Parallel Approximation Schemes for Problems on Planar Graphs

J. Díaz
M. J. Serna
J. Torán

Report LSI–93–10–R

Parallel Approximation Schemes for problems on planar graphs *

J.Díaz† M.J.Serna† J. Torán†

Abstract

This paper describes a technique to obtain NC Approximations Schemes for the Maximun Independent Set in planar graphs and related optimization problems.

1 Introduction

The generalized conjecture that no NP-hard problem can be solved in polynomial time has motivated the study of fast approximation algorithms for NP-hard optimization problems. In this way, optimization problems have been classified according to their degree of approximability. Basically, the NP-hard optimization problems that can be approximated in polynomial time to any degree of (relative) accuracy are divided in the classes PTAS (Polynomial time approximation Schemes) and FPTAS (Fully Polynomial time approximation Schemes). The difference between both definitions is that for a problem to be in PTAS there must be an algorithm that given as input a pair (x, ϵ), where x is an instance of the problem, and $\epsilon \in \mathbb{R}^+$, produces a solution to the problem within $1+\epsilon$ of the optimum, running in time that is polynomial in the length of x (and may depend arbitrarily on ϵ), while in the case of FPTAS it is required that the running time of the algorithm is polynomial in both the length of x and $1/\epsilon$. Asymptotic versions of the approximation classes have also been defined: FPTAS$^\infty$ and PTAS$^\infty$. Their definition is as before, but here ϵ is a function on the instance length instead of part of the input. As usual, asymptotically means that $\epsilon(|x|) \to 0$

*This research was supported by the ESPRIT BRA Program of the EC under contract no. 7141,project ALCOM II.

†Departament de Llenguatges i Sistemes, Universitat Politècnica Catalunya, Pau Gargallo 5, 08028-Barcelona

as $|x| \to \infty$ [GJ79]. A good survey on classes of problems approximable by polynomial time algorithms is given in Chapter 4 of Kann's dissertation [Kan92].

When dealing with polynomial time approximability, a natural question to consider is whether the full power of P is needed or there are approximation algorithms for NP-hard problems that use less resources. In particular we are interested in the study of parallel approximation algorithms, in the sense of NC; using polynomial number of processors and running in polylogarithmic time. Some work has been done on parallel approximation algorithms [AM86, KSS89, SS89, Ser90, Ser91, LN92]. It is even known that some NP-hard problems can be fully approximated in NC [AM86, KS93]. In particular, Peters and Rudolph obtained a *fully NC approximation schemes* for several Subset Sum and Knapsack Problems [PR87].

In the present paper, we present NC approximation schemes for the Maximum Independent Set on planar graphs, and some other related NP-hard optimization problems restricted to planar graphs, (Minimum Vertex Cover, Minimum Dominating Set, Minimum Edge Dominating Set, Maximum Cut, Maximum Matching and Maximum Even Degree Set) showing that they belong to the classes NCAS and NCAS$^\infty$. The class of optimization problems with an NC approximation scheme is defined in the following way. A problem is in NCAS if there is a parallel algorithm such that given as input a pair (x, ϵ), where x is an instance of the problem, and $\epsilon \in \mathbb{R}^+$, produces a solution to the problem within $1 + \epsilon$ of the optimum, running in polylogarithmic time in the length of x and using a number of processors bounded by a polynomial in the length of x. In the same way as before, we can define the asymptotic version NCAS$^\infty$.

The Maximum Independent Set problem on planar graphs was shown to belong to the class PTAS by Baker in [Bak83]. The idea in her proof is to decompose the graph in k-outerplanar graphs, and then using dynamic programming techniques obtain the MIS for each of the k-outerplanar graphs. Although we follow the same general scheme in our parallel approximation algorithms, the techniques we use are rather different, in particular we have to introduce original methods to obtain in parallel a representation of the k-outerplanar graphs s binary trees. The dynamic programming part in Bakers proofs is translated into a shunt operation in the binary trees representing the graphs. For any k the algorithm achieves a $(k-1)/k$ performance ratio using a polynomial number of processors (in n).

Chrobak and Naor gave an NC algorithm with a linear number of processors to approximate MIS within 1/2 [CN89]. In the same paper they

mention the convenience of obtaining a parallel version of Baker's result.

The advantage of our method is that it can be transformed (in the same way as Baker's algorithm) to obtain parallel approximation algorithms for a whole set of NP-hard planar graphs problems. In particular we show how the technique can be transformed to approximate Minimum Vertex Cover Minimum Dominating Set and Minimum Edge Dominating Set on planar graphs. We also show that it can be transformed to obtain NC approximations to some polynomial time computable problems, as Maximum Cut, Maximum Even Degree Set and Maximum Matching.

We also obtain exact NC algorithms for these problems as well as for the 3-Colorability and the Graph Bisection problems restricted to k-outerplanar graphs. To our knowledge this is the first time that these problems are proved to be in the class NC.

The paper is organized as follows; in section 2 we introduce the basic definitions, and explain the representation of outerplanar and k-outerplanar graphs. Section 3 contains the parallel approximation scheme for the special case of the MIS for planar graphs. The extension of the technique to other planar graph problems is sketched in Section 4.

The algorithms in this paper are designed for the Concurrent Read Exclusive Write PRAM model of computation [KR90, JJ92].

2 Definitions and Preliminaries

In this section we introduce the notation and basic facts used in the article. We introduce a method to represent a k-outerplanar graph G by a tree. The nodes of this tree will keep enough information to process the graph. Some of the basic definitions are taken from [Hag90].

2.1 Definitions

An *undirected graph* $G = (V, E)$ consists of a set of vertices V of size n and a set of edges E of size m. Each edge is an unordered pair (v, w) of disjoint vertices v and w. The subgraph *induced* by a set of vertices is formed by these vertices and every edge in E between them. A *path* joining v_1 and v_k is a sequence of vertices $v_1, v_2, \ldots, v_k$ such that $(v_i, v_{i+1}) \in E$, for $i = 1 \ldots k-1$. A *cycle* is a path $v_1, v_2, \ldots, v_k$ such that $(v_k, v_1) \in E$. A *simple cycle* is a cycle that does not contain any other cycle.

A graph G is *connected* if there is a path between every pair of vertices in V. The *connected components* of a non-connected graph are its maximal

connected subgraphs. An *articulation point* (or *cut point*) in a connected graph is a vertex v such that the graph obtained by removing v (and the edges incident with v) has at least two connected components. A *bridge* in a connected graph is an edge $e = (u, v)$ such that the graph obtained by removing e has at least two connected components. A *biconnected* graph is a connected graph containing no articulation point. Through the remaining of the paper, we shall be talking about undirected connected graphs.

A *planar embedding* of an undirected graph $G = (V, E)$ is a function $\mathcal{G}$ that maps the vertices of G to distinct points in $\mathbb{R}^2$ and each edge $\{u, v\} \in E$ to a Jordan curve in $\mathbb{R}^2$ from $\mathcal{G}(u)$ to $\mathcal{G}(v)$ such that for all $e = \{u, v\} \in E$, $\mathcal{G}(e) \cap (\mathcal{G}(V) \cup \mathcal{G}(E - \{e\})) = \{\mathcal{G}(u), \mathcal{G}(v)\}$. A graph G is *planar* if there exists a planar embedding of G. Let $\mathcal{G}$ be a planar embedding of G. The *faces* of $\mathcal{G}$ are the connected regions of $\mathbb{R}^2 - \mathcal{G}(V \cup E)$. Let $\mathcal{F}$ denote the set of faces in $\mathcal{G}$, and let f_∞ denote the *exterior face* of $\mathcal{G}$. Two faces of $\mathcal{G}$ are said to be *adjacent* if they share at least one edge of E. The *exterior boundary* $\mathcal{B}$ of $\mathcal{G}$ is the set of edges separating f_∞ from the other faces of $\mathcal{G}$.

The usual way to give an embedding of a graph is by the cyclic (counterclockwise) order of edges around each vertex. In the following we will assume that a planar graph G is given as a list of vertices and for each vertex a linked list of its neighbours, in the order specified by the embedding. (For details on the datastructure see for example [KX92].)

It is known that a plane embedding of a planar graph can be constructed in $O(\log^2 n)$ time by using $O(n)$ processors in the CREW PRAM model see [KR86].

Through the remaining of the paper, the term *embedded graph* will denote a planar graph G together with a particular planar embedding of G. Given an embedded graph G, the *quasidual graph* $G^* = (V^*, E^*)$ is the graph whose vertex set is the set $\{\mathcal{F} - f_\infty\} \cup \mathcal{B}$, for any two faces $f_1, f_2 \in \mathcal{F} - f_\infty$, $\{f_1, f_2\} \in E^*$ iff both faces are adjacent, and for any face $f \in \mathcal{F} - f_\infty$ and any $e \in \mathcal{B}$, $\{f, e\} \in E^*$ iff e separates f and f_∞. Given an embedded graph G, the *face incidence graph* $G^+ = (V^+, E^+)$ is the graph whose vertex set is the set $\mathcal{F}$ of faces of G, and for any two faces $f_1 \neq f_2$, $\{f_1, f_2\} \in E^+$ exactly if f_1 and f_2 have at least one vertex in common. For any face $f \in V^+$ let $d(f)$ be the minimum distance from f to the node representing the exterior face in G^+.

2.2 Levels, k-outerplanarity

First we introduce the notion of level in an embedded graph G. A vertex is a *level 1* vertex if it is on the exterior face. Let G^i be the graph obtained by deleting all vertices in levels 1 to i, then the vertices on the exterior face of G^i are the *level* $i+1$ vertices.

An embedded graph is *k-outerplanar* if it has no vertices of level greater than k (see figure 1). Every embedded graph is k-outerplanar for some k. The terms outerplanar and 1-outerplanar are equivalent.

We show now how to compute the levels of a connected plane graph G in parallel.

The following result can be proved by a straightforward induction argument on $d(f)$,

Lemma 1 *Given an embedded graph G and its face incidence graph G^+, for any face f, $d(f) = k$ iff f contains at least one level k vertex and the remaining vertices in f are level k or $k+1$ vertices.*

As a corollary we get the following way to compute the level of a vertex,

Corollary 1 *Given an embedded graph G, for each vertex v of G, the level of vertex v in G is given by $l(v) = 1 + \min_{v \in f} d(f)$.*

We show now a method to compute the levels of a planar graph in parallel.

Theorem 1 *Given a embedded graph G, the level of each vertex can be computed in parallel time $O(\log^2 n)$ using $O(n^3)$ processors.*

Proof. We first construct the face incidence graph G^+. For each edge in G consider two directed edges (one in each direction). Construct the matrix $M(i,j)$ defined by $M(i,j) = (j,k)$ if k is the vertex next to i in the adjacency list of j. Matrix M can be obtained in constant time using $O(n)$ processors. The associated graph to M is a disjoint union of cycles, each cycle corresponding to a face of G. Using pointer jumping assign a name to each face (for example the largest number of a processor assigned to an edge of the face). To get G^+, we only need to test for each pair of faces whether they share a vertex, which can be done in $O(\log n)$ time with $O(n^2)$ processors.

Using transitive closure compute the distances to the exterior face in $O(\log^2 n)$ time and $O(n^3)$ processors . Finally the minimum distance of the faces containing a given vertex can be computed in $O(\log n)$ time and with $O(n)$ processors. □

In order to simplify notation we will use the term *level i subgraph* to denote a connected component of the subgraph induced by the level i vertices (see figure 2). It follows from the definition of levels that every level subgraph is outerplanar. Furthermore, every level $i+1$ subgraph is in a face of a level i graph. A face in a level i subgraph can have inside more than one level $i+1$ graph. If this is the case, we add dummy edges to G to split the face in such a way that each new face contains exactly one level i graph taking care of preserving the planarity. (In figure 2 we have two level-2 subgraphs.) Therefore from now on, we assume that inside of each face at level i there is at most one level $i+1$ subgraph.

2.3 Outerplanar graph representation

Let us assume we are given a biconnected outerplanar graph G'. The *face-face tree* representation of G' is a rooted ordered tree that has as leaves the edges in the exterior boundary of G', and constructed in such a way that each internal node x in the tree corresponds to an interior face f_x of G'. In fact for every interior node x of the tree, we can associate two vertices of G', $b_1(x)$ and $b_2(x)$ such that if y denotes the father of x in the tree, then $(b_1(x), b_2(x))$ is the interior edge of G' separating f_x from f_y. Moreover we also can identify x with the portion of the graph G'_x, induced by all nodes encountered in a counterclockwise tour on the exterior face of G', starting at $b_1(x)$ and ending at $b_2(x)$. In the case of the root r, we have $b_1(r) = b_2(r)$, so that $G'_r = G'$.

Notice the face-face tree representation is different from the face-edge tree in the sequential algorithm [Bak83], however both representations verify the conditions needed to process the graph, namely that for any node x, a preorder traversal of the subtree rooted at x gaves a counterclockwise traversal of the exterior face G'_x, starting at $b_1(x)$ and ending at $b_2(x)$.

Lemma 2 *Given a biconnected outerplanar graph G' a a face-face tree representation can be obtained in $O(\log n)$ parallel time using $O(n^2)$ processors.*

Proof.

Given G', in parallel construct the quasidual graph G'^* in $O(\log n)$ time and using $O(n)$ processors. Notice that G'^* is a unrooted tree and the leaves are vertices labeled as boundary edges of G'. Choose an interior vertex of G'^* as root r and choose any vertex of the face in G' represented by r as $b(r)$. Use pointer jumping to obtain the rooted face-face tree representation. We associate to each node in the tree (different than r) the edge (in G') shared with the face represented by its father, with the corresponding orientation. The parallel complexity of the tree construction is $O(\log n)$ steps with $O(n^2)$ processors. □

In order to construct the face-face tree of an outerplanar (not necessarily biconnected) graph G, we first transform G into a biconnected outerplanar graph G', an use the tree representation of G' as tree representation for G. The construction of G' from G is done by the following procedure,

1. For each cutpoint p of G whose removal gives $k \geq 2$ components, we add k vertices connected as a cycle. (In the case k=2, the 2 added vertices just form an edge.) Every interior face having p as cutpoint, will be glued to one different edge on the cycle, and every single edge in G having p as endpoint will be transformed into a new edge. (Notice that the point itself expands into a new interior face of G' delimited by the cycle.) The planar embedding gives a cyclic order of the edges from p, giving also a cyclic ordering of the components. The new vertices will follow this order, each vertex will be attached to two consecutive components, and connected to the last vertex in the first component and to all vertices previous to the last (if any) in the next component.

2. After this step every edge from p has a copy as an edge from some of the new nodes in the neighborhood of p. Moreover, each bridge has been converted into four nodes (two from each endpoint), we connect the end points by two parallel edges.

3. Finally we remove all vertices that were cut points in G with the corresponding edges.

(Figure 5 shows two examples of the described transformation, and figure 6 shows two other examples without the drawing for step 2. Figure 3 presents the transformation corresponding to the level 3 subgraph in figure 2). The constructed G' has no cutpoints, so is a biconnected graph. Also all new edges lie on the new exterior face, preserving outerplanarity. To keep information about G in G' we will classify the edges of the constructed G' in

two types; the *virtual* edges that are the edges joining two vertices in G' that correspond to the same vertex in G, and *real* edges that are all the remaining edges in G'

Lemma 3 *Given an outerplanar graph G, the associated biconnected graph G' can be obtained in parallel time $O(\log^2 n)$ and using $O(n^2)$ processors.*

Proof. Obtain the list of cut points and bridges of G. This can be done in parallel time $O(\log^2(n))$ using $O(n^2)$ processors [HCS79]. From the list, the new graph G' can be obtained in parallel constant time using $O(n)$ processors. □

It is easy to verify that the face-face tree for G' also verifies all the required conditions for the outerplanar graph G.

Lemma 4 *Given an outerplanar graph G and the associated biconnected graph G', a tree representation of G can be obtained in parallel time $O(\log n)$ and using $O(n^2)$ processors.*

As a last step, we have to convert the face-face tree representation in a binary tree (this step is necessary to apply in the next section the tree-contraction technique). The conversion can be done in $O(\log n)$ parallel steps using a linear number of processors [ADKP89]. As usual, during the transformation from non-binary to binary, we need to add some extra dummy nodes (without meaning with respect to the topology of G'). All the remaining nodes in the binary tree representation are also nodes in the non-binary tree representation and they are associated with an edge (b_1, b_2) of G'. The non-dummy nodes of the binary tree representation can be further classified in two types. A node x is said to be *virtual* if $(b_1(x), b_2(x))$ is a virtual edge of G' that is a vertex in G, otherwise x is said to be *real*. Note that the root r of the tree will be classified as virtual node (see figure 7, where the black dots represent dummy nodes, the black squares represent the virtual nodes and the remaining are real nodes).

2.4 k-outerplanar graph representation

In this section we extend the previous procedure to obtain in parallel a tree representation of a connected k-outerplanar graph. In this representation we will keep information over each of the level graphs (a face-face tree as presented before) but now the graph associated to an internal node at level

i in the face-face tree will be a subgraph of the graph induced by all nodes in levels 1 to i.

Given an k-outerplanar graph G, we convert it into an associated graph G' with the property that each level subgraph is biconnected. We proceed in parallel for each level subgraph, using the procedure described in the previous section. (Recall that for a given value of i, we may have more than one level-i subgraph, each of those is treated separetely). When we copy part of the edges from a given cutpoint, we also include the edges joining this cutpoint to the previous level. Notice that an edge joining two cutpoints of two different levels has been converted into two edges, each one connecting a cutpoint with a new edge, we connect both new vertices (see figure 8). We finally remove all vertices that were cutpoints in a level subgraph.

To construct the face-face tree for the whole graph, we begin by constructing the face-face tree for each level subgraph. As each level i subgraph is inside a face of a level $i-1$ subgraph, we know from where to hang the tree, however it is necessary to choose appropriately the roots of the different face-face trees, so when the tree for the whole k level graph is constructed, it will have the desired properties. Let us start with a definition. A level vertex v in a subgraph at level i is said to be *consistent* with a vertex u in a subgraph at level $i-1$ (respectively a level $i-1$ edge e) if the two vertices (the vertex and the middle point of the edge) can be joined by a line preserving planarity. A *consistent set of roots* is a selection of pairs face vertex (vertex in face), one for each level subgraph, such that after constructing the corresponding face-face trees, each root of a level ($i > 1$) tree is consistent with the edge (vertex) associated to the enclosing face.

Lemma 5 *A consistent set of roots can be constructed in $O(\log n)$ parallel time using $O(n^2)$ processors.*

Proof. To choose roots of the level trees in parallel we construct an auxiliary graph. This graph has the following set of nodes: all pairs (f, v) with f a face in a level subgraph of G' and such that v belongs to f, together with all pairs (f, e) where f is a face in a level subgraph and e is an interior edge of the subgraph such that e belongs to f. The set of edges are the pairs:

> $((f, v), (f', e))$ where f and f' are different faces in the same level i subgraph with e being the closing edge of f' in a counterclokwise traversal starting at v (notice that when f and f' are neighbors, then e will be the boundary between both faces).

$((f,e),(f',v))$ where f' and v are a level $i+1$ face and vertex, f is the corresponding enclosing face in the level i subgraph, e is an edge in f' and v and e are consistent.

$((f,v),(f',v'))$ where f' and v' are a level $i+1$ face and vertex, f is the corresponding enclosing face in the level i subgraph, v is an edge in f' and v and v' are consistent.

Notice the whole auxiliary graph will look as $2k-1$ bipartite graphs, starting from the bipartite graph between (f,v) and (f',e) at level one and alternates in bipartite layers representing correct possible connections between levels and correct face-face subtree for a same level subgraph, and correct connections between face-face subtrees for different levels. This graph can be constructed in $O(\log n)$ parallel time using $O(n^2)$ processors.

From the previous construction, to select in parallel a consistent set of roots for the level trees, for each node (f,e), corresponding to a level i face, we choose one of the existing edges (if any) joining this node with a level $i+1$ vertex. By choosing only a level 1 vertex face pair, we get a tree, the set of all pairs vertex face in this tree form a consistent set of roots. □

Once we have a consistent set of roots, we construct the face-face trees. We want to relate vertices in a given face-face tree of a subgraph level, to vertices in the enclosing face in such a way that when the resulting tree has the correct properties.

For a given face-face level ($i > 1$) tree T, let $a_1,\ldots,a_s$ be the exterior cycle of edges at level i going from the first endpoint to the last (that is the exterior boundary of the corresponding subgraph), and $d_1,\ldots,d_q$ be the path (cycle) of level $i-1$ edges in the enclosing face. To each vertex v we associate a vertex in the enclosing face $d(v)$ as follows. Let v_0 be the first vertex of a_1, v_i be the common vertex to a_{i-1} and a_i, and v_{s+1} the last vertex of a_s. (Notice that v_0 and v_{s+1} are in fact the same vertex but are considered different as first and last vertex on the tour). $d(v_0)$ is the first vertex of d_1 and $d(v_{s+1})$ is the last vertex of d_q. For other i we consider two cases, v_i has no edges to vertices in the enclosing face, if this is the case $d(v_i) = d(v_{i-1})$, otherwise $d(v_i) = w$ where w is the last visited level $i-1$ vertex to which v_i is connected.

To associate a portion of the graph G to each node x in a level tree, we consider two ordered set of vertices $B_1(x)$ and $B_2(x)$. Let x be a a node in the tree representing a level i subgraph with $u = b_1(x)$ and $v = b_2(x)$ then $B_1(x) = \langle u, d(u),\ldots,d^{i-1}(u)\rangle$ and $B_2(x) = \langle v, d(v),\ldots,d^{i-1}(v)\rangle$.

The associated graph G'_x will be the subgraph containing the vertices in $B_1(x)$ and $B_2(x)$, all vertices encountered in a counterclockwise traversal of level subgraphs starting at $B_1(x)$ and ending at $B_2(x)$, all edges with both endpoints in G'_x are included, except any that leaves $B_1(x)$ in a clockwise direction.

To obtain the final tree we have to connect the trees for the different level subgraphs. We construct the tree as follows:

Suppose that x is the node corresponding to a face that have inside a level subgraph having face-face tree T. We connect tree T as the only son of x. Let y be a leaf of T, $u = b_1(y)$ and $v = b_2(y)$. We consider two cases

Case 1 $d(u) \neq d(v)$ let z be the first son of x such that $b_1(z) = d(u)$, z' be the last son of x such that $b_2(z') = d(v)$, and z'' be the first son of x such that $b_2(z'')$ is connected to v. We add a new nodes labeled u as first son of y, the sons of u will be all sons of x from z to z''. The second son of y when $z' \neq z''$ is a node labeled v that has as sons the next sons of x before z', and when $z' = z$ a node labeled $B_2(y)$ that has no son.

(Note, when we finally convert the tree in a binary-unary one, a node labeled u will expand into a set of dummy nodes, all of them will keep the same label u)

Case 2 $d(u) = d(v)$, we add two new nodes labeled $B_1(y)$, $B_2(y)$ as first and second son of y.

Again this construction can be done in $O(\log n)$ parallel time using $O(n^2)$ processors. Thus we get,

Lemma 6 *Given a k-outerplanar graph, we can compute in parallel a face-face tree representation in $O(\log^2 n)$ time and using $O(n^2)$ processors.*

Note that in the previous result, the bounds on the number of processors and on the number of steps, are independent of k.

3 Approximating MIS

In this section we describe the procedure to obtain an approximation to the MIS Problem on planar graphs. The description is divided into three parts. First we describe an exact algorithm to compute a MIS for outerplanar graphs. After we extend the technique to obtain an exact parallel algorithm to compute a MIS for a k-outerplanar graph. Finally we show how to use the exact algorithm for k-outerplanar graphs to obtain an approximate MIS.

The algorithm we will describe computes the size of the IS, however a slight modification will allow us to compute an approximate IS.

3.1 Outerplanar graphs

In this section, given a binary face-face tree, we compute the MIS of the corresponding outerplanar graph. Recall that by construction, to each node x in the tree we have associated two vertices $b_1(x), b_2(x)$, and a subgraph G_x of the graph G. Moreover we associate to each node x in the tree, a table t_x with four entries; $t_x(0,0), t_x(0,1), t_x(1,0)$ and $t_x(1,1)$. Each entry contains the maximum size of an independent set in G_x depending on which of the associated vertices are or are not forced to be in the independent set. Thus entry $t_x(0,0)$ contains the maximum size of an IS not containing $b_1(x)$ and $b_2(x)$, entry $t_x(0,1)$ contains the maximum size of an IS not containing $b_1(x)$ but contains $b_2(x)$, entry $t_x(1,0)$ contains the maximum size of an IS not containing $b_2(x)$ but containing $b_1(x)$ and finally entry $t_x(1,1)$ contains the maximum size of an IS containing both $b_1(x)$ and $b_2(x)$.

For a real leaf x the corresponding table is easy to compute, note that G_x is just an edge, thus we let $t_x(0,0) = 0$, $t_x(0,1) = t_x(1,0) = 1$ and $t_x(1,1)$ is undefined (there is no IS containing both end points). If the leaf is virtual, the associated table is defined $t_x(0,0) = 0$, $t_x(1,1) = 1$, $t_x(0,1)$ and $t_x(1,0)$ are undefined.

To compute the MIS for the whole outerplanar graph, we transvers the tree in a bottom-up way, computing at each interior node of the tree an operation *merge* of tables, as we have three types of internal nodes we need three types of merging. The basic merge of two tables t_x and t_y is defined as follows; for any $a, b \in \{0,1\}$

$$t_{xy}(a,b) = \max_{c \in \{0,1\}} \{t_x(a,c) + t_y(c,b) - c\}.$$

Suppose that z is an internal node with left child x and right child y the table for z will be $t_z = t_{xy}$ when z is a marked node, $t_z = t_{xy}$ except for the entry (1,1) that will be undefined when z is real, and $t_z(0,0) = t_{xy}(0,0)$, $t_z(1,1) = t_{xy}(1,1) - 1$, and $t_z(1,0)$ and $t_z(0,1)$ will be both undefined, when z is virtual.

An easy induction proof shows that the table obtained for each internal node, corresponds to the maximum size of an IS set for the associated subgraph. Once we have a table for the root, the two entries in this table (recall that the root is a virtual node) give us the maximum size of an IS

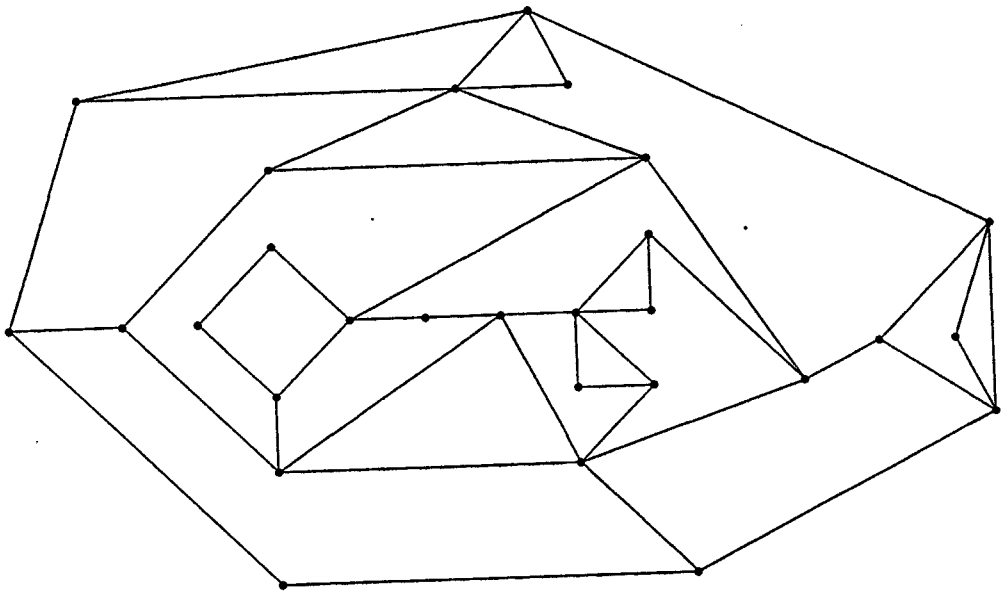

Figure 1: A 3-outerplanar graph G

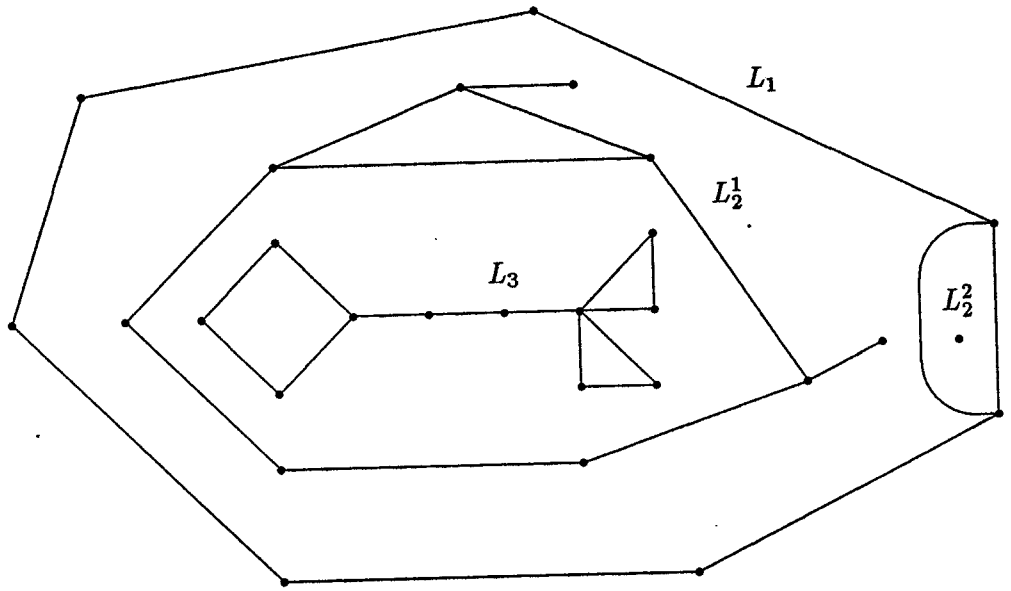

Figure 2: The level subgraphs of graph G

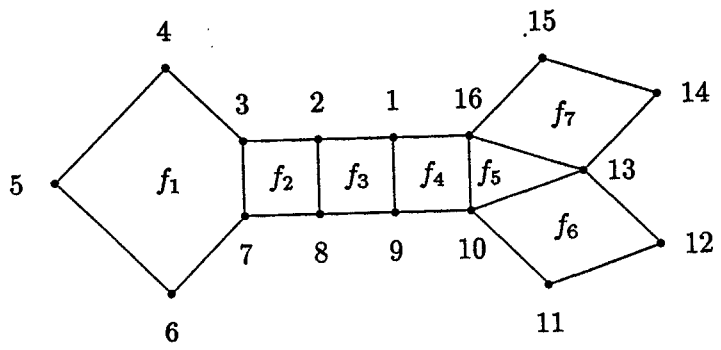

Figure 3: A bi-connected outerplanar graph L

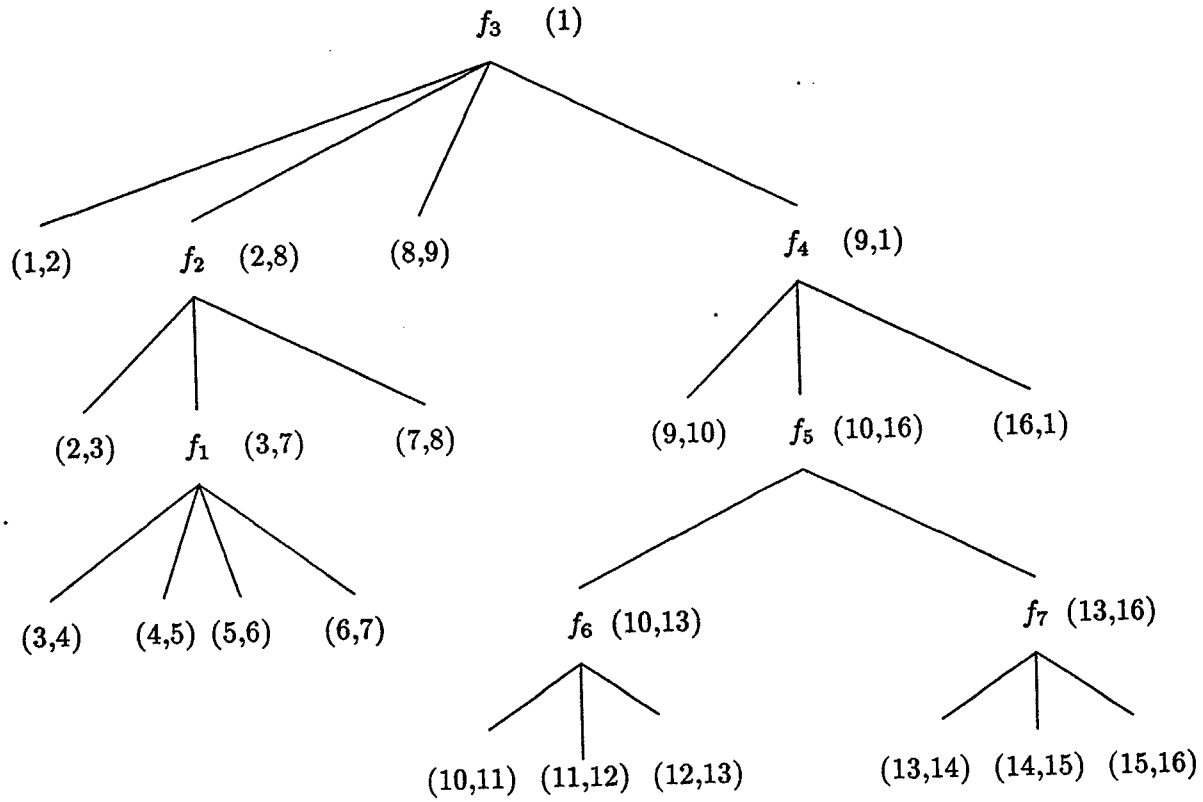

Figure 4: The face-face tree of L

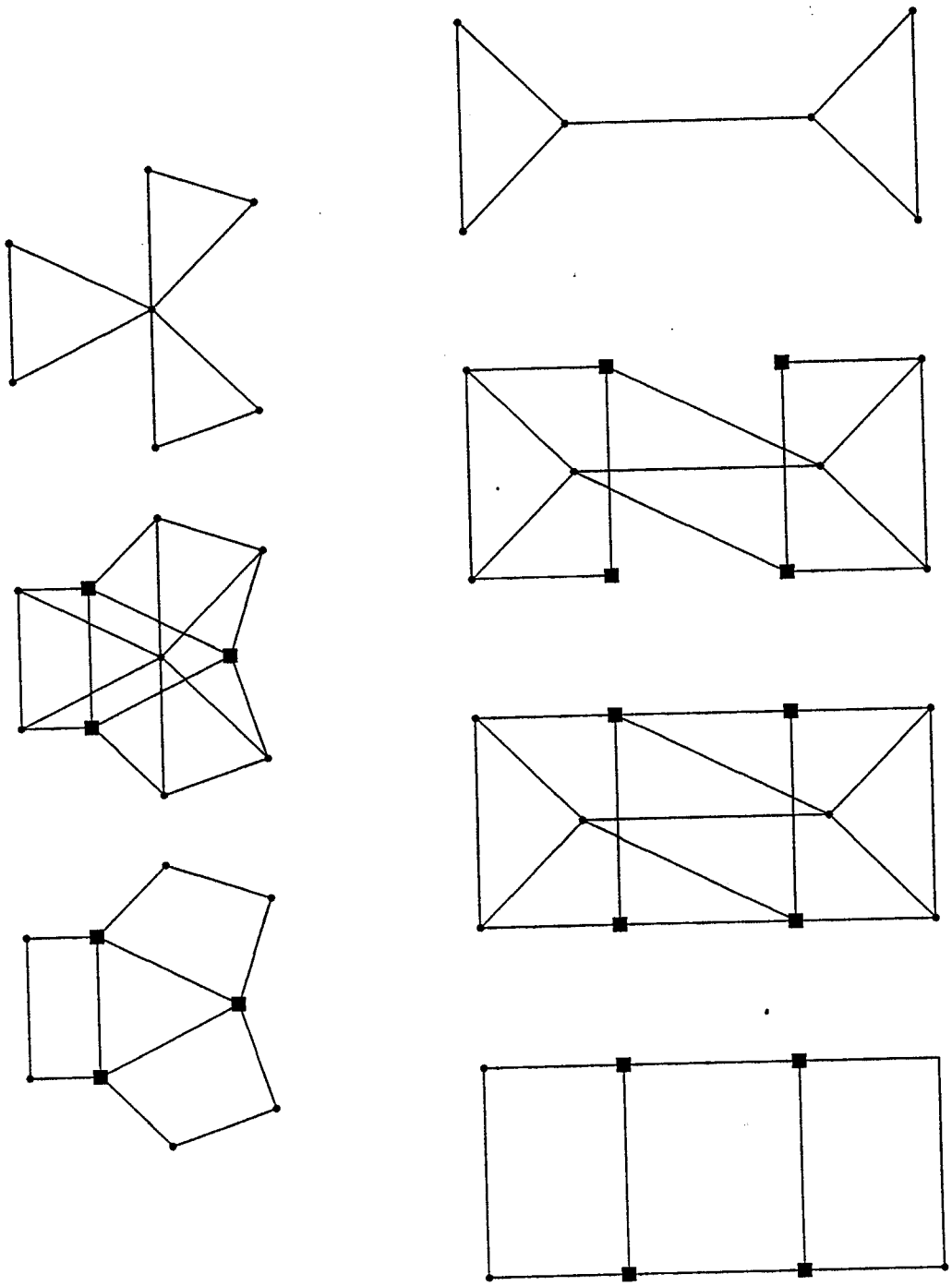

Figure 5: Transformation of a cutpoint with three components and a bridge

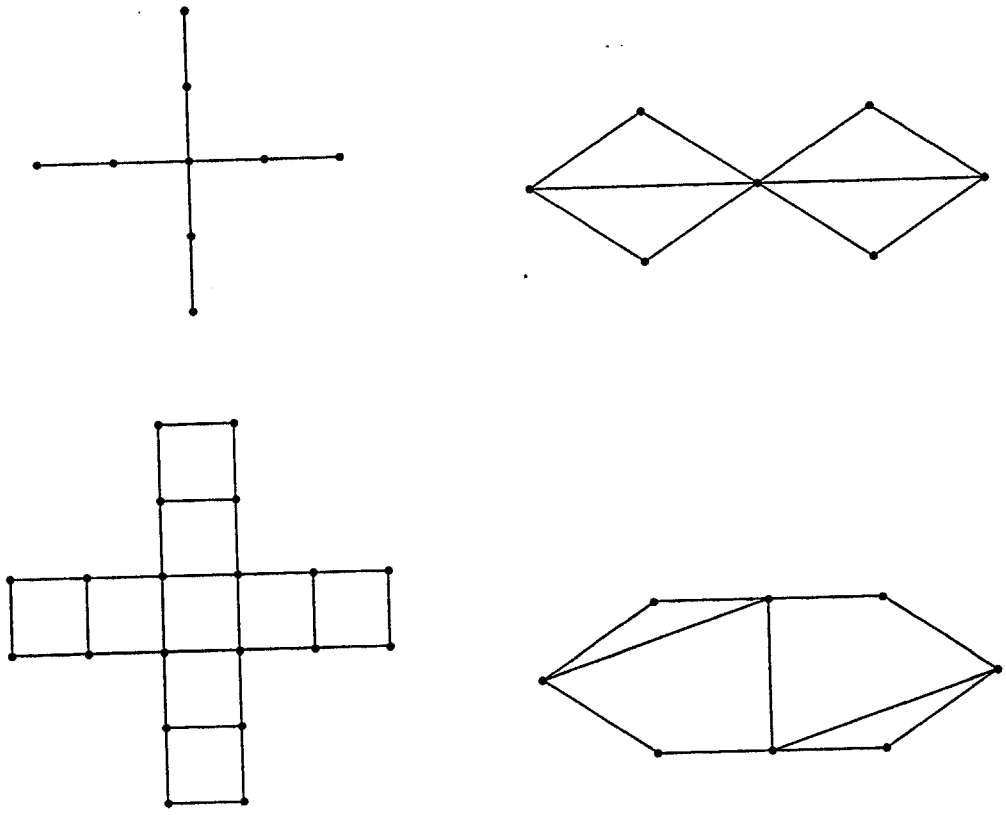

Figure 6: The associated graph for some outerplanar graphs

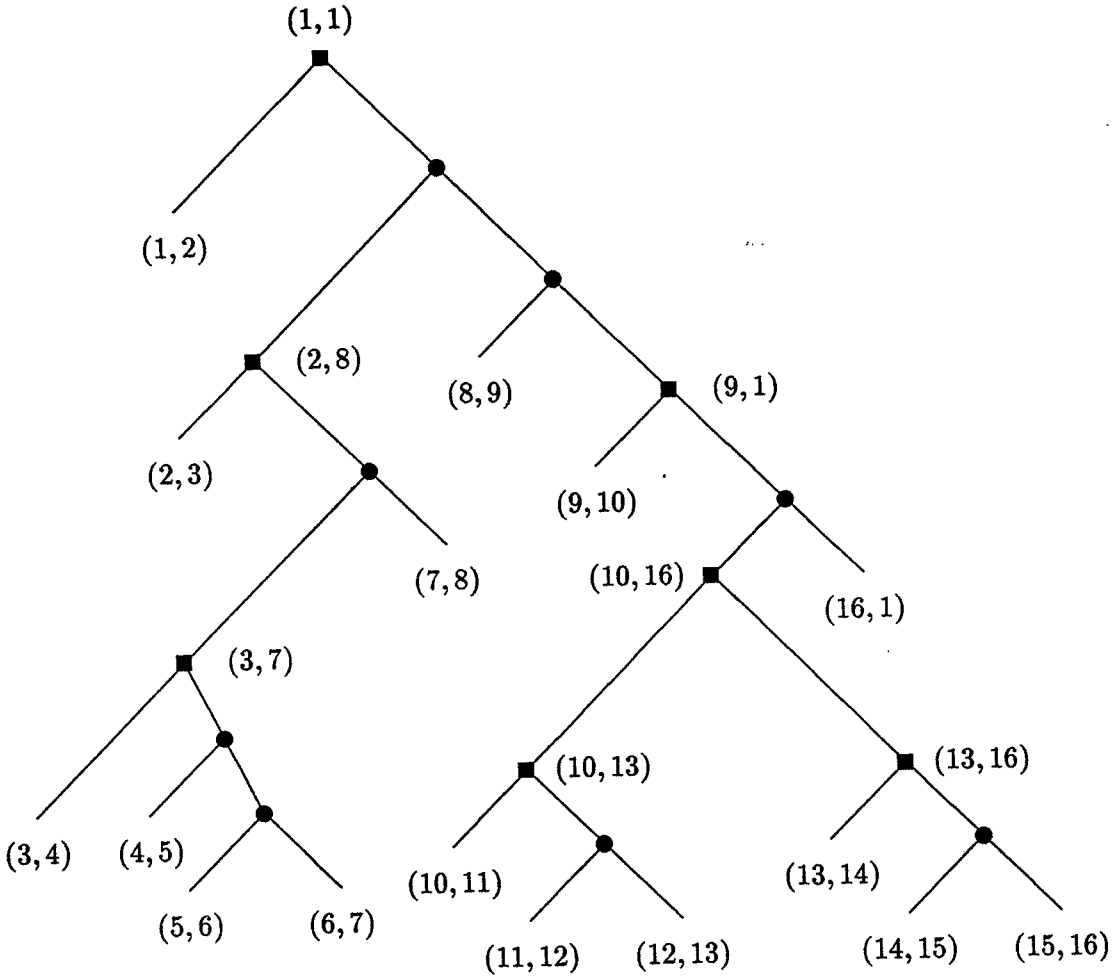

Figure 7: The binary face-face tree of L_3. Black dots represent dummy nodes, black squares represent virtual nodes, the remaining are real.

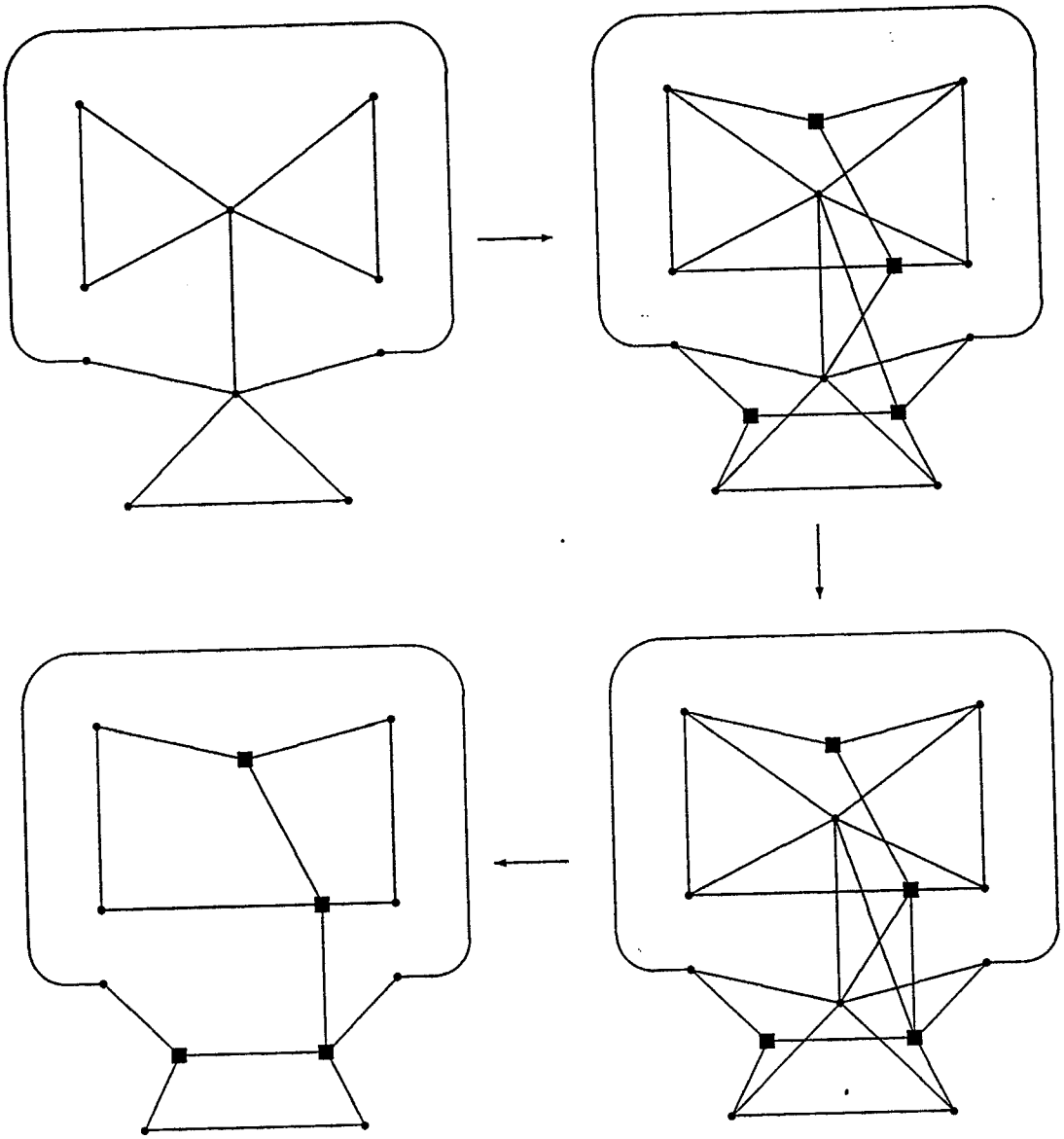

Figure 8: The transformation of an edge joining two cutpoints in different levels

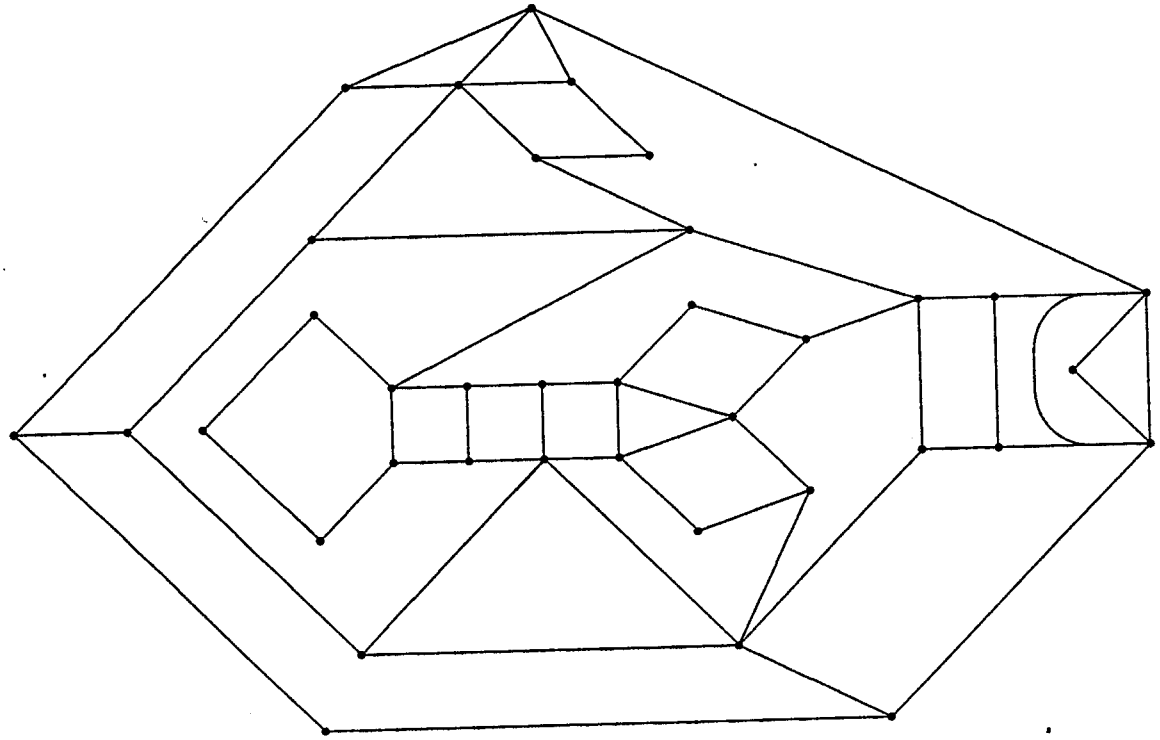

Figure 9: The associated graph G'

Departament de Llengatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

List of research reports (1993).

LSI–93–1–R "A methodology for semantically enriching interoperable databases", Malú Castellanos.

LSI–93–2–R "Extraction of data dependencies", Malú Castellanos and Fèlix Saltor.

LSI–93–3–R "The use of visibility coherence for radiosity computation", X. Pueyo.

LSI–93–4–R "An integral geometry based method for fast form-factor computation", Mateu Sbert.

LSI–93–5–R "Temporal coherence in progressive radiosity", D. Tost and X. Pueyo.

LSI–93–6–R "Multilevel use of coherence for complex radiosity environments", Josep Vilaplana and Xavier Pueyo.

LSI–93–7–R "A characterization of $PF^{NP\|} = PF^{NP[log]}$", Antoni Lozano.

LSI–93–8–R "Computing functions with parallel queries to NP", Birgit Jenner and Jacobo Torán.

LSI–93–9–R "Simple LPO-constraint solving methods", Robert Nieuwenhuis.

LSI–93–10–R "Parallel approximation schemes for problems on planar graphs", Josep Díaz, María J. Serna, and Jacobo Torán.

Internal reports can be ordered from:

Nuria Sánchez
Departament de Llenguatges i Sistemes Informàtics (U.P.C.)
Pau Gargallo 5
08028 Barcelona, Spain
secrelsi@lsi.upc.es

for G including or not the associated vertex, thus taking the maximum of both entries we get the size of an MIS for the graph G.

We define now the shunt operation of the tree contraction technique [ADKP89, KR90]. Suppose that x is the leaf which is ready to perform the shunt, that z is its father and that y is the other child of z. We will contract the three nodes into a single one, note that the piece of the graph represented after the contraction, for an internal node, will have three contact vertices, so that we have to keep a three entries table in the internal contracted nodes. To define the new operation we proceed by cases. In order to simplify the definitions we assume that x is the left child of z, and that when we have a three entries table, the first an the second entry correspond to the vertices shared with the left child, the second and the third with the right child, and the first and the last with the father. Furthermore the term "incorporate the type of node x" to a table has the following meaning: if x is real the values corresponding to (1,1) will be undefined; if x is virtual all values corresponding to (1,0) and (0,1) will be undefined, and we subtract 1 to all values corresponding to (1,1); otherwise the table is not modified. We distinguish among two cases

Case 1. z is a node that has not been processed by the shunt operation, thus z contains only information about the type of an edge.

Case 1.1. y is a leaf, using the corresponding merging we get a two entries table for z, note that now z becomes a leaf.

Case 1.2. y is not a leaf and has not been processed by the shunt operation. In this case we construct a three entries table as follows

$$t_{xyz}(a,b,c) = t_x(a,b) + c$$

for $a, b, c \in \{0,1\}$. After, we incorporate the type of nodes y and z.

Case 1.3. y is not a leaf but a node obtained after a shunt. In this case y has a three entries table, and we construct a three entries table as follows,

$$t_{xyz}(a,b,c) = \max_{d\in\{0,1\}} \{t_x(a,d) + t_y(d,b,c) - d\}$$

for $a, b, c \in \{0,1\}$. After, we incorporate the type of node z.

Case 2. z is a node obtained after a shunt, now z has a table with three entries.

Case 2.1. y is a leaf, and thus have a two entries table, note that the resulting graph have only two associated vertices, again the new node will be a leaf. We construct a table as follows,

$$t_{xyz}(a,b) = \max_{c\in\{0,1\}}\{t_x(a,c) + t_z(a,c,b) + t_y(c,b) - c\} - a - b$$

for $a, b \in \{0,1\}$.

Case 2.2. y is not a leaf and not a node processed by the shunt operation, in this case we construct a new table as follows,

$$t_{xyz}(a,b,c) = t_x(a,b) + t_z(a,b,c) - a - b$$

for $a, b, c \in \{0,1\}$. After that we incorporate the type of node y.

Case 2.3. y is not a leaf but a node obtained after a shunt. In this case y has a three entry table. The new table is computed as

$$t_{xyz}(a,b,c) = \max_{d\in\{0,1\}}\{t_x(a,d) + t_z(a,d,b) + t_y(d,c,b) - 2d\} - a - b$$

for $a, b, c \in \{0,1\}$.

It is easy to show that in each case the values contained in each entry correspond to maximum sizes of IS for the corresponding subgraph, taking into account the associated vertices forced to be in the IS.

Note that each one of the operations can be computed in constant time using a constant number of processors, thus using the tree contraction technique we can compute the final MIS in $O(\log n)$ time using $O(n)$ processors, however the resources needed are dominated by the step in which we construct the associated bi-connected outerplanar graph, thus we have,

Theorem 2 *There is a parallel algorithm to compute a maximum independent set for an outerplanar graph that runs in $O(\log^2 n)$ time using $O(n^2)$ processors.*

3.2 k-outerplanar graphs

Finally, let us compute the MIS of a given k-outerplanar graph. We consider the face-face tree described in section 2.4. Recall that by construction, to each node x in a face-face tree we associated two ordered sets of vertices, $B_1(x), B_2(x)$, and a subgraph G_x of the k-outerplanar graph G.

For each node x in the tree with i associated vertices, we compute a table t_x with 2^{i+1} entries $t_x(a,b)$ with $a,b \in \{0,1\}^i$, each entry contains the maximim size of an IS in G_x depending on which of the vertices in $B_1(x)$, or $B_2(x)$ are forced to be or not to be in the IS.

In order to compute tables for the nodes of the face-face tree, we consider three operations on tables; *table extension*, *table contraction* and *table merging*.

Given a table for a leaf x on level i, and given a level $i+1$ node w, we will extend t_x to obtain a table t_{x+w} for a node with associated set of vertices $\langle w, B_1(x)\rangle$, $\langle w, B_2(x)\rangle$. For any of $(a,b) \in \{0,1\}^i \times \{0,1\}^i$ representing whether each of the vertices in $B_1(x), B_2(x)$ are forced to be in the IS, the value t_{x+w} will be:

$$
\begin{aligned}
t_{x+w}(0a,0b) &= t_x(a,b) \\
t_{x+w}(1a,0b) &= t_{x+w}(0a,1b) = \text{ undefined} \\
t_{x+w}(1a,1b) &= \begin{cases} \text{undefined} & \text{if } w \text{ and a level } i \text{ vertex present} \\ & \text{in the IS are connected} \\ t_x(a,b)+1 & \text{otherwise} \end{cases}
\end{aligned}
$$

In the reverse sense we define a table contraction operation. Given a table for the root of a level i tree, recall that $B_1(x) = \langle w, B_1'(x)\rangle$ and $B_2(x) = \langle w, B_2'(x)\rangle$, we will contract t_x to obtain a table t_{x-w} for a node with associated set of vertices $B_1'(x)$, $B_2'(x)$.

$$t_{x-w}(a,b) = \max\{t_x(0a,0b), t_x(1a,1b)\}$$

We also extend the *merging* operation described in 3.1 to deal with sets of more than one vertex, for any $a,b \in \{0,1\}^i$ $(i \leq k)$, let $|c|$ denote the number of 1's in c,

$$t_{xy}(a,b) = \max_{c\in\{0,1\}^i}\{t_x(a,c) + t_y(c,b) - |c|\}.$$

Let's analyze the nodes on the face-face tree corresponding to a k-outerplanar graph. We have two kinds of leaves, all leaves in the level 1 tree are leaves in the tree, the initial table for these leaves will be just the initial tables in the outerplanar case. Other leaves are nodes x labeled by a set of vertices $B(x)$, the initial table is defined as follows; $t_x(a,b)$ will be undefined when $a \neq b$ or when both values are the same but there is an edge between two vertices that have to be included in the IS, otherwise the value will be the number of 1's in a.

The operation assigned to a non leaf node x is the following: if x is a node in a level tree that corresponds to a face enclosing a level subgraph, in this case node x has only a unique child and we contract the table for its son. If x is a node labeled u (or v), we extend the table of the left son with u (v) and merge with the table for the right son using the merge operation corresponding to a virtual node. In the rest of the nodes a merging will be performed according to the type of the node.

Note that now the value in each entry of the new table is the size of a MIS including the corresponding vertices, provided that the original tables were correct. Working in a similar way as in section 3.1 we can define a shunt operation to apply the tree contraction technique.

As the merging of tables now needs $O(k)$ time and $O(8^k)$ processors, a shunt operation can be done in the same time bounds, but using $O(16^k)$ processors thus final computation over the tree needs $O(k \log n)$ time and $O(16^k n)$ processors. Putting together all the bounds we get

Theorem 3 *There is a parallel algorithm to compute a maximum independent set for a k-outerplanar graph that runs in* $O(\log n(k+\log n))$ *time using* $O(n(16^k + n))$ *processors.*

3.3 Planar graphs

The decomposition of a planar graph into k-outerplanar graphs is the same used in [Bak83]. For each i, $0 \leq i \leq k$. Let G_i be the graph obtained by deleting all nodes of G whose levels are congruent to $i \pmod{k+1}$. Now every connected component of G_i is k-outerplanar.

An independent set for G_i can be computed as the union of independent sets for each component. Furthermore, for some r, $0 \leq r \leq k$ the solution for G_r is at least $k/(k+1)$ as large as the optimal solution for G, that follows from the fact that for some r, at most $1/(k+1)$ of the nodes in a maximum independent set for G are at a level that is congruent to $r \pmod{k+1}$. Thus

the largest of the solutions for the G_i's is an independent set whose size is at least $k/(k+1)$ optimal.

Algorithm IS planar

1. Compute levels, the associated graph G', a tree representation.
2. Get k-copies of the tree.
3. For all $0 \leq i \leq k$ in parallel remove the corresponding parts of the tree to get a representation of the connected components of G_i.
4. For all connected component in parallel

   Run the k-outerplanar algorithm.
5. For each $0 \leq i \leq k$ add up the sizes of the corresponding components.
6. Compute the maximum over all i, $0 \leq i \leq k$

Putting together all the complexity bounds we get

Theorem 4 *Given k there is a parallel approximation for maximum independent set that runs in $O(\log n(k + \log n))$ time, uses $O(n^2(16^k + n))$ processors and achieves a solution at least $k/(k+1)$ optimal.*

Thus we get an NC^2 parallel approximation scheme by letting $\epsilon = 1/k$ and an NC^2 asymptotic approximation scheme taking $k = c \log\log n$ for some constant c.

Theorem 5 *The Maximum Independent Set Problem for planar graphs is in $NCAS$ and $NCAS^\infty$*

4 Conclusions and algorithms for other problems

We have shown that for any k there is parallel algorithm that approximate the Maximum Independent Set problem on planar graphs to a factor $\frac{k-1}{k}$, and runs in time $O(\log n(k + \log n))$ using $O(n^2(16^k + n))$ processors. From this follows that the MIS problem for planar graphs is in NCAS and in $NCAS^\infty$.

The techniques to obtain a parallel approximation algorithm for the MIS problem explained in the previous sections can be transformed to obtain

approximations for other optimization problems on planar graphs, as well as exact NC algorithms for these problems when restricted to k-outerplanar graphs. In particular, by a simple adaptation of the explained techniques we are able to show that the following problems restricted to planar graphs are in NCAS and NCAS$^\infty$: Minimum Vertex Cover, Minimum Dominating Set, Minimum Edge Dominating Set, Maximum Cut, Maximum Even Degree Set [SWK90] and Maximum Matching. A consequence of the proofs of these results is that there are exact polylogarithmic parallel algorithms for these problems restricted to k outerplanar graphs. Adapting the same technique it can be shown, the 3-colorability and Graph Bisection [BP92] problems on k outerplanar graphs belong also to the class NC. For space reasons we just sketch the proof for the Vertex Cover Problem. This gives an idea about the changes that have to be made in the algorithms from Section 3 to compute the other mentioned problems. Moreover, an interesting open problem is lower the complexity bounds in the number of processors. It will be nice to obtain an optimal parallel approximation scheme, that is a work complexity of $O(n)$ for constant k.

a/ Vertex Cover in outerplanar graphs:

Given a face-face tree for the graph we associate to each node x in the tree a table t_x with four entries following the same ideas as in Section 3.1. For the Vertex Cover Problem the table t_x is defined in the following way:

If x is a real leaf, then we let $t_x(0,0)$ undefined, $t_x(0,1) = t_x(1,0) = 1$, and $t_x(1,1) = 2$. If x is a virtual leaf then t_x is undefined for all entries except for $(1,1)$ and the value in this case is 1.

If x is an internal node with left and right children y and z, we distinguish three cases depending on whether x is marked, real or virtual. If x is a marked node then for any $a, b \in \{0,1\}$, $t_z(a,b)$ is defined as

$$t_x(a,b) = t_{y,z}(a,b) = \min_{c\in\{0,1\}}\{t_y(a,c) + t_z(c,b) - c\}.$$

If x is real then $t_x(a,b) = t_{y,z}(a,b)$ except in the case $(a,b) = (0,0)$ in which t_x is undefined. In case x is virtual then $t_x(a,b)$ is only defined for the case $(a,b) = (1,1)$ and $t_x(1,1) = t_{y,z}(1,1)$.

b/ Vertex Cover in k-outerplanar graphs:

As in Section 3.2 to compute the tables for the nodes in the face-face tree we consider three operations on tables; table extension, table contraction and table merging. In the case of the Vertex Cover Problem the extension

is done as follows: Given a table for a leaf x on level i and a level $i+1$ node w, the table for x is extended to t_{x+w} as

$$\begin{aligned} t_{x+w}(0a,0b) &= \begin{cases} \text{undefined} & \text{if } w \text{ and a level } i \text{ vertex not present in the VC are connected} \\ t_x(a,b) & \text{otherwise} \end{cases} \\ t_{x+w}(1a,0b) &= t_{x+w}(0a,1b) = \text{undefined} \\ t_{x+w}(1a,1b) &= t_x(a,b)+1 \end{aligned}$$

the table contraction operation is defined as

$$t_{x-w}(a,b) = \min\{t_x(0a,0b), t_x(1a,1b)\}$$

and the merging operation is

$$t_{xy}(a,b) = \min_{c\in\{0,1\}^j}\{t_x(a,c)+t_y(c,b)-|c|\}.$$

for $a,b \in \{0,1\}^i$ $(i \le k)$, ($|c|$ denotes the number of 1's in c).

Acknowledgment
The authors would like to thank Torben Hagerup for many helpful comments.

References

[ADKP89] K. Abrahamson, N. Dadoun, D. Kirkpatrick, and K. Przytycka. A simple parallel tree contraction algorithm. *Journal of Algorithms*, 10:287–302, 1989.

[AM86] R.J. Anderson and E.W. Mayr. Approximating P-complete problems. Technical report, Stanford University, 1986.

[Bak83] B.S. Baker. Approximation algorithms for NP-complete problems on planar graphs. In *24 IEEE Symposium on Foundations of Computer Science*, pages 265–273, 1983.

[BP92] T.N. Bui and A. Peck. Partitioning planar graphs. *SIAM Journal on Computing*, 21:203–215, 1992.

[CN89] M. Chrobak and J. Naor. An efficient parallel algorithm for computing large independent set in a planar graph. In *SPAA-89*, pages 379–387. Association for Computing Machinery, 1989.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

[Hag90] T. Hagerup. Planar depth-first search in log n parallel time. *SIAM Journal on Computing*, 19:678–703, 1990.

[HCS79] D.S. Hirschberg, A.K. Chandra, and D.V. Sarwate. Computing connected components on parallel computers. *CACM*, 22:461–464, 1979.

[JJ92] J. JaJa. *An introduction to parallel algorithms.* Addison-Wesley, New York, 1992.

[Kan92] V. Kann. *On the approximability of NP-complete optimization problems.* PhD thesis, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, Sweden, 1992.

[KR86] P.H. Klein and J.H. Reif. An efficient parallel algorithm for planarity. In *IEEE Symposium on Foundations of Computer Science*, pages 465–477. IEEE Society, 1986.

[KR90] R. Karp and V. Ramachandran. Parallel algorithms for shared memory machines. In Jan van Leewen, editor, *Handbook of Theoretical Computer Science, Vol. A*, pages 869–942. Elsevier Science Publishers, 1990.

[KS93] P. Klein and C. Stein. A parallel algorithm for approximating the minimum cycle cover. *Algorithmica*, 9:23–31, 1993.

[KSS89] L.M. Kirousis, M.J. Serna, and P. Spirakis. The parallel complexity of the connected subgraph problem. In *30th. IEEE Symposium on Foundations of Computer Science*, pages 446–456. IEEE, 1989.

[KX92] G. Kant and He Xin. Two algorithms for finding rectangular duals of planar graphs. Technical Report RUU-CS-92-41, Utrecht University, December 1992.

[LN92] M. Luby and N. Nisan. A parallel approximation algorithm for positive linear programming. Technical report, International Computer Science Institute, Berkeley, 1992.

[PR87] J.G. Peters and L. Rudolph. Parallel approximation schemes for subset sum and knapsack problems. *Acta Informatica*, 24:417–432, 1987.

[Ser90] M.J. Serna. *The parallel approximability of P-complete problems.* PhD thesis, Universitat Politecnica de Catalunya, 1990.

[Ser91] M.J. Serna. Approximating linear programming is log-space complete for P. *Information Processing Letters*, 37:233–236, 1991.

[SS89] M.J. Serna and P. Spirakis. The approximability of problems complete for P. In *Proceedings in Optimal Algorithms*, pages 193–204. Lecture Notes in Computer Science, Springer Verlag, 1989.

[SWK90] W-K. Shih, S. Wu, and Y.S. Kuo. Unifying maximum cut and minimum cut of planar graphs. *IEEE Transactions on computers*, 39(5):694–697, May 1990.