs $G_l(s_l,t_l)$ and $G_r(s_r,t_r)$ such that $s = s_l = s_r$ and $t = t_l = t_r$.

Each SP-graph can be represented by a binary tree [5] in which nodes are or three types: S, P, and Q. Their meaning is the following:

- An S-node represents the serial composition of the graphs corresponding to its children, with the convention that the left child represents G_u and the right child represents G_d
- A P-node represents the parallel compositions of the graphs corresponding to its children, with the convention that the left child represents G_l and the right child represents G_r
- A Q-node represents a single edge

In order to guarantee the uniqueness of such tree for a given SP-graph, we further require that if a node n_1 and its parent n_2 have the same type, then n_1 is a right child of n_2. Such tree is called *parse tree*. It is possible to recognize an SP-graph (and build its parse tree) in linear time [5].

We associate each node of the parse tree with an SP-graph. For S-nodes and P-nodes the associated SP-graph is obtained by applying the rule represented by the node to the SP-graphs associated with the children. For brevity, in the following we do not distinguish between the node of an parse tree and the graph associated with it.

Our algorithm takes as input an SP-graph G and returns as output the matching-cut of G if it exists. It is based on a post-order traversal of the parse tree. In the following we say that a for a given SP-graph a matching-cut is *st-separating* if it leaves s and t in distinct sets. We say that vertex s (t) is *engaged* by a st-separating matching-cut if one of the edges of the matching-cut is incident to s (t).

We associate two labels $l_1(G')$ and $l_2(G')$ with each node $G'(s,t)$ of the parse tree of G. Label $l_2(G')$ signals if G' admits at least one non st-separating matching-cut ($l_2(G') = 1$) or not ($l_2(G') = 0$). Note that, if $l_2(G') = 1$ then for each ancestor G'' of G' it must hold $l_2(G'') = 1$.

Label $l_1(G')$ signals whether G' has at least one st-separating matching-cut and whether nodes s and t are necessarily engaged. The six possible values of label $l_1(G')$ are listed below along with their meanings.

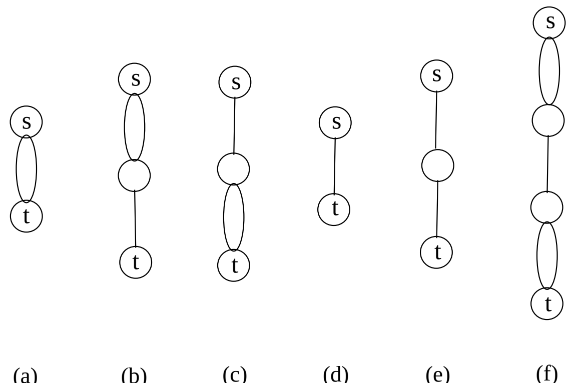

Figure 8: Examples of SP-graph with its l_1 labeling. (a) No matching-cut exists: $l_1 = 0$. (b) There exsist at least one matching-cut engaging only t: $l_1 = t$. (c) There exsist at least one matching-cut engaging only s: $l_1 = s$. (d) Any matching-cut engages both s and t: $l_1 = s \cdot t$. (e) There exsist both matching-cuts engaging s and matching-cuts engaging t: $l_1 = s \oplus t$. (f) A matching-cuts exists which does not engage neither s nor t: $l_1 = 1$.

G_l / G_r	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

(a) Parallel composition

G_u / G_d	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
t	t	$s \oplus t$	$s \oplus t$	1	1	1
s	1	1	1	1	1	1
$s \oplus t$	1	1	1	1	1	1
1	1	1	1	1	1	1

(b) Serial composition

Figure 9: Tables for computing $l_1(G)$ from the labels of the children of G.

0 There does not exist an st-separating matching-cut for such node.

$s \cdot t$ There exists at least one st-separating matching-cut and in all matching-cuts both s and t are engaged.

s There exists at least one st-separating matching-cut. In all matching-cuts s is engaged. Further, in some of them, t is not engaged.

t There exists at least one st-separating matching-cut. In all matching-cuts t is engaged. Further, in some of them, s is not engaged.

$s \oplus t$ There exists at least one st-separating matching-cut that engages s and not t and there exists at least one st-separating matching-cut that engages t and not s. Further, no matching-cut that does not engage either s or t exist.

1 There exists at least one st-separating matching-cut in which neither s nor t are engaged.

The labels l_1 and l_2 may be computed recursively for each node G, applying one of the following rules, depending on the type of the node.

Q-node $l_1(G) = s \cdot t$ and $l_2 = 0$.

P-node Let G_l and G_r be the children of G. The value of $l_1(G)$ is given in the table shown in Figure 9(a) which summarizes the following observations. If one between G_l and G_r does not admit an st-separating matching-cut, so does G. If G_l (G_r) admits an unconditioned st-separating matching-cut then $l_1(G) = l_1(G_r)$ ($l_1(G) = l_1(G_l)$). In all other cases G may have an st-separating matching-cut if and only if it is possible to find two matching-cuts, one in G_l and one in G_r, that do not engage the same vertex (s or t). Hence, in such cases, $l_1(G) = s \cdot t$.

The label $l_2(G)$ is 1 if and only if $l_2(G_l) = 1$ or $l_2(G_r) = 1$.

S-node Let $G_u(s_u, t_u)$ and $G_d(s_d, t_d)$ be the children of G. The value of $l_1(G)$ is given in the table shown in Figure 9(b) which summarizes the following observations. Recall that $t_u = s_d$ and we call v such vertex. If $l_1(G_u) = l_1(G_d) = 0$ then G does not have an st-separating matching-cut. In all other cases conditions on v are not propagated to $l_1(G)$, while conditions on s_u and t_d are. Note that, since an st-separating matching-cut either st-separates G_u or st-separates G_d it can never be $l_1(G) = s \cdot t$.

The label $l_2(G)$ is 1 if and only if either $l_2(G_u) = 1$ or $l_2(G_d) = 1$, or if it is possible to find two st-separating matching-cut, one in G_u and one in G_d, with the condition that no both matching-cut engage v. Such observations are summarized by the table in Figure 10:

G_d \ G_u	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
t	0	1	1	1	1	1
s	0	0	1	0	1	1
$s \oplus t$	0	1	1	1	1	1
1	0	1	1	1	1	1

Figure 10: The table for computing $l_2(G)$ from the labels of the children of G.

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

Proof: (sketch) The correctness of the algorithm follows from the correctness of the recursive construction of the labels l_1 and l_2 described above. For the time complexity, it suffices observing that the number of the nodes in the parse tree associated with an SP-graph of m edges 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 parse tree can be computed in constant time, since they depend only on the values of its two children.

Further, the algorithm can be easily modified as follows in order to actually return a matching-cut if it exists. Given the input SP-graph, we associate with each node $G(s,t)$ of its parse tree three sets S_1, S_1', and S_2. Such sets are subsets of the edges of the input SP-graph. If $l_1(G) = s \cdot t$ then subset S_1 is a matching-cut for G. If $l_1(G) = s$ ($l_1(G) = t$) then subset S_1 is a matching-cut that engages s (t) only. If $l_1(G) = s \oplus t$ then S_1 is a matching-cut that engages s only, and subset S_1' is an alternative matching-cut that engages t only. If $l_1(G) = 1$ then subset S_1 is a matching-cut that engages neither s nor t. If $l_2(G) = 1$ then S_2 is a non st-separating matching-cut.

For each node G of the parse tree the three sets S_1, S'_1, and S_2 can be updated in the same post-order traversal that calculates the values of the labels l_1 and l_2 by suitably combining the sets associated with the children of G. Since the sets of the children are not needed anymore, the involved set operations (union and copy) may be performed in constant time. 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 NP-complete, and that it 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 described 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 open whether the problem is tractable or not for planar graphs and for graphs with maximum degree 3.

Acknowledgements

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

References

[1] G. Di Battista, M. Patrignani, and F. Vargiu. A Split&Push approach to 3D orthogonal drawing. *Journ. Graph Alg. Appl.*, 4:105–133, 2000.

[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] M. Patrignani and F. 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, 1997.

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