Optimal Parallel Algorithms for Sparse Graphs*

Grammati E. Pantziou [1] *Paul G. Spirakis* [1,2] *Christos D. Zaroliagis* [1]

(1) Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece
Computer Sc and Eng Dept, University of Patras, Greece

(2) Courant Institute of Math. Sciences, NYU, USA

Abstract

We present here techniques which exhibit optimal processor-time tradeoff for many important problems on sparse graphs. These problems include: maximal coloring and maximal independent set in trees and bounded degree graphs; 7-colorability, maximal independent set and maximal matching in planar graphs; maximum independent set, maximum matching and Hamiltonian path on rectangular grid graphs. Our techniques are based on the *general list ranking* problem: given k lists having a total of n elements, find for each element the membership relation and the rank of the element in its list. We solve this problem in $O(\log n)$ time with $n/\log n$ processors on an EREW PRAM. For trees and bounded degree graphs our methods need $O(\log n)$ time and $n/\log n$ processors on an EREW PRAM. For planar graphs they need $O(\log^2 n)$ time and $n/\log^2 n$ processors on an EREW PRAM using linear space. For the case of rectangular grid graphs they need $O(\log n)$ time with $n/\log n$ processors on a CRCW PRAM, or on an EREW PRAM (if the embedding is given).

1 Introduction

In graph theory, dealing with problems on sparse graphs is of particular importance. The reason is that sparse graphs have nice structural properties that help in the design of more efficient algorithms for a lot of problems. Furthermore, the solution of a problem on sparse graphs is often used as a subroutine in many algorithms, thus improving their efficiency. Some classes of sparse graphs of particular interest are: trees, bounded-degree graphs, planar graphs and rectangular grid graphs.

In this paper we deal with graph problems concerning the above classes. More precisely, we examine the parallel complexity of the following problems.

- For trees: find a maximal independent set (i.e. 2-colorability), a maximal dividing set and a maximal coloring.
- For bounded-degree graphs, with maximum degree Δ: find a $(\Delta+1)$-coloring, and a maximal independent set.
- For planar graphs: find a 7-coloring, a maximal independent set and a maximal matching.

*This work was partially supported by the EEC ESPRIT Basic Research Action No. 3075 (ALCOM) and by the Ministry of Industry, Energy and Technology of Greece.

- For rectangular grid graphs: find a maximum independent set, a maximum matching and a Hamiltonian path.

The heart of our approach is (a) the Euler tour technique on trees [11] and especially its optimal parallel version [10] (using the optimal list ranking algorithm of [1]), and (b) an optimal solution, that we give, to a fundamental problem in parallel computation. We call it the *general list ranking* (GLR) problem: given k linked lists $l_1, l_2, ..., l_k$, such that $\sum_{i=1}^{k} |l_i| = n$, find for each element the membership relation (i.e. to which list it belongs) as well as the rank of the element in its list. We assume that the input is given in an array $L[1:n]$, and $L(i)$ (for some element i) is the pointer to the next element in the list of i. We also assume that the heads of the lists constitute a separate linked list stored in an array $H[1:k]$. (This is a reasonable hypothesis, if someone assumes e.g. the adjacency list representation of a graph.)

We provide here optimal parallel algorithms for all the above mentioned problems on the CRCW or EREW PRAM model of computation. More precisely, the following results have been achieved in the EREW PRAM model: (i) $O(\log n)$-time, $n/\log n$-processor algorithms for the GLR problem, and for the problems on trees and bounded-degree graphs mentioned above. (ii) $O(\log^2 n)$-time, $n/\log^2 n$-processor algorithms for the problems on planar graphs. The GLR algorithm has an immediate (and very useful) application to the *prefix computation problem on graphs*: given a graph G with n vertices and m edges in its adjacency list representation, perform any parallel prefix computation on the elements of the adjacency lists of G optimally in $O(\log n)$ time. The optimal solution of the above prefix computation problem leads to optimal parallel algorithms on planar graphs. The new result derived is an optimal parallel algorithm for the maximal matching problem. We believe that the GLR algorithm has many applications to a variety of NC algorithms towards optimal speedup.

In the CRCW PRAM model we give $O(\log n)$-time, $n/\log n$-processor algorithms for the above problems on rectangular grid graphs. We must note here that the results on grid graphs depend only on the complexity of finding the embedding. If the embedding is also provided by the input, then our algorithms are optimal in the same resource bounds on an EREW PRAM.

Our results for trees and bounded-degree graphs improve all previous ones for the same problems. The 2-colorability algorithm has a twofold purpose: First, it answers the open problem posed by [2], by giving an optimal solution; and second, it solves the problem in a way that is especially noteworthy for its simplicity ([5] found an algorithm for the above problem in the same time and processor bounds but on a CREW PRAM). Furthermore, the 2-colorability algorithm can be extended to maximally color any tree (or forest) within the same resource bounds. The best known algorithms for the above mentioned problems in bounded-degree graphs have $O(\log^* n)$ running time using a linear number of processors [2]. For the MIS problem in bounded-degree graphs, [5] and also [3] independently of us, found an optimal algorithm which runs in $O(\log n)$ time using $n/\log n$ processors. Our algorithm is simpler compared with that of [5]. The best known algorithm for the maximal matching problem on planar graphs was due to [2], and runs in $O(\log^2 n)$ time using a linear number of processors on an EREW PRAM. The application of our optimal prefix computation technique to the 7-coloring and the maximal independent set problems achieves the same resource bounds (as maximal matching) using linear space. It must be mentioned here that faster (and optimal with respect to time-processor tradeoff) NC algorithms for these problems have been previously found [3, 4], but they use $O(n^2)$ space. Thus in the above sense our algorithms are space-efficient in the same model of computation (EREW PRAM).

The results for rectangular grid graphs are completely new. We are not aware of any other parallel algorithm for the same problems on grid graphs.

2 Definitions and Notation

Throughout this paper we employ simple, undirected graphs $G = (V, E)$ with n nodes. A *coloring* of a graph G is an integer function $C : V \rightarrow N$. A coloring is *valid* if no two adjacent nodes have been assigned the same color. Given $G = (V, E)$ and a coloring $C : V \rightarrow C_v$, we say that C is a *maximal* coloring (MC) of G if C is valid, and if v is not assigned a color then each color in C_v must be assigned to at least one neighbor of v. A subset of nodes $I \subset V$ is called an *independent set* if

$\forall v, u \in I : \{v, u\} \notin E$. A maximal independent set (MIS) is an independent set I such that $\nexists I', I'$ is an independent set and $I' \supset I$. An independent set is called *maximum* if it has the maximum cardinality among the all possible MIS in G. A MIS I in a tree T is called a *maximal dividing set* (MDS) if the removal of the nodes of I dissects T into trees of $O(1)$ height. Note that the MIS problem is a special case of the MC problem, where $C_v = \{c_1\}$. The set of vertices colored c_1 is a MIS. The $(\Delta + 1)$ vertex coloring problem ($(\Delta + 1)$VC) is another special case of the MC problem where $C_v = \{c_1, c_2, ..., c_{\Delta+1}\}$.

A *flat forest* of a graph $G = (V, E)$ is a forest $F = (V, E')$ such that $E' \subset E$ and each tree in F has height at most 1. Any zero-degree node in G is a zero-degree node in F, too. Given an undirected graph $G = (V, E)$, a *matching* is a set of edges $M \subset E$ in which no two edges share the same node. A matching is *maximal* if it is not properly contained in any other matching. A matching is *maximum* if it has the maximum cardinality among the all possible maximal matchings of G. A *Hamiltonian path* between two nodes s, t of a graph G is a simple path starting on s, visiting all the remaining nodes exactly once, and ending to t.

A grid graph is a finite node-induced subgraph of the infinite grid G^∞. The node set of G^∞ consists of all points of the plane with integer coordinates. Two nodes are connected iff their Euclidean distance is equal to 1. A grid graph is completely specified by its node set. A grid graph $R(m, n)$ is called *rectangular* if its node set $V(R(m, n)) = \{v : 1 \leq v_x \leq m, 1 \leq v_y \leq n\}$, where v_x and v_y are the x and y coordinates of v respectively. A node that has degree 2 or 3 is called a *boundary* node. The four nodes of degree 2 are called *corners*. A node v is called *even* if $v_x + v_y \equiv 0 \pmod 2$; otherwise it is called *odd*. Then $R(m, n)$ can be considered as a bipartite graph (V_0, V_1, E), where $V_0(V_1)$ is the set of even (odd) nodes, $V(R(m, n)) = V_0 \cup V_1$ and either $|V_0| = |V_1|$ or $|V_0| = |V_1| + 1$. We can also consider the nodes of the graph as colored by two colors. All nodes in V_0 are colored white and all nodes in V_1 are colored black. Furthermore, $R(m, n)$ is called even (odd) if $m \times n$ is even (odd). In the sequel with the term grid graph we shall denote a rectangular grid graph.

The model used in this paper is the PRAM model of computation where each processor may have access to a shared-memory (or to its local memory) and may execute simple word and bit operations in constant time (see [12]). By allowing or disallowing simultaneous access by more than one processor to the same memory location, we get a lot of variants of the PRAM model. In this paper we use the CRCW PRAM (which allows simultaneous access to the same memory location for read or write purposes by more than one processor) and the EREW PRAM (which does not allow simultaneous access to the same memory location by more than one processor).

3 The Basic Algorithms

In this section we give optimal algorithms, (i) for the 2-coloring of a tree (and thus an optimal algorithm for finding a MIS, a MDS, and a MC of a tree), and (ii) for the GLR problem and the prefix computation problem on graphs. All the algorithms in this section are based on the Euler tour technique of [11] and especially on its optimal parallel version [10].

The Euler tour technique provides an easy way to compute simple tree functions as pre- and post-order numbering of nodes in a tree, the level and the number of descedants of each node, etc. The idea is to reduce the computation of these functions to weighted list ranking. Because of the EREW model the tree must be ordered. (That is, we have to know the relative order of the subtrees of a node, see e.g [7] page 306.) The tree may be rooted or unrooted. We briefly describe here the Euler tour technique for the unrooted case. (The rooted case is similar.) Given an unrooted tree T, we can convert it into an Eulerian digraph T_E as follows: replace each edge $\{u, v\}$ with two directed arcs (u, v) and (v, u). (We assume also that a pointer is created from each (u, v) to its reversal (v, u).) Then in $O(1)$ time with n processors we can find an Euler tour in T_E as follows. The Euler tour is represented into a vector E, where $\forall e \in T_E$, $E(e)$ will have the succesor edge of e in the tour. For each edge $(a, b) \in T_E$, its next edge in E is the edge next to its reversal (b, a) in the adjacency list of b. (The adjacency lists in T_E can be easily obtained since T is ordered.) If (b, a) is the last edge in the adjacency list, then its next edge will be the first edge of the adjacency list. We pick any edge (u, v) to be the first edge of the Euler tour (and this implies that u is the root of T). Then E represents a depth-first search of T starting at u. Now, we can use list ranking to determine the parent-child

relationship as well as to compute the tree functions. (We make the correction $E(w,u) := null$ where w is the last child of the root u.) To establish the parent-child relationship we have only to observe that each undirected edge $\{a,b\}$ appears twice in E (once as (a,b) and once as (b,a)). If (a,b) preceeds (b,a) then a is the father of b. Otherwise b is the father of a.

We can compute, for example, the functions $preorder(v)$ and $level(v)$ for each vertex $v \in T$ if we employ the optimal list ranking algorithm of Cole and Vishkin [1]: (i) with weights (w_p) 1 for the forward arcs of T_E and 0 for the backward arcs (towards the root), for the *preorder* function, and (ii) with weights (w_l) 1 for the forward arcs and -1 for the backward ones, for the *level* function. Then, $preorder(x) = rank_p(y,x) + 1$ and $level(x) = rank_l(y,x)$ where $rank_p$ ($rank_l$) are subject to the weights w_p (w_l). Thus we have the following.

Lemma 1 *The preorder number and the level of each vertex of a tree can be computed optimally in $O(\log n)$ time using $n/\log n$ processors in the EREW PRAM model of computation.*

3.1 Coloring Trees

Assume that we are given a tree and the colors $C = \{c_1, c_2\}$. The outline of the 2-coloring algorithm is the following:

Algorithm *2-Color-Tree*
begin
(1) Compute level numbers, for each node in the tree (using the Euler tour technique)
(2) For each node v, color odd level nodes with one color and even level nodes with the other
end

Theorem 1 *The algorithm 2-Color-Tree provides a valid 2-coloring of a tree in $O(\log n)$ time using $n/\log n$ processors in the EREW PRAM model of computation.*

Proof: The time and processor bounds follow from Lemma 1, since step 2 of the algorithm, implemented with n processors in $O(1)$ time, can be easily simulated to run in $O(\log n)$ time using $n/\log n$ processors. The validity of the coloring is obvious. ∎

Corollary 1 *A MIS of a tree can be found in $O(\log n)$ time using $n/\log n$ processors on an EREW PRAM.*

Corollary 2 *A MDS of a tree can be found in $O(\log n)$ time using $n/\log n$ processors on an EREW PRAM.*

The algorithm for 2-coloring a tree T can be extended (in the obvious way) to maximally color any tree. Thus we have the following.

Corollary 3 *A maximal coloring of a tree can be computed in $O(\log n)$ time using $n/\log n$ processors in an EREW PRAM.*

The coloring algorithms presented in this section work also on any forest F. We create a supernode called super-root and connect each root $r_1, r_2, ..., r_m$ of the trees of the forest to this node. (We assume that the roots are connected by a linked list.) Then, we apply the algorithms with the modification that the edges $(r_i, super - root)$ and their reversals have zero (0) initial weights in the list ranking algorithm. Thus, we have proved the following:

Theorem 2 *Let F be a forest, having a total of n nodes. Then simple tree functions as well as a maximal coloring of F can be computed in time $O(\log n)$ using $n/\log n$ processors in the EREW PRAM model of computation.*

3.2 The GLR algorithm

The algorithm for the GLR problem is also based on the Euler tour technique. First, recall the definition of the GLR problem. The k linked lists can be thought of as k trees of special kind (long chains). The edges of such a tree are directed towards the unique leaf of the tree. As we saw, it is very easy to convert each tree into an Eulerian digraph. Furthermore, the roots (heads of the lists) are connected by a linked list (recall the array H). Hence, it is like having a forest of k trees with a total of n nodes. Therefore, theorem 2 applies. The rank of each element (of the k lists) as well as its membership relation can be computed as follows. We assume that the elements are refered now by their preorder numbers, and that each element v has two fields: $rank(v)$ (for keeping its rank in the list) and $head(v)$ (where the pointer to the head of its list is stored). Then, in $O(1)$ time every processor associated with the node v executes the following: (1) $rank(v) = level(v)$; (2) $head(v) = preorder(v) - level(v) + 1$.

Remark. Although the Euler tour technique needs (in general) $O(n^2)$ space for the computation of the pointers between the reversal edges, here we can create these pointers using only $O(n)$ space. This is because the trees are of a special kind (long chains).

Theorem 3 *The general list ranking problem can be solved in $O(\log n)$ time using $n/\log n$ processors on an EREW PRAM.*

Proof: Follows from the above discussion and by theorem 2. ∎

The above theorem has an immediate application to the prefix computation problem on graphs which is often encountered as a subroutine in many graph algorithms.

Theorem 4 *Given a graph G with n nodes and m edges in its adjacency list representation, we can perform any parallel prefix computation on the elements of the adjacency lists of G in $O(\log n)$ time using $m/\log n$ processors on an EREW PRAM.*

Proof: From theorem 3, we can find (for every element of the adjacency lists) the rank and the head of the list (to which it belongs) in $O(\log n)$ time using $m/\log n$ processors. Having this information we can convert the lists into arrays in $O(1)$ time using m processors (each element knows the array as well as its position in it). Now, we have n arrays, each of length m_i, $0 \leq m_i \leq n, 1 \leq i \leq n$ and $\sum_{i=1}^{n} m_i = m$. Any parallel prefix computation algorithm can be performed (in such an array) in $O(\log m_i)$ time using $m_i/\log m_i$ processors on an EREW PRAM. But the same algorithm can be performed in $O(\log n)$ (greater) time using $m_i/\log n$ (fewer) processors by simulation. Thus, the prefix computation can be done to all the arrays in time $O(\log n)$ using $\sum_{i=1}^{n} m_i/\log n = m/\log n$ processors on an EREW PRAM. ∎

4 Coloring of Bounded-Degree Graphs

In this section we show how to improve the algorithms presented in [2] for the $(\Delta+1)$VC and the MIS problems in bounded-degree graphs, with the use of our coloring algorithms for trees and thus achieving optimal speedup. We assume that the maximum degree of any node is Δ (Δ is a constant) and that each node has a unique name (id) between 1 and n. The main idea of our algorithm is similar to that of [2] for the algorithm *Color-Constant-Degree-Graph*. The difference is that *we do not use the pseudo-forest technique*. The algorithm works in two phases. In the first phase a forest is created and in each iteration we delete its edges from the initial graph G. This phase stops when no edges remain on G. Then we perform an initial coloring of nodes in G (with one color). In the second phase we iteratively 2-color the nodes of the current forest and return them to G. At the beginning of each iteration of the second phase the new edges added make the existing $(\Delta+1)$-coloring invalid. For this reason the iteration terminates with an appropriate recoloring of the nodes with a color different from the colors of its neighbors. The recolorability can be done in parallel since the nodes having the same color are independent. A more formal description of the algorithm follows:

Algorithm *Color-Bounded-Degree-Graph*(V, E)
(01) $E' := E$
(02) $i := 0$
(03) (* first phase *)
(04) **while** $E' \neq \emptyset$ **do**
(05) **for all** $v \in V$ **pardo**
(06) find a $u \in V : id(u) := min\{id(u_l) | \{v, u_l\} \in E', 1 \leq l \leq \Delta\}$
(07) **if** $id(v) > id(u)$ **then** $E_i := E_i \cup (v, u)$
(08) **odpar**
(09) $E' := E' - E_i$ (* E_i are edges of a forest *)
(10) $i := i + 1$
(11) **od**
(12) **for all** $v \in V$ **pardo** (* initial coloring *)
(13) $C(v) := 1$
(14) **odpar**
(15) (* second phase *)
(16) **for** $i := i - 1$ **downto** 0 **do**
(17) $C' :=$2-Color-Forest(V, E_i) (* 2-coloring of the current forest *)
(18) $E' := E' + E_i$
(19) **for** $j := 1$ **to** $\Delta + 1$ **do**
(20) $V' := V$
(21) **for all** $v \in V'$ **pardo**
(22) **if** $C(v) = j$ and $C'(v) = 2$ **then begin**
(23) $C(v) := max\{\{1, 2, ..., \Delta + 1\} - \{C(w) | (v, w) \in E'\}\}$
(24) $V' := V' - \{v\}$
(25) **end**
(26) **odpar**
(27) **od**
(28) **od**

Theorem 5 *The algorithm Color-Bounded-Degree-Graph provides a valid* $(\Delta + 1)$*-coloring of a bounded-degree graph with maximum degree* Δ *in* $O(\log n)$ *time using* $n/\log n$ *processors in the EREW PRAM model of computation.*

Proof: Correctness follows from arguments similar to those of [2]. Here, we prove the claimed time and processor bounds. The first phase of the algorithm terminates after at most Δ iterations and each iteration takes at most Δ time (each processor has to find the minimum among Δ elements). Thus, the first phase needs n processors and $O(1)$ time. The same time and processor bounds hold for the initial coloring. The second phase terminates obviously in at most Δ iterations. Each iteration consists of two steps. In the first step we 2-color the current forest in $O(\log n)$ time using $n/\log n$ processors on an EREW PRAM. (From the adjacency list representation of the graph we can construct the appropriate input to our coloring algorithms.) In the second step we have two nested for-loops. The for-loop of line (19) iterates $\Delta + 1$ times. The other for-loop of line (22) needs $O(\Delta)$ time for each iteration because we must find the maximum among at most $\Delta + 1$ colors. Therefore, one iteration of the second step needs n processors and $O(1)$ time. Hence, the second step can be easily simulated to run in $O(\log n)$ time using $n/\log n$ processors, thus achieving the same time and processor bounds for the second phase. A similar simulation can be applied to the first phase and the initial coloring phase, too. Notice that there are no read or write conflicts. ∎

It is easy to find a MIS in a bounded-degree graph, if we have a $(\Delta + 1)$-coloring of the graph. The algorithm works as follows: iterate over all colors, each time taking the nodes of the current color, adding them to the MIS I and deleting them and all their neighbors from the initial graph, and continue with the remaining graph. Call the above algorithm *Bounded-Degree-MIS.*

Corollary 4 *The algorithm Bounded-Degree-MIS provides a MIS of a bounded-degree graph in $O(\log n)$ time using $n/\log n$ processors in an EREW PRAM.*

5 7-Coloring and Maximal Matching in Planar Graphs

In this section we consider the problems of 7-coloring, finding a MIS and a maximal matching in planar graphs. We show how to improve the algorithms of [2] for the above problems using the algorithms presented in the previous sections. Especially the optimal solution of the prefix computation problem on graphs leads to a faster, optimal algorithm for the maximal matching problem. The application of this solution to the two other problems leads to optimal algorithms that use only $O(n)$ space, but it does not improve the running time of previously known optimal algorithms for the same problems (see [3, 4]). We shall give first some technical lemmas.

Lemma 2 *In a planar graph, the number of nodes with degree at most 6 is greater than $\frac{1}{7}n$.*

Proof: Suppose in the contrary that the number of nodes with degree at most 6 are $\leq \frac{1}{7}n$. Then $n - \frac{n}{7}$ nodes have degree at least 7. Let d_i be the degree of the node i. We have that $D = \sum_{i=1}^{n} d_i \geq (n - \frac{n}{7}) \cdot 7 = 6n$. But from Euler's formula $D \leq 6n - 12$, a contradiction. Thus the lemma is proved. ∎

The algorithm for 7-colorability of planar graphs is based on a procedure, called *Planar-LIS*, which finds a large independent set (LIS) of nodes of degree at most 6. The procedure is the following.

Procedure *Planar-LIS*
begin
(1) Construct a graph G', induced by the nodes of G with degree at most 6.
(2) I :=Bounded-Degree-MIS(G')
end

Lemma 3 *Let I be the independent set produced by the above algorithm. Then $|I| \geq \frac{1}{7}|G'|$.*

Proof: Notice that the application of the Color-Bounded-Degree-Graph algorithm to G' partitions its nodes into 7 classes. Then the larger class is at least $\frac{1}{7}|G'|$. ∎

Now, the algorithm for 7-colorability works as follows. If the planar graph G has maximum degree 6, then call *Color-Bounded-Degree-Graph* and stop. Otherwise, find a LIS using the *Planar-LIS* procedure and delete its nodes from G. Recurse to find a 7-coloring of $G - I$. Finally, recolor each node in I to produce a valid 7-coloring. Less informally we write the algorithm as follows:

Algorithm *7-Color-Planar-Graph(G)*
begin
(1) $G' := G$; $I := \emptyset$;
if $\nexists v \in G'$ with $deg(v) > 7$ **then** Color-Bounded-Degree-Graph(G')
else begin
I :=Planar-LIS(G')
$G' := G' - I$
7-Color-Planar-Graph(G')

end
(2) color each $v \in I$ to produce a valid 7-coloring
end

Theorem 6 *The algorithm 7-Color-Planar-Graph provides a valid 7-coloring of a planar graph in $O(\log^2 n)$ time with $n/\log^2 n$ processors in the EREW PRAM model of computation using $O(n)$ space.*

Proof: The correctness of the algorithm can be easily seen. Here we are concerned only with the resource bounds. Since I is an independent set, the vertices in I can be colored independently. Thus, step 2 is implemented in $O(1)$ time using n processors. The *Planar-LIS* procedure runs in $O(\log n)$ time using $n/\log n$ processors according to corollary 4. In each iteration we must compact the edge lists, because of the EREW model. Since the compaction is nothing more than an application of a parallel prefix computation algorithm, it can be done in $O(\log n)$ time using $n/\log n$ processors (according to theorem 4). Obviously the edge-list compaction keeps the space linear. There are $O(\log n)$ iterations since procedure *Planar-LIS* produces a LIS of $\Theta(n)$ size. Therefore, the whole algorithm runs in $O(\log^2 n)$ time using $n/\log n$ processors. However, observe that at each iteration the computation is reduced by a constant factor. In the first iteration the computation is n, in the second $\frac{48}{49}n$ (at most, due to Lemmas 2 and 3), etc. Generally, the i-th iteration has computation $\varepsilon^i \cdot n$ where $\varepsilon \leq \frac{48}{49}$ and can be implemented by $\varepsilon^i \cdot n/\log \varepsilon^i \cdot n$ processors in $O(\log \varepsilon^i \cdot n)$ time. But notice that the i-th iteration can also be implemented in $O(\varepsilon^i \cdot \log^2 n)$ (greater) time, using $n/\log^2 n$ (fewer) processors (by simulation). This results in a total time of $\sum_{i=0}^{\log n} O(\varepsilon^i \cdot \log^2 n) = O(\log^2 n)$ for the whole algorithm, but now using only $n/\log^2 n$ processors. ∎

We can easily find a MIS in a planar graph after a 7-coloring, in the same way as we did in bounded-degree graphs. Therefore, we have the following corollary.

Corollary 5 *A MIS in a planar graph can be computed in $O(\log^2 n)$ time by an EREW PRAM with $n/\log^2 n$ processors using linear space.*

Now we show how to improve the algorithm of [2] to find a maximal matching in a planar graph. The main idea of our algorithm is almost the same, but *we use a forest* (instead of a pseudo-forest) for the building of a flat forest of a graph. An outline of the algorithm follows.

Algorithm *MM-Planar-Graph*
begin
$G' := G$
repeat
(1) *find a flat forest of G'*
(2) *arbitrarily add one edge from each tree in the forest to M*
(3) $G' := G' - M$
until $G' = \emptyset$
end

Lemma 4 *A flat forest of a planar graph G can be constructed in $O(\log n)$ time using $n/\log n$ processors in an EREW PRAM.*

Proof: To find a flat forest in a planar graph $G = (V, E)$ we proceed as follows. First, we find a forest $P = (V, E')$ of G in the same way as we did in the first phase of the algorithm *Color-Bouned-Degree-Graph*. Note that every node $v \in V$ has to select an edge $\{v, u\}$ and to direct it outwards if the following are satisfied: (1) $id(u)$ is the smallest id among the id's of the neighbors of v, and (2) $id(v) > id(u)$. Thus every node v must know the element with the smallest id in its adjacency list. To find that element is nothing more than an application of a parallel prefix min algorithm to the adjacency list of v. According to theorem 4, this can be done in $O(\log n)$ time using $n/\log n$ processors on an EREW PRAM. Thus, the forest P can be constructed in the same time and processor bounds in the EREW PRAM model. Second, find a MIS I of P. The flat forest $F = (V, E'')$ is then constructed as follows:

1. $\forall v \notin I$, $E'' := E'' \cup (v, w)$, $w \in I$ and $(v, w) \in E'$
2. $\forall$ node $v \in I$ that has no adjacent edges in E'' but some adjacent edges in E', choose one adjacent edge in E' and add it to E''

The graph F induced by the edges in E'' is almost a flat forest since each tree has height at most 2. We can split those trees of height 2 to height 1 trees, to get a flat forest. All operations take $O(1)$ time using n processors, except for the construction of P and the MIS which need $O(\log n)$ time with $n/\log n$ processors. Thus, we can find a flat forest in $O(\log n)$ time using $n/\log n$ processors in an EREW PRAM. ∎

Theorem 7 *The algorithm MM-Planar-Graph produces a maximal matching of a planar graph in $O(\log^2 n)$ time with $n/\log^2 n$ processors in the EREW PRAM model of computation using $O(n)$ space.*

Proof: Each iteration is dominated by the finding of a flat forest. Operations (2) and (3) can be run in $O(1)$ time using n processors. Again, after operation (3) we must compact the edge lists. Thus each iteration takes $O(\log n)$ time using $n/\log n$ processors. There are $O(\log n)$ iterations since a flat forest has at least $n/2$ edges (see [2]). Therefore, the algorithm runs in $O(\log^2 n)$ time using $n/\log n$ processors. Again, by observing that the computation reduces by a constant factor (of at least $1/2$) after each iteration and by applying the same simulation as in the proof of theorem 6 (but now with $\varepsilon \leq 1/2$), we have the desired resource bounds. ∎

6 Algorithms for Grid Graphs

6.1 A Basic Algorithm and its Applications

Assume that a grid graph $R(m, n)$ is given as a set of adjacency lists ($R = |V(R(m, n))|$). In sequential computation it is straightforward to convert the adjacency list representation into another one in which every node v is represented by its x and y coordinates in the plane. We call such a representation the *xy representation* of a grid graph. The conversion takes time linear on the size of the graph. The xy representation is very useful in the solution of many problems on grid graphs (see e.g. [6]), and it also is the key to find a maximum independent set, a maximum matching and a Hamiltonian path.

However, the parallel conversion to the xy representation is not obvious. We shall give an optimal parallel algorithm for this problem. The algorithm depends on the complexity of finding the embedding of a planar graph. Two algorithms are known for finding the embedding. The most recent one [9] is optimal for the CRCW PRAM and runs in $O(\log n)$ time with $n/\log n$ processors. (n is the size of the graph.) The other one [8] runs in $O(\log^2 n)$ time with n processors on a CREW PRAM. Note that only the algorithm for finding the embedding prevents our algorithm to be optimal on an EREW PRAM in $O(\log n)$ time.

In the sequel we assume that our input is an embedded grid graph, i.e. each node is associated with a cyclic list of its neighbors in clockwise order. The algorithm for converting the above representation into a xy one is the following (see also fig.1):

Algorithm XY
begin
(1) Construct a linked list of the boundary nodes and find their ranks (assuming as starting point the node a). This can be done as follows. Let z be any of the boundary nodes $\{a, b, c, d\}$. Temporarily we add one pseudo-edge (dashed line) to z and update the embedding (see fig.1). (This can be done in $O(1)$ time since the maximum degree in a grid graph is 4.) Now, consider the induced subgraph G_b of nodes having degree ≤ 3. Then G_b contains only the boundary nodes. We first construct a list of edges as follows. Let u_1, u_2, u_3 be boundary nodes (see fig.1). Then $next(u_1, u_2) = next_e(u_2, u_1) = (u_2, u_3)$, where $next_e$ refers to the next edge in the embedding. Note that the constructed list is a clockwise one. We can not have a similar counterclockwise list because e.g. $next(u_3, u_2) = next_e(u_2, u_3) = null$

(since the edge (u_2, w) does not exist). The list can be broken by setting $next(t, a) = null$. Now, we are ready to execute the optimal parallel algorithm of [1] to find the rank (distance from a) of each boundary node. Obviously $rank(v) = rank(u, v)$.
(2) Associate a processor to each boundary node v. Then execute the following 4 steps sequentially.
(2.1) **if** $rank(v) \in [rank(a), rank(b)]$ **then** $(v_x, v_y) = (1, rank(v) + 1)$
(2.2) **if** $rank(v) \in (rank(b), rank(c)]$ **then** $(v_x, v_y) = (rank(v) - rank(b) + 1, b_y)$
(2.3) **if** $rank(v) \in (rank(c), rank(d)]$ **then** $(v_x, v_y) = (c_x, rank(d) - rank(v) + 1)$
(2.4) **if** $rank(v) \in (rank(a), rank(d))$ **then** $(v_x, v_y) = (rank(t) - rank(v) + 2, 1)$
(3) Construct lists of nodes as follows. Associate a processor with node v. Let u be an adjacent node of v. Each processor executes the following:
if $deg(u) = 4$ **then** $next(v, u) = next_e(next_e(u, v))$ **else** $next(v, u) = null$.
(4) Consider only the lists whose head h satisfies: $rank(h) \in (rank(a), rank(b))$ or $rank(h) \in (rank(a), rank(d))$. Apply the GLR algorithm of section 3.2. If a node v belongs to a list such that $rank(h) \in (rank(a), rank(b))$ then $(v_x, v_y) = (rank(v)+1, h_y)$. If $rank(h) \in (rank(a), rank(d))$ then $(v_x, v_y) = (h_x, rank(v) + 1)$.
end

Theorem 8 *The xy representation of a (rectangular) grid graph can be computed in $O(\log R)$ time with $R/\log R$ processors on a CRCW PRAM.*

Proof: From the above discussion, theorem 3 and the algorithm of [9] for finding the embedding of a planar graph. ∎

Corollary 6 *Given the embedding of a (rectangular) grid graph, we can find the xy representation in $O(\log R)$ time with $R/\log R$ processors on an EREW PRAM.*

The XY algorithm has immediate applications to the problems of maximum independent set, maximum matching and Hamiltonian cycle on grid graphs.

The maximum independent set of a grid graph is the set of even (white) nodes (V_0). Given the xy representation this can be computed in $O(1)$ time.

For the maximum matching we have two cases. First, associate a processor with every even node v.
Case 1: n or m is even. Suppose that m is even. Then each processor (associated with an even node v) executes the following:

if $v_y \equiv 1 \pmod 2$ **then** $M := M \cup \{v, u\}$, where $u_x = v_x + 1$
else $M := M \cup \{v, u\}$, where $u_x = v_x - 1$

Case 2: both n and m are odd. Each processor associated with an even node v such that $v_x \leq m - 1$ performs the same as in case 1. If $v_x = m$, then $M := M \cup \{v, u\}$ where u is odd and $u_y = v_y + 1$.

In a similar way we can find a Hamiltonian cycle (HC) (starting e.g. at some node $z \in \{a, b, c, d\}$). For there exists a Hamiltonian cycle one dimension (n or m) must be even. The cycle can be easily found in $O(1)$ time as fig.2 shows.

Corollary 7 *A maximum independent set, a maximum matching and a Hamiltonian cycle in a (rectangular) grid graph can be computed: (i) in $O(\log R)$ time with $R/\log R$ processors on a CRCW PRAM, (ii) in $O(\log R)$ time with $R/\log R$ processors on an EREW PRAM, if the embedding is given, and (iii) in $O(1)$ time (with R processors) on an EREW PRAM, if the xy representation is provided.*

6.2 Hamiltonian Paths on Grid Graphs

The problem of finding a Hamiltonian path on a grid graph was first studied in [6]. In that paper ([6]) necessary and sufficient conditions for the existence of a Hamiltonian path are provided, as well

as an algorithm is given which finds the Hamiltonian path between any two nodes (if it exists) in $O(R)$ time. However, the algorithm of [6] does not seem to be easily parallelizable.

Although we adopt some main concepts of the study of the problem from [6] (e.g. definitions, conditions, etc), we present here a different algorithm for finding a Hamiltonian path between any two nodes (if it exists) on a grid graph. Our algorithm is easy and amenable to parallelization.

First recall some definitions from [6]. A Hamiltonian path problem $HP(R(m,n),s,t)$ between two nodes s,t is called *color compatible* if (a) $R(m,n)$ is even and s,t have different colors, or (b) $R(m,n)$ is odd and both s,t are white. We say that $HP(G,s,t)$ is *forbidden* if one of the following is satisfied:

1. G is of the form $R(m,1)$ and either s or t is not a corner.
2. G is of the form $R(m,2)$ and $\{s,t\}$ is not a boundary edge.
3. $HP(G,s,t)$ is isomorphic to $HP(G',s',t')$ which satisfies the following:
   (a) $G' = R(m,3)$ where m is even.
   (b) s' is colored differently from t' and the left corners of G'.
   (c) $s'_x < t'_x - 1$, or $(s'_y = 2$ and $s'_x < t'_x)$.

A Hamiltonian path problem is called *acceptable* if it is color compatible and not forbidden. The following theorem has been proved in [6].

Theorem 9 *Acceptability is a necessary and sufficient condition for $HP(R(m,n),s,t)$ to have a solution.*

In the sequel, without loss of generality we assume that $s_x \leq t_x$ and $s_y \geq t_y$. The main idea of our algorithm for finding a Hamiltonian path in a grid graph is the following. (We assume that $HP(R(m,n),s,t)$ is acceptable.) Reduce the initial problem to $HP(R(m_1,n_1),s,t)$ such that one of the following is satisfied:

(P1) $s_x \leq 2$, $s_y \geq n_1 - 1$, $t_x \geq m_1 - 1$ and $t_y \leq 2$.
(P2) $s_x \leq 3$, $s_y \geq n_1 - 2$, $t_x \geq m_1 - 2$, $t_y \leq 3$, and either ($n_1 = 5$ and m_1 is even) or (n_1 is even and $m_1 = 5$).

Split $HP(R(m_1,n_1),s,t)$ into two subproblems $HP(S,s,t_1)$ and $HP(T,s_1,t)$ where $\{t_1,s_1\}$ will be an edge of $HP(R(m,n),s,t)$. Solve these subproblems and merge their solution using the edge $\{t_1,s_1\}$.

The reduction of the initial problem consists of the following four steps (see fig.3).
(1) Consider the subgraph R_1 of R such that $V(R_1) = \{v : v_x \leq x_0$ where $x_0 \leq s_x - 1$ and x_0 is the maximum value such that R_1 has a HC and $HP(R - R_1,s,t)$ is acceptable$\}$. Let $R' = R - R_1$.
(2) Consider the subgraph R_2 of R' such that $V(R_2) = \{v : v_x \geq x_0$ where $x_0 \geq t_x + 1$ and x_0 is the minimum value such that R_2 has a HC and $HP(R' - R_2,s,t)$ is acceptable$\}$. Let $R' = R' - R_2$.
(3) Consider the subgraph R_3 of R' such that $V(R_3) = \{v : v_y \geq y_0$ where $y_0 \geq s_y + 1$ and y_0 is the minimum value such that R_3 has a HC and $HP(R' - R_3,s,t)$ is acceptable$\}$. Let $R' = R' - R_3$.
(4) Consider the subgraph R_4 of R' such that $V(R_4) = \{v : v_y \leq y_0$ where $y_0 \leq t_y - 1$ and y_0 is the maximum value such that R_4 has a HC and $HP(R' - R_4,s,t)$ is acceptable$\}$. Let $R' = R' - R_4$ with dimensions m_1, n_1.

Note that some R_i, $1 \leq i \leq 4$ may be empty. One can easily see that $HP(R',s,t)$ satisfies (P1) or (P2). The HC in each R_i can be found as it was explained in the previous section. Hence it remains to show how $HP(R'(m_1,n_1),s,t)$ can be found. (For if we find the Hamiltonian path $HP(R'(m_1,n_1),s,t)$, we can merge it with the HC of each R_i as fig.4 shows.)

Suppose that $HP(R',s,t)$ satisfies (P1). We have four cases depending on the parity of m_1, n_1 (odd or even).
Case P1.1: both m_1, n_1 are even. We split $HP(R'(m_1,n_1),s,t)$ into $HP(S,s,t_1)$ and $HP(T,s_1,t)$ where $\{t_1,s_1\}$ is an edge of $HP(R(m,n),s,t)$. The subgraph S of R' is defined by $V(S) = \{v : v_x \leq$

$m_1 - 2\}$. Then $T = R' - S$. The edge $\{t_1, s_1\}$ is determined as follows.
(a) if $(s_x, s_y) \in \{(1, n_1), (2, n_1 - 1)\}$ then $(t_{1_x}, t_{1_y}) = (m_1 - 2, n_1)$ and $(s_{1_x}, s_{1_y}) = (m_1 - 1, n_1)$
(b) if $(s_x, s_y) \in \{(2, n_1), (1, n_1 - 1)\}$ then $(t_{1_x}, t_{1_y}) = (m_1 - 2, n_1 - 1)$ and $(s_{1_x}, s_{1_y}) = (m_1 - 1, n_1 - 1)$

That is, each of $HP(S, s, t_1)$ and $HP(T, s_1, t)$ must be color compatible.
Case P1.2: both m_1, n_1 are odd. We split $HP(R'(m_1, n_1), s, t)$ into two subproblems (as above) such that $V(S) = \{v : v_x \leq m_1 - 3\}$ and $T = R' - S$. (We assume that $m_1 > 3$. If $m_1 = 3$, there is no split. We shall see the solution for this case later.) Again both $HP(S, s, t_1)$ and $HP(T, s_1, t)$ must be color compatible, i.e. since $(s_x, s_y) \in \{(1, n_1), (2, n_1 - 1)\}$ we have that $(t_{1_x}, t_{1_y}) = (m_1 - 3, n_1)$ and $(s_{1_x}, s_{1_y}) = (m_1 - 2, n_1)$.

The other two cases are similar to the first case.

Lemma 5 *Both $HP(S, s, t_1)$ and $HP(T, s_1, t)$ are acceptable.*

Proof: Observe that they are color compatible by construction. Consider the subgraph S_1 of S defined by either $V(S_1) = \{v : 1 \leq v_x \leq m_1 - 2 \text{ and } 1 \leq v_y \leq n_1 - 2\}$ (case P1.1), or by $V(S_1) = \{v : 1 \leq v_x \leq m_1 - 3 \text{ and } 1 \leq v_y \leq n_1 - 2\}$ (case P1.2). Let $S_2 = S - S_1$ (see fig.5). From the construction of S we have that either $m_1 - 2$ ($m_1 - 3$) or n_1 or both are even. Therefore the subgraph S_1 has a HC. The subgraph S_2 is of the form $S_2(m_1 - 2, 2)$ (or $S_2(m_1 - 3, 2)$) and there is no case such that $\{s, t_1\}$ is a nonboundary edge. Thus $HP(S_2, s, t_1)$ is not forbidden.

Also $HP(T(c, n_1), s_1, t)$ (where $c = 2$ or $c = 3$) is not forbidden since $\{s_1, t\}$ is not a nonboundary edge ($c = 2$) or n_1 is odd ($c = 3$). Thus the lemma is proved. ∎

From the above it is obvious that in order to find a solution for $HP(R'(m_1, n_1), s, t)$ it is enough to find a HC in S_1 and solve $HP(S_2, s, t_1)$ and $HP(T, s_1, t)$, and merge the solutions. Finding a solution for $HP(S_2, s, t_1)$ is the same as finding a solution for $HP(R(m, 2), s, t_1)$ where $s_x \leq 2$ and $t_{1_x} = m$. For a solution to exist it is enough for s, t_1 to be differently colored. Then the path is constructed easily (one case is shown in fig.6.a). The solution of $HP(T(c, n_1), s_1, t)$ where $s_{1_x} = 1$ and $t_y \leq 2$, is similar to the above if $c = 2$. In the case where $c = 3$ a Hamiltonian path can be also easily constructed. (Fig.6.b shows one case.)

Suppose now that $HP(R', s, t)$ satisfies (P2). In this case we provide the solution for $n_1 = 5$ and m_1 is even. (The other case is similar.) If $(s_x, s_y) \in \{(1, 5), (1, 4), (2, 5), (2, 4)\}$ and $(t_x, t_y) \in \{(m_1 - 1, 1), (m_1 - 1, 2), (m_1, 1), (m_1, 2)\}$, then we follow case (P1). Hence it remains to examine the cases where $s_y = 3$ and either $t_y = 2$ or $t_y = 3$. The solutions can be easily obtained and illustrated in fig.7-8.

The cases where $s_y = 2$ and $t_y = 3$ are similar.

Theorem 10 *A Hamiltonian path (if it exists) between any two nodes of a (rectangular) grid graph can be computed: (i) in $O(\log R)$ time with $R/\log R$ processors on a CRCW PRAM, (ii) in $O(\log R)$ time with $R/\log R$ processors on an EREW PRAM, if the embedding is given, and (iii) in $O(1)$ time (with R processors) on an EREW PRAM, if the xy representation is provided.*

Proof: Follows from the above discussion. ∎

7 Acknowledgements

We are greatful to Lefteris Kirousis for his essential help in many technical discussions and to Christos Papadimitriou for his careful reading of a preliminary version of this paper and his encouragment.

References

[1] R.Cole, U.Viskin, "Approximate and Exact Parallel Scheduling with applications to list, tree and graph problems", Proc. 27th IEEE Symp. on FOCS, 1986, pp. 478-491.

[2] A.Goldberg, S.Plotkin, G.Shannon, "Parallel Symmetry-Breaking in Sparse Graphs", Proc. of the ACM 19th STOC (Symp. on Theory of Computing), 1987, pp. 315-324.

[3] T.Hagerup, "Parallel Algorithms on Planar Graphs", Ph.D. Thesis, University of Saarlandes, Saarbrücken, 1988.

[4] T.Hagerup, M.Chrobak, K.Diks, "Parallel 5-coloring of planar graphs", Proc. of the 14th ICALP, pp. 304-313, LNCS, Vol. 267, Springer-Verlag.

[5] H.Jung, K.Mehlhorn, "Parallel Algorithms for Computing Maximal Independent Sets in Trees and for Updating Minimum Spanning Trees", Information Processing Letters 27, pp. 227-236, April 1988.

[6] A.Itai, C.H.Papadimitriou, J.L. Szwarcfiter, "Hamilton Paths in Grid Graphs", SIAM Jour. on Comp., Vol.11, No.4, November 1982, pp.676-686.

[7] D.E.Knuth, "The Art of Computer Programming", Vol.1, Fundamental Algorithms, 2nd ed. Addison-Wesley, 1973.

[8] P.Klein, J.H.Reif, "An Efficient Parallel Algorithm for Planarity", Proc. 27th IEEE Symp. on FOCS, 1986, pp.465-477.

[9] V.Ramachandran, J.H.Reif, "An Optimal Parallel Algorithm for Graph Planarity", Proc. 30th IEEE Symp. on FOCS, 1989, pp.282-287.

[10] B.Schieber, U.Viskin, "On Finding Lowest Common Ancestors: Simplification and Parallelization", Proc. of 3rd Aegean Workshop on Computing (AWOC 88), Corfu, Greece, June/July 1988, pp. 111-123, LNCS 319, ed. J.H Reif, Spriger-Verlag.

[11] R.E.Tarjan, U.Vishkin, "An Efficient Parallel Biconnectivity Algorithm", SIAM Jour. on Comp., Vol.14, No.4, November 1985, pp.862-874.

[12] J.C.Wyllie, "The Complexity of Parallel Computation", Ph.D. Thesis, TR 79-387, Dept of Computer Science, Cornell University, Ithaca, NY, 1979.

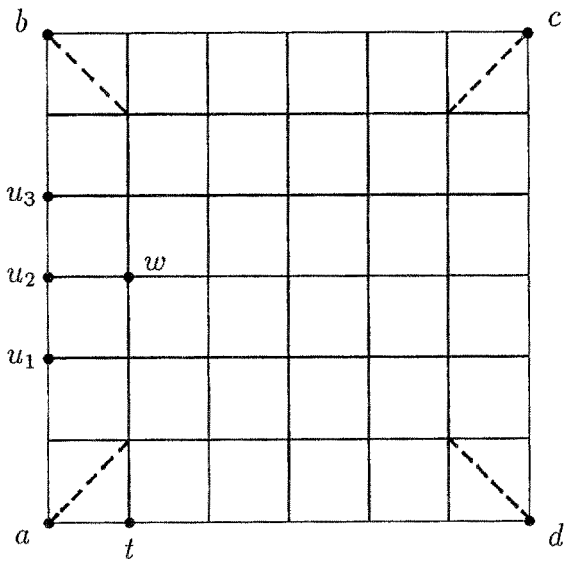

Figure 1: A rectangular grid graph. $(a_x, a_y) = (1, 1)$

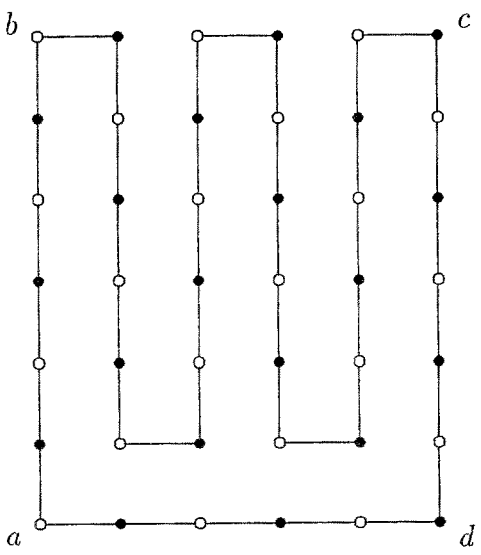

Figure 2: A Hamiltonian cycle

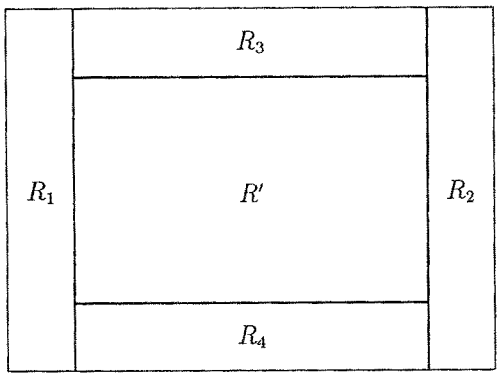

Figure 3: Reduction

Figure 4: Merging of a Hamiltonian cycle with a Hamiltonian path

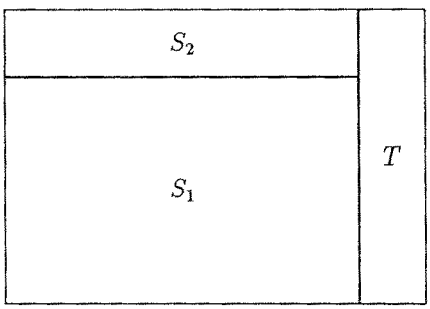

Figure 5: $R'(m_1, n_1)$

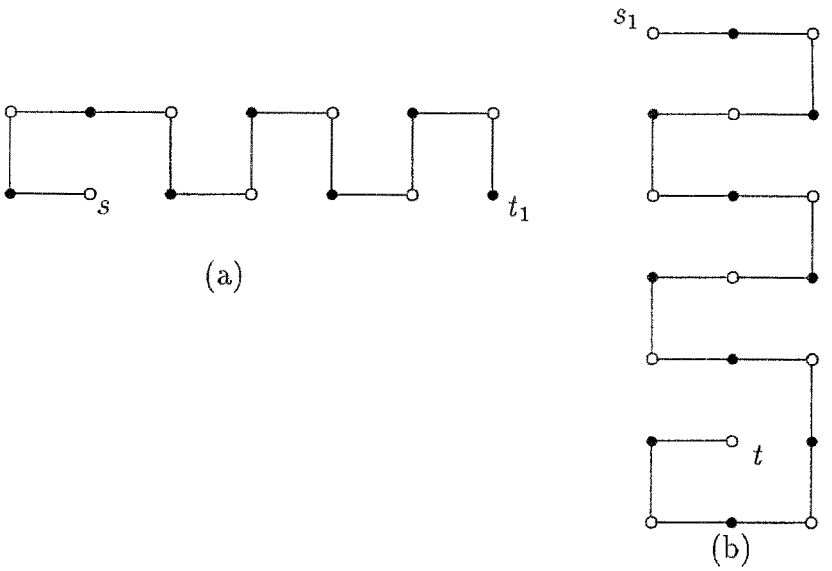

Figure 6

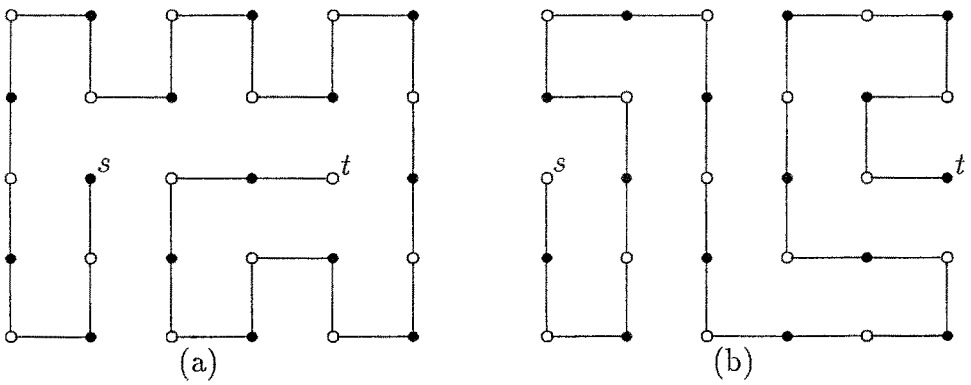

Figure 7: Case P2.1. $s_y = 3$, $t_y = 3$

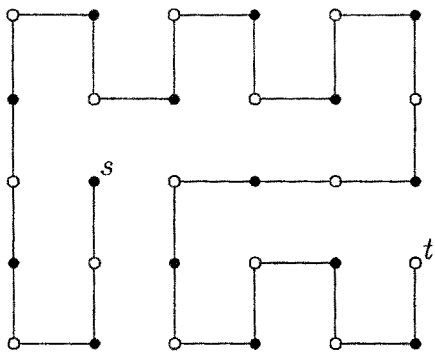

Figure 8: Case P2.2. $s_y = 3$, $t_y = 2$. (One case. The other cases are similar.)