OPTIMAL PARALLEL ALGORITHMS FOR DIRECT DOMINANCE PROBLEMS

Amitava Datta, Anil Maheshwari and
Jörg-Rüdiger Sack

TR-229, OCTOBER 1993

School of Computer Science, Carleton University
Ottawa, Canada, KIS 5B6

Optimal Parallel Algorithms for Direct Dominance Problems*

Amitava Datta
Max-Planck Institut für Informatik
Im Stadtwald, W-6600 Saarbrücken, Germany
datta@mpi-sb.mpg.de

Anil Maheshwari
Computer Systems and Communications Group
Tata Institute of Fundamental Research
Homi Bhabha Road, Bombay 400 005, India
manil@tifrvax.tifr.res.in

Jörg-Rüdiger Sack†
School of Computer Science
Carleton University
Ottawa, Ontario K1S 5B6, Canada
sack@scs.carleton.ca.

Abstract

We present optimal parallel solutions to direct dominance problems for planar point sets and provide an application. The algorithms presented here are deterministic and designed to run on the concurrent read exclusive write parallel random-access machine (CREW PRAM). In particular, we provide algorithms for counting the number of points that are directly dominated by each point of an n-point set and for reporting these point sets. The counting algorithm runs in $O(\log n)$ time using $O(n)$ processors; the reporting algorithm runs in $O(\log n)$ time using $O(n + k/\log n)$ processors, where k is the size of the output. The algorithms are therefore optimal. As an application of our results, we present an algorithm for the maximum empty rectangle problem, which is work optimal in the expected case.

1 Introduction

Let $P = \{p_1, p_2, ..., p_n\}$ be a planar point set of n distinct points $p_i=(x_i, y_i)$, $i=1,...,n$. A point p_i is said to *dominate* a point p_j, if $x_i \geq x_j$ and $y_i \geq y_j$ and $i \neq j$. The dominance problem is to enumerate all dominances of a given point set.

Dominance problems arise naturally in a variety of applications and they are directly related to well studied geometric and non-geometric problems. These problems include: range searching,

*This research was supported in part by the ESPRIT Basic Research Actions Program, under contract No. 7141 (project ALCOM II). A preliminary version of this paper is appearing in the Proceedings of the First European Symposium on Algorithms, 1993

†This research was in part supported by Natural Sciences and Engineering Council of Canada.

finding maximal elements and maximal layers, computing a largest area empty rectangle in a point set, determining the longest common sequence between two strings, and interval/rectangle intersection problems (see e.g. [22]). The 2-dimensional dominance problem has been shown to be equivalent to the interval enclosure problem which asks for all enclosures of a given interval set [16]. The d-dimensional dominance problem is equivalent to the enclosure problem for planar d-gons, where the corresponding sides are parallel; as was also shown in [16]. The enclosure problem and hence the dominance problem has several practical applications in computer aided design systems for VLSI circuits [16].

Preparata and Shamos [22] presented optimal sequential algorithms for counting and reporting the dominances for each point of the set P, running in $O(n \log n)$ and $O(n \log n + k)$ time, respectively, where k is the total number of dominance pairs. Some dominance problems have previously been studied for the CREW-PRAM (for details on this model see, e.g., [17, 18]). These problems include the *two-set dominance problem.* In counting mode of this problem, given two point sets A and B, all pairs (a, b) are to be counted where $a \in A$ dominates $b \in B$. In the reporting mode, all dominance pairs are to be enumerated. The two-set dominance counting problem was solved by Atallah et al. [3] in optimal $O(\log n)$ time using $O(n)$ processors, where n is the total number of points in the given sets. The reporting problem was solved by Goodrich [15] in $O(\log n)$ time using $O(n/\log n + k)$ processors, where k is the total number of dominance pairs.

From this discussion it follows that the complexities of sequential and parallel algorithms for dominance problems depend strongly on the number of dominances which, in the worst case, is $\Omega(n^2)$. The dominance relation is transitive and typically many dominances in the point set can be derived from others. For example, in the 1-dimensional case there are $\Omega(n^2)$ dominance pairs of which only a linear number are required. Hence, a set of dominances may contain many redundant elements. This motivates considering the problem of counting and reporting only those dominance pairs which are non-redundant.

We now define non-redundant dominance pairs, called *direct dominance pairs*, for a planar point set P. A point $p_i \in P$ *directly dominates* a point $p_j \in P$ if p_i dominates p_j and no other point $p_k \in P$ dominates p_j and is itself dominated by p_i. The pair (p_i, p_j) is referred to as *direct dominance pair.* It is easily seen that the dominance relation is the transitive closure of the direct dominance relation. The set of points which are directly dominated by p_i is denoted by *DOM(p_i).* The *direct dominance counting* problem is to compute the numbers $|DOM(p_i)|$ for every point $p_i \in P$; where $||$ denotes the cardinality of a set. The *direct dominance reporting* problem is to report the sets $DOM(p_i)$, i.e., to list the elements, for every point $p_i \in P$. Some points in P are not dominated by any point; these points are called *maximal*, and the set of all maximal points is denoted by *MAX(P).* Maximal points play a role when computing the convex hulls of point sets (see [22] for a discussion). Overmars and Wood have studied the direct dominance in the context of rectangular visibility [21]. Gewali, Keil and Ntafos use rectangular visibility for solving a covering problem [14].

In the sequential model of computation, the direct dominance counting and reporting problems have tight $\Theta(n \log n)$ and $\Theta(n \log n + k)$ time bounds, respectively, where k is the total number of direct dominance pairs in P [16]. The value of k is at least $n - 1$, but can be as large as $\Omega(n^2)$ (the reader may wish to construct a simple example illustrating this).

In the parallel model of computation, Chen and Friesen [9] presented a CREW PRAM-algorithm for solving the direct dominance reporting problem which takes $O(\log n + j)$ time and uses $O(n)$ processors, where j denotes the maximum of the number of direct dominances

reported by a single point in the set. Since one point can directly dominate all other points in the set, in the worst case the runtime of their algorithm is $\Omega(n)$. Their result left the question open whether a sublinear time algorithm is possible using linear number of processors.

In this paper we settle this question affirmatively by presenting optimal parallel algorithms for both direct dominance problems, i.e., the counting and reporting problem. We apply our results to the design of an algorithm for computing the maximum empty rectangle inside a planar point set. Our algorithm is work optimal in the expected case. All our parallel algorithms are designed to run on the CREW-PRAM variety of parallel model of computation. The direct dominance counting algorithm runs in optimal $O(\log n)$ time and uses $O(n)$ processors. The direct dominance reporting algorithm runs in $O(\log n)$ time and uses $O(n + k/\log n)$ processors, where k is the size of the output. The total work, i.e., the processor-time product is $O(n \log n + k)$, which matches the tight sequential bound known for this problem.

Our algorithms use an output sensitive number of processors. The output size k is not known in advance. We compute k on the fly and allocate the required number of processors in the spirit of [15]. The processors are 'spawned' depending on k. When new processors are allocated, a global array of pointers is created. A processor knows the exact location it starts working from by accessing the array. For more details on this model of computation the reader is referred to Goodrich [15].

We give a brief sketch of the underlying ideas. Our algorithms for dominance problems compute a data structure, called *maximal forest data structure*. It is a balanced binary tree T, consisting of n leaves; each leaf contains a point of P. The points are sorted by x-coordinate (in increasing order). Let v and w be left and right child, respectively of a node u of T; let T_v and T_w be the subtrees rooted at v and w, respectively. Each node v of T consists of a *maximal forest*, $MF(v)$, on the set of points in T_v. $MF(v)$ is a directed forest, where the points in T_v are nodes; two nodes are connected by a directed edge if the corresponding points satisfy a particular order relation.

We show that for any point $p_i \in T_w$, the points it directly dominates in T_v, are precisely the points located on a unique path in $MF(v)$. Thus, the problem of computing the direct dominances of p_i in T_v is reduced to that of locating a pair of nodes in $MF(v)$ and reporting the unique path between them. We solve the problem of locating the appropriate pair of nodes of $MF(v)$ by exploiting the structural properties of the data structure. We report the path between a pair of nodes in $MF(v)$ by providing an optimal parallel solution to a more general path problem on trees.

Our overall strategy therefore consists of traversing T for each point $p_i \in P$, from the leaf node containing p_i to the root of T. At each internal node of T in the path, we compute the points directly dominated by p_i by locating and reporting unique path between the appropriate nodes of a maximal forest.

We make use of the following tools from the area of parallel computing: Parallel merge-sort [7], parallel prefix, list ranking and doubling, described, e.g., in [17, 18], and operations on tree including lowest common ancestors [23], Eulerian tours [24], centroid levels of nodes [10].

The remainder of the paper is organized as follows. In Section 2 we state some preliminary results. In Section 3 we introduce a parallel data structure and state several of its properties; our computations of direct dominances are based on these. In Section 4 we describe and analyze the algorithms for the direct dominance counting and reporting problems. In Section 5 we discuss, as an application of our results, an algorithm for solving the maximum empty rectangle problem.

2 Preliminaries

Let p_i be a point in a planar point set P. We say that a point set is x-sorted (or y-sorted) if the points in the set are sorted by their x-coordinates (or y-coordinates). We define $Window(p_i)$ to be the set $\{p_j | p_j \in P \text{ and } x_j > x_i, y_j < y_i\}$ which corresponds to the lower right quadrant of an orthogonal coordinate system centered at p_i. We denote by $MAX(Window(p_i))$ the points of P which are maximal in $Window(p_i)$. We define a data structure, denoted by $MF(P)$, on P. The nodes of $MF(P)$ are the points of P. If no ambiguity arises we will not distinguish between a node representing a point and the point itself. The parent of p_i is the point in $MAX(Window(p_i))$ having the highest y-coordinate. (Ties are broken by chosing the point with least x-coordinate among all points with highest y-coordinate.) In the following, we state properties of this data structure as they are relevant in the context of this paper.

Lemma 2.1 *For input point set P, $MF(P)$ is a directed forest.*

Proof: Follows from the definition of $MF(P)$. □

We refer to the directed forest $MF(P)$ as the *Maximal Forest* for the set P.

Corollary 2.2 *If p_i, p_j and p_k are three points for which $x_i < x_j < x_k$, $y_i > y_j > y_k$, and there are directed paths from p_i to p_j and from p_j to p_k, then the directed path from p_i to p_k visits p_j.*

Suppose all points of P are placed into leaf positions of a complete binary tree T so that the points are x-sorted (increasing from left to right). Let u be a node in T whose left and right children are called v and w, respectively. The subtrees rooted at v and w are referred to as $T(v)$ and $T(w)$, respectively. Let p_i be a point in $T(w)$. We wish to determine those points of $T(v)$ which are directly dominated by p_i. Let p_j be the point having the highest y-coordinate among the points in $T(w)$ which are directly dominated by p_i. (If there are several such points we choose the one with the least x-coordinate among them.) Define $H(p_i)$ and $H(p_j)$ to be horizontal lines through p_i and p_j, respectively. Let $Strip(p_i, p_j)$ be the set of points in $T(v)$ whose y coordinate lies between $H(p_i)$ and $H(p_j)$.

Lemma 2.3 *A point q of $T(v)$ is directly dominated by p_i if and only if it belongs to the set $MAX(Strip(p_i, p_j))$.*

Proof: Assume that $q \in v$ is a point directly dominated by p_i. Let p_j be the point defined above. We show that (i) $q \in Strip(p_i, p_j)$ and (ii) q is maximal in $Strip(p_i, p_j)$.
(i) The choice of p_j and the assumption imply that q lies below p_i and above p_j. Therefore $q \in Strip(p_i, p_j)$.
(ii) Since p_i directly dominates q no point in v between $H(p_i)$ and $H(P_j)$ dominates q. Thus q is maximal.

The other direction is proven as follows. Since q is maximal in $MAX(Strip(p_i, p_j))$ no point in v dominates q. Since q lies in $Strip(p_i, p_j)$ the point p_i directly dominates q. □

Consider now the maximal forest at v, i.e., $MF(v)$. Let $below(p_i)$ and $below(p_j)$ be the points in $MF(v)$ just below the lines $H(p_i)$ and $H(p_j)$, respectively. We denote the *lowest common ancestor* of $below(p_i)$ and $below(p_j)$ in $MF(v)$ by $lca(below(p_i), below(p_j))$.

Assume that $below(p_i)$ and $below(p_j)$ are in the same connected component in $MF(v)$. Then denote by p_l the point located just before $lca(below(p_i), below(p_j))$ on the path from $below(p_i)$ to $lca(below(p_i), below(p_j))$ in $MF(v)$. In case that $below(p_i)$ and $below(p_j)$ are in the different components in $MF(v)$, p_l is the root of the component of $MF(v)$ containing $below(p_i)$.

Lemma 2.4 *The points of the set $MAX(Strip(p_i, p_j))$ are the vertices of the directed path in $MF(v)$ from $below(p_i)$ to p_l.*

Proof: Consider the case when $below(p_i)$ and $below(p_j)$ are in the same connected component $MF_i(v)$ of $MF(v)$. Suppose $lca(below(p_i), below(p_j))$ is the node p_k (Figure 1). Observe that p_k is below $H(p_j)$. We show that the points directly dominated by p_i in the subtree rooted at v are the points from $below(p_i)$ to p_l (including these two points). First we show that p_l is above $H(p_j)$. We prove this by a contradiction. Therefore, assume that p_l is below $H(p_j)$. There are two cases depending on whether p_l is either higher or lower than $below(p_j)$. The first case contradicts the definition of $below(p_j)$. Consider therefore the second case. Corollary 2.2, imply the existence of a directed path from $below(p_j)$ to p_k; this path must visit p_l. Furthermore, p_l is on the directed path from $below(p_i)$ to p_k. This implies that p_l is the lowest common ancestor of $below(p_i)$ and $below(p_j)$, which is a contradiction.

The case that $below(p_i)$ and $below(p_j)$ are in different components can be proven analogously. □

3 Parallel Construction of the Maximal Forest Data Structure

In this section we describe the parallel construction of the maximal forest data structure and state some of its properties. The data structure which represents the maximal forest for the given input set will be instrumental in the design of the direct dominance counting and reporting algorithms presented in the following section. The data structure also stores additional size information at nodes and is preprocessed for answering the lowest common ancestor queries. The construction consists of four phases, as described in Algorithm 1; each phase is discussed in detail in the following. For simplicity we assume that the number of points in the input set P is $n = 2^k$, for some $k \geq 1$.

Phase 1 : Sort the points in P by increasing x-coordinate. Store them in the leaves of a complete binary tree T in order, from left to right. This step can be performed in $O(\log n)$ time using n processors by applying Cole's algorithm [7].

Phase 2 : This phase consists of the following steps and is based on Cole's parallel-merge sort algorithm [7]. Sort the points according to decreasing y-coordinate. At each internal node u of T, maintain a y-sorted list of the points in $T(u)$. Compute the maximal forest $MF(u)$ of u for the points in $T(u)$. In addition, establish cross-ranking pointers (as described below) from a child node to its parent in T to facilitate search in Phase 4.

We state the generic merge step. Let v and w be the left and the right child, respectively, of the node u in T. Assume that the maximal forests for the nodes v and w have been computed. We describe the procedure to compute $MF(u)$ from $MF(v)$ and $MF(w)$. Denote by $Y(u)$, $Y(v)$ and $Y(w)$ the array of elements corresponding to the nodes u,v and w, respectively. The arrays are sorted by decreasing y-coordinates of the points. Note that the parent pointers of

1. Place the elements of the point set P into the leaf positions of a complete binary tree T so that the leaves are x-sorted (increasing from left to right).
2. Perform a bottom-up tree-sort of the points by increasing y-coordinate and compute, for each node u of T, the maximal forest $MF(u)$ of the set of points in $T(u)$.
3. For each node $u \in T$ and each $p_i \in MF(u)$, compute the number of nodes located on the directed path from p_i to the root of the component of $MF(u)$ to which p_i belongs.
4. For each node $u \in T$, preprocess the maximal forest $MF(u)$ for answering lowest common ancestor queries.

Algorithm 1: The main steps in constructing the data structure

the elements of $MF(u)$ which are also in $MF(w)$ remain unchanged during the merge phase. Let $p_i \in Y(v)$ and denote its rank in $Y(w)$ by $rank(p_i)$. Let $succ(p_i)$ denote the parent of p_i in $MF(v)$. Observe that if $rank(p_i) = rank(succ(p_i))$, then both p_i and $succ(p_i)$ have been ranked between the same pair of elements in $Y(w)$. Since p_i has a higher y-coordinate than $succ(p_i)$, the successor pointer of p_i remains unchanged. If, however, $rank(p_i) > rank(succ(p_i))$, then at least one point in $Y(w)$ exists which lies between p_i and $succ(p_i)$. The highest such point is called p_j; p_j becomes the new successor of p_i in the merged set. It is easy to see that the above computation can be performed in $O(\log n)$ time using $O(n)$ processors for the entire tree.

During this phase a pointer is established from an element $p_i \in Y(u)$: in case $p_i \notin Y(v)$ the pointer points to the element just below it in the array $Y(v)$. Otherwise, the pointer points to p_i. Analogous links are created between the elements of $Y(u)$ and $Y(w)$. Recall that the algorithm of Cole [7] cross ranks the elements in $Y(v)$ and $Y(w)$ through the elements of $Y(u)$. In other words, a point $p_i \in Y(v)$ is ranked in $Y(u)$ and each element of $Y(u)$ is ranked in $Y(w)$. So, in effect the rank of p_i in $Y(w)$ is also known. In Cole's algorithm the pointers from $Y(v)$ to $Y(u)$ and from $Y(u)$ to $Y(w)$ are not maintained throughout the execution of the algorithm. Our algorithm will maintain these pointers. These pointers are used to get, in $O(1)$ time, from an element $p_i \in Y(u)$ to the corresponding element in $Y(v)$ (or $Y(w)$), or to the element immediately below p_i in $Y(v)$ (or $Y(w)$). We call these pointers from $p_i \in Y(u)$ to $Y(v)$ and to $Y(w)$ the $l\_link(p_i)$ and $r\_link(p_i)$, respectively. There are only two pointers per element of $Y(u)$. Thus the space requirement is linear in the size of the array $Y(u)$. This concludes the description of Phase 2 of the algorithm. It can be seen that this phase requires $O(\log n)$ time using $O(n)$ processors.

Phase 3 : Consider again a node u of T and its maximal forest $MF(u)$. When Phase 2 is completed, every point $p_i \in u$ knows its successor in $MF(u)$. We want to compute the number of nodes, denoted by $count(p_i)$, of $MF(u)$ on the directed path from p_i. This is equivalent to computing the vertex level of each node and can be determined by the list ranking algorithm of [10], or [24]. Let n_i be the number of points in the subtree rooted at u. Assign $O(n_i/\log n)$ processors for the array $Y(u)$ at the node u. These processors can compute the rank of every element in $Y(u)$ in $O(\log n_i)$ time. This computation can be performed simultaneously at all the nodes of T and thus can be performed in $O(\log n)$ time. Note that each point p_i in T appears in $O(\log n)$ nodes along the path from the root to the leaf containing p_i. Therefore, the total

number of processors required is $O(n \log n / \log n)$, i.e., $O(n)$. Hence, Phase 3 can be performed in $O(\log n)$ time using $O(n)$ processors.

Phase 4 : In this phase, we invoke the algorithm of Schieber and Vishkin [23] to prepare all maximal forests $MF(u)$ for answering the *lowest common ancestor* queries. Each $MF(u)$ can be preprocessed in $O(\log |MF(u)|)$ time using $O(|MF(u)|/\log |MF(u)|)$ processors. This preprocessing is done simultaneously at all nodes of the tree in $O(\log n)$ time using $O(n)$ processors. After preprocessing, the lowest common ancestor of two points $p_i, p_j \in MF(u)$ can be computed in $O(1)$ time by a uniprocessor. In the following theorem we summarize the properties of the data structure so constructed.

Theorem 3.1 *In $O(\log n)$ time using $O(n)$ processors a data structure can be constructed on the tree T so that for any node $u \in T$, a uniprocessor can perform the following operations in $O(1)$ time*
(i) *Given a point $p_i \in u$, locate p_i, or the point just below p_i among the children of u.*
(ii) *Given a point $p_i \in u$, compute the number of points on the path from p_i to the root of the component of $MF(u)$ containing p_i.*
(iii) *Given two points $p_i, p_j \in MF(u)$, compute the lowest common ancestor of p_i and p_j.*

4 Algorithms for the Direct Dominance Problems

In this section we present optimal parallel algorithms for the direct dominance counting and reporting problems. To solve these problems we use the data structure developed in the previous section. First we discuss the algorithm for counting the number of elements in $DOM(p_i)$, for all $p_i \in P$.

4.1 Direct Dominance Counting Problem

Consider the path of p_i in T, denoted by $path(p_i)$, from a leaf node containing p_i to the root of T. Our algorithm for the direct dominance counting problem traverses $path(p_i)$ and counts the direct dominances of p_i at each node encountered along the path. Let u be a node of T encountered along $path(p_i)$. Two cases arise depending on whether the left child v of u belongs to $path(p_i)$, or the right child w belongs to $path(p_i)$.

First, assume that $v \in path(p_i)$. The point p_i does not dominate any point in the subtree rooted at w since all points in this subtree have a larger x-coordinate than p_i. So the set of points which p_i directly dominates in $T(u)$ equals the set of points it directly dominates in $T(v)$.

Now consider the case that $w \in path(p_i)$. The point p_i may directly dominate several points of $T(v)$. Let p_j be the point having the largest y-coordinate among the set of points directly dominated by p_i in $T(w)$. Let $below(p_i)$ and $below(p_j)$ be defined as in Lemma 2.4. The points $below(p_i)$ and $below(p_j)$ are identified among the points in $T(v)$ by using l_link pointers created in Phase 2 of Algorithm 1. This is done as follows: Using Lemma 2.4 the number of points is determined which are directly dominated by p_i in $T(v)$; which is done as follows. First compute the lowest common ancestor node of $below(p_i)$ and $below(p_j)$ in $MF(v)$ by the algorithm of Schieber and Vishkin [23]. Let $lca(below(p_i), below(p_j))$ be the lowest common ancestor node. Using the preprocessing of Phase 4 of Algorithm 1, compute the number of nodes between $below(p_i)$ and $lca(below(p_i), below(p_j))$. Hence for each node u in $path(p_i)$, we can report the number of nodes directly dominated by p_i. We summarize the results in the following theorem.

Theorem 4.1 *The direct dominance counting problem can be solved in optimal $O(\log n)$ time using $O(n)$ processors and $O(n \log n)$ space on the CREW PRAM.*

Proof: Any point directly dominated by p_i has x-coordinate less than p_i. All such points appear in the left child of the nodes along $path(p_i)$. By counting all the direct dominances in these nodes we obtain the total count $|DOM(p_i)|$. Lemmas 2.3 and 2.4 ensure that no point is missed which is directly dominated by p_i at such a node.

Now we analyze the complexity of the algorithm. Since $path(p_i)$ for any point $p_i \in P$ has $O(\log n)$ nodes, a single processor can compute $|DOM(p_i)|$ in $O(\log n)$ time. Hence for all points p_i, we can report $|DOM(p_i)|$ in $O(\log n)$ time using $O(n)$ processors. $O(n \log n)$ space is taken as each point appears in $O(\log n)$ nodes along $path(p_i)$. □

Corollary 4.2 *Given a set of n points P in the plane, a data structure can be computed in $O(\log n)$ time using $O(n)$ processors and $O(n \log n)$ space on the CREW PRAM, such that a processor can report in $O(\log n)$ time the number of points directly dominated in P by a query point.*

4.2 Direct Dominance Reporting Problem

In this section we present a parallel algorithm for reporting the sets $DOM(p_i)$ for all $p_i \in P$, i.e., we solve the direct dominance reporting problem. As in the algorithm for counting direct dominances, we first compute $path(p_i)$. Let u and w be two nodes on $path(p_i)$. Assume that the set of points have been reported which are directly dominated by p_i among the points in $T(w)$. From Lemma 2.4 it follows that the set of points in $T(v)$ which are directly dominated by p_i are the vertices on the path from $below(p_i)$ to the point p_l (i.e. the point just above $lca(below(p_i), below(p_j))$) in $MF(v)$. Thus the problem reduces to reporting this path in $MF(v)$. In the following we solve a more general problem on reporting paths in binary trees efficiently. The direct dominance reporting problem can be solved using the solution to the general problem.

A parallel algorithm (see Algorithm 2) is presented next for preprocessing an n-node rooted binary tree B such that the path between the two query nodes a and b in B can be reported efficiently. Let $size(v)$ be the number of vertices in the subtree rooted at v. The *centroid level* of a vertex v is given by $clevel(v) = \lceil \log_2(size(v)) \rceil$ [10]. Observe that for any node $v \in B$, there is at most one child of v which has the same centroid level as v.

Using Algorithm 2, the vertices on the path, $path(a, b)$, are reported between two query nodes a and b in B; this is done as follows. First compute the lowest common ancestor node, c, of a and b in B. Now the problem reduces to reporting the paths $path(a, c)$ and $path(b, c)$. For simplicity assume that b is an ancestor of a. If $clevel(a) = clevel(b)$ then both a and b belong to the same array and the vertices on $path(a, b)$ are the elements in the array between a and b. These elements can be reported in $O(\log n)$ time using $O(max\{1, |path(a, b)|/\log n\})$ processors.

Consider now the case when $clevel(a) \neq clevel(b)$. Note that there are at most $O(\log n)$ distinct centroid levels on any $path(a, b)$. Hence a processor can identify the arrays which contain the vertices along $path(a, b)$ in $O(\log n)$ time. Using the pointers computed in Steps 3 and 4 of Algorithm 2, the two indices in each of these arrays can be located such that the elements between them are in $path(a, b)$. The problem reduces therefore to reporting elements between these two indices in each array. Allocate a total of $O(max\{1, |path(a, b)|/\log n\})$ processors to report the elements on $path(a, b)$. Each processor reports at most $O(\log n)$ nodes. The ith

1. Using the algorithm of Schieber and Vishkin [23], preprocess B for answering the lowest common ancestor queries.
2. Compute Euler Tour of B and compute $size(v)$ for each node v of B by the algorithm of Tarjan and Vishkin [24].
3. For each node v of B, compute $clevel(v)$.
4. Partition the set of vertices of B into maximal disjoint paths, where each vertex v on a maximal path has the same value of $clevel(v)$. Store the vertices on a path in an array. This step can be performed by the algorithm of [10] or [17].
5. Set up pointers from nodes of B to the indices in the array where it appears and vice versa.
6. For each node v in B set up a pointer between v and the furthest ancestor of v in B which has the same centroid level as $clevel(v)$. If v and the parent of v have different centroid levels, then we set up a pointer from v to the parent of v. These pointers can be computed by the algorithm of [17].

Algorithm 2: Algorithm for preprocessing a binary tree for reporting paths

processor finds, in $O(\log n)$ time, the array and the indices of the elements in the array which it needs to report. Now each processor has sufficient information to report the elements in $path(a,b)$ in $O(\log n)$ time. Hence, using the above data structure, the vertices in $path(a,b)$ can be reported in $O(\log n)$ time using $O(max\{1, |path(a,b)|/\log n\})$ processors.

Theorem 4.3 *An $n-$node binary tree can be preprocessed in $O(\log n)$ time using $O(n)$ space and $O(n/\log n)$ processors, such that the vertices in the path between two query nodes a and b, $path(a,b)$, can be reported in $O(\log n)$ time using $O(max\{1, |path(a,b)|/\log n\})$ processors on the CREW PRAM.*

Proof: The correctness of the algorithm is easily observed. Now we analyze the complexity of the algorithm. Each step of Algorithm 2 runs in $O(\log n)$ time using $O(n/\log n)$ processors [10, 17, 23, 24]. There are at most $O(\log n)$ different centroid levels on any $path(a,b)$ in B. Hence the preprocessing algorithm runs within the claimed complexity bounds. The data structure required is of linear size, since each step of Algorithm 2 requires a linear space. □

Using the above theorem, we design an optimal parallel algorithm for reporting the set of points of P which are directly dominated by $p_i \in P$. Recall that the outstanding subproblem was to report the vertices on the path from $below(p_i)$ to the point p_l in $MF(v)$. If $MF(v)$ is a forest, then create a dummy root and this transforms $MF(v)$ to a tree. We may therefore assume that $MF(v)$ is a tree. Since $MF(v)$ is not necessarily a binary tree we apply the optimal algorithm described in [17] to transform $MF(v)$ into a binary tree. Using Theorem 4.3, we obtain an optimal procedure for reporting paths in $MF(v)$.

Finally we analyze the complexity of the above algorithm. The preprocessing of each $MF(v)$ takes $O(\log |MF(v)|)$ time using $O(|MF(v)|/\log |MF(v)|)$ processors (to efficiently report the

paths between two query nodes). Since each point of $p_i \in P$ is stored in $O(\log n)$ nodes, the total number of vertices summed over $MF(v)$ for all v is $O(n \log n)$. Hence the preprocessing algorithm runs in $O(\log n)$ time using $O(n)$ processors and requiring $O(n \log n)$ space. The algorithm for the direct dominance reporting requires an output sensitive number of processors. The algorithm computes the size of the output on the fly and allocates processors accordingly. This is similar to the scheme described in [15]. We summarize the results in the following theorem.

Theorem 4.4 *The direct dominance reporting problem for an n-point set can be solved in optimal $O(\log n)$ time using $O(n + K/\log n)$ processors and $O(n \log n)$ space, where $K = \Sigma_{p_i \in P} |DOM(p_i)|$, on the CREW PRAM.*

Corollary 4.5 *Given a set of n points P in the plane, a data structure can be computed in $O(\log n)$ time using $O(n)$ processors and $O(n \log n)$ space on the CREW PRAM, such that $O(max\{1, K/\log n\})$ processors can report in $O(\log n)$ time, the points directly dominated by a query point in P, where K is the number of points directly dominated by the query point in P.*

5 Fast Parallel Algorithm for the Maximum Empty Rectangle Problem

In this section we describe an algorithm for the *Maximum Empty Rectangle (MER)* problem stated next. Given a planar n-point set P inside a bounding isothetic rectangle BR, find the maximum area/perimeter isothetic rectangle R such that R lies completely inside BR and R does not include any point from the set P in its interior. This problem has received considerable attention in the literature; see, e.g., [2, 4, 5, 8, 12, 19, 20].

Aggarwal and Suri [2] have shown a sequential lower bound of $\Omega(n \log n)$ for this problem. The best known sequential algorithm runs in $O(n \log^2 n)$ time for the area problem and in $O(n \log n)$ time for the perimeter problem [2]. Several algorithms [5, 12, 20] solve the area/perimeter problem by enumerating all *restricted rectangles* (rectangles whose sides are supported by the sides of BR, or points from the set P and their interiors do not contain any point of P) in the point set P. The resulting time complexity for these algorithms is $O(n \log n + K)$, where K is the number of restricted rectangles for a given problem instance. It has been shown that K is $O(n^2)$ in the worst case, and $O(n \log n)$ in the expected case, respectively.

In the PRAM domain, the MER problem has been studied in [1, 13]. Datta and Krithivasan [13] have provided an EREW PRAM algorithm that runs in $O(\log n)$ time using $O(n^2/\log n)$ processors. Aggarwal et al. [1] have provided a CREW PRAM algorithm and it runs in $O(\log^2 n \log\log n)$ time using $O(n \log n/\log\log n)$ processors.

We present a parallel algorithm which matches the performance of the best known sequential algorithms (as derived in [5, 12, 20]). Our algorithm runs in $O(\log n)$ time using $O(n + K/\log n)$ processors on a CREW PRAM, where K is the number of restricted rectangles for a given problem instance. Hence, our algorithm is work optimal in the expected case. In the following we mention several relevant properties of maximum empty rectangles and restricted rectangles.

Property 1 *[19] A maximum empty rectangle is a restricted rectangle.*

A maximum empty rectangle can thus be found as follows. First enumerate all restricted rectangles in P and then find the one which has the maximum area/perimeter. In the following,

we discuss only the area problem since the perimeter problem can be solved analogously. To facilitate the discussion, we introduce some notation. For the bounding rectangle BR, the side with minimum (or maximum) x-coordinate is called $BR.left$ (respectively, $BR.right$). Similarly, the side with minimum (respectively, maximum) y-coordinate is called $BR.bottom$ (respectively, $BR.top$). The restricted rectangles (RR) are divided into two categories, as in [4, 19].
Type 1: The top side of RR is supported by $BR.top$.
Type 2: The top side of RR is supported by a point from the set P.

Property 2 *The number of restricted rectangles of Type 1 is* $2n + 1$.

Lemma 5.1 *The MER among the Type 1 RRs can be found in* $O(\log n)$ *time using* $O(n)$ *processors.*

Proof: First classify RRs of Type 1 into two categories. In the first category (*Type 1(i)*), the bottom side of a RR is supported by $BR.bottom$. In the second category (*Type 1(ii)*), the bottom side of a RR is supported by a point from the set P.

To compute *Type 1(i)* RRs, sort the points in P according to increasing x-coordinate in $O(\log n)$ time using $O(n)$ processors [7]. Associate one processor to each point in this sorted list. The processor associated with point p_i computes the area of the RR for which p_i is the left support. The right support of this RR is either the next point in the sorted list, or is $BR.right$. Hence the maximum area RR can be computed within the stated processor and time bounds.

Now we state a procedure for computing *Type 1(ii)* RRs. First sort the points by increasing x-coordinate and store them in an array X. Consider a point p_i in X. Let RR_i be the restricted rectangle for which p_i is the bottom support. The left support of RR_i is the point p_j, such that $x_j < x_i$, $y_j > y_i$ and the x-coordinate of p_j is the maximum among all such points. Similarly, the right support of RR_i is a point p_k, such that $x_k > x_i$, $y_k > y_i$ and the x-coordinate of p_k is the minimum among all such points. We can compute the points p_j and p_k corresponding to each p_i using the all nearest-smaller-values algorithm of Berkman et al. [6]. In the following we provide a simpler and thus more practical algorithm to perform the above step.

First preprocess the array X for answering *range maxima* queries, i.e., for any two indices i and j, $i \leq j$ report the point having the largest y-coordinate in the sub-array $X[i], \ldots, X[j]$. The preprocessing can be performed in $O(\log n)$ time using $O(n)$ processors and a query can be answered in $O(1)$ time by a single processor [17].

To find the left support for RR_i, assign a processor to p_i. The processor associated with p_i can find the left support by recursively performing a binary search in the subarray $X[1], \ldots, X[i-1]$. Suppose, during the binary search, a subarray $X[\alpha], \ldots, X[\beta], (\alpha < \beta < i)$ has been split into two subarrays $X[\alpha], \ldots, X[\lceil \frac{\alpha+\beta}{2} \rceil]$ and $X[\lceil \frac{\alpha+\beta}{2} \rceil + 1], \ldots, X[\beta]$. A range maxima query is performed in $X[\lceil \frac{\alpha+\beta}{2} \rceil + 1], \ldots, X[\beta]$ to find the point p_s having the maximum y-coordinate in this subarray. If $y_s < y_i$ (or $y_s > y_i$) the binary search is done recursively in the subarray $X[\alpha], \ldots, X[\lceil \frac{\alpha+\beta}{2} \rceil]$ (respectively, $X[\lceil \frac{\alpha+\beta}{2} \rceil + 1], \ldots, X[\beta]$). It is easy to see that at the end of the binary search, the left support for RR_i is found. Since each range maximum query takes $O(1)$ time, the overall time spent to perform this step is $O(\log n)$. Similarly, we can find the right support in $O(\log n)$ time. Hence, a MER among all such RRs can be computed within the stated processor and time bounds. □

We now discuss an algorithm for enumerating RRs of *Type 2*. For this define two kinds of *direct dominances*, viz. *left* and *right direct dominance*. If a point $p_i \in P$ directly dominates

a point $p_j \in P$, p_j is in the *left direct dominance set* of p_i. Consider two points p_i and p_k. If $x_i < x_k$ and $y_i > y_k$ and no other point p_m exists for which $x_i < x_m < x_k$ and $y_i > y_m > y_k$, then p_k is in the *right direct dominance set* of p_i. Left and right direct dominance set for each point of P can be computed by the algorithm presented in Section 4. Denote the left (or right) direct dominance of p_i by L_i (respectively, R_i). Assume that the elements in both L_i and R_i are sorted by decreasing y-coordinates (Figure 2). The following property holds.

Property 3 *If a point p_i is the top support of a RR of Type 2,*
(i) the left support is either $BR.left$ or an element of L_i,
(ii) the right support is either $BR.right$ or an element of R_i and,
(iii) the bottom support is an element of either L_i or R_i.
Moreover, if the left support is a point $p_j \in L_i$ and the right support is a point $p_k \in R_i$, the bottom support is either the point next to p_j in L_i or the point next to p_k in R_i.

Property 4 *A point $p_j \in L_i \cup R_i$ is the bottom support of exactly one RR.*

Proof: Without loss of generality assume that $p_j \in L_i$. The left support for this RR is either $BR.left$ or the point just above p_j in L_i. Similarly, the right support is the point just above p_j in R_i. So, in effect both the left and the right supports for such an RR are fixed. It is easy to see that any other choice of the supports contradicts the assumption that RR does not contain any point of P in its interior. □

Property 5 *The number of MERs with p_i as the top support is $|L_i \cup R_i|$.*

Algorithm 3 for enumerating the RRs of *Type 2* is based on Properties 3, 4 and 5. Let K_i denote the total number of elements in $L_i \cup R_i$. Let MER_i denote the maximum area empty rectangle of *Type 2* with p_i as the top support.

We now analyse the complexity of Algorithm 3. Step 1 takes time $O(\log n)$ using $O(max\{1, K_i/\log n\})$ processors. The cross ranking pointers in Step 2 can be computed in $O(\log n)$ time using $O(K_i/\log n)$ processors [17]. In Step 4, each processor can compute the MER in its group in $O(\log n)$ time. Step 5 can be executed in $O(\log n)$ time using $O(K_i/\log n)$ processors. Hence, the overall complexity of Algorithm 3 is $O(\log n)$ time using $O(max\{1, K_i/\log n\})$ processors. We summarize the result in the following lemma.

Lemma 5.2 *Among all RRs having point p_i as top support, the MER can be computed in $O(\log n)$ time using $O(K_i/\log n)$ processors, where $K_i = |L_i \cup R_i|$.*

Algorithm 3 is executed in parallel for all points in the set P. Hence, the overall complexity of the algorithm is $O(\log n)$ time using $O(K/\log n)$ processors, where $K = \Sigma K_i$. After this step, n candidates are obtained for being (global) MER. This (global) MER can then be found in $O(\log n)$ time using $O(n)$ processors. We summarize the result in the following theorem.

Theorem 5.3 *The maximum empty rectangle problem for a planar n-point set can be solved in $O(\log n)$ time using $O(n + K/\log n)$ processors on the CREW PRAM, where K is the number of restricted rectangles for a problem instance.*

Acknowledgement: The authors thank Kurt Mehlhorn for providing the excellent environment that allowed this research to be carried out.

1. For each point $p_i \in P$, compute the left and right direct dominance sets L_i and R_i by applying the algorithm stated in Section 4. Store the points in L_i and R_i sorted according to decreasing y-coordinate.

2. Establish cross-ranking pointers between the elements of the arrays L_i and R_i [17].

3. Partition the elements of L_i and R_i into groups of size $O(\log n)$ and associate one processor with each group.

4. The processor associated with a particular group enumerates the RRs taking each element of its group as bottom support. This can be done by using Property 4 and the cross-ranking information computed in Step 2. The processor also computes the overall MER for the elements in its group.

5. Compute MER_i out of $K_i/\log n$ MERs.

Algorithm 3: The main steps of the algorithm for enumerating RRs of Type 2 with p_i as the top support.

References

[1] A. Aggarwal, D. Kravets, J. Park and S. Sen. *Parallel searching in generalized Monge arrays with applications.* Proc. 2nd ACM Symp. on Parallel Algorithms and Architectures, 1990, pp. 259-268.

[2] A. Aggarwal and S. Suri. *Fast algorithms for computing the largest empty rectangle.* Proc. 3rd Annual ACM Symp. on Comp. Geom., 1987, pp. 278-290.

[3] M. Atallah, R. Cole and M. Goodrich. *Cascading divide-and-conquer : a technique for designing parallel algorithms.* SIAM J. Computing, **18** (1989), pp. 499-532.

[4] M. Atallah and G. Fredrickson. *A note on finding the maximum empty rectangle.* Discrete Applied Mathematics. **13** (1986) pp. 87-91.

[5] M. Atallah and S. R. Kosaraju. *An efficient algorithm for maxdominance, with applications.* Algorithmica 4 (1989) pp. 221-236.

[6] O. Berkman, B. Schieber and U. Vishkin, *Some doubly logarithmic optimal parallel algorithm based on finding all nearest smaller values*, Technical Report UMIACS-TR-88-79, University of Maryland, 1988.

[7] R. Cole. *Parallel merge sort.* SIAM J. Computing, **17**, (1988), pp. 770-785.

[8] B. Chazelle, R. Drysdale and D. T. Lee. *Computing the largest empty rectangle.* SIAM J. Computing **15** (1986), pp. 300-315.

[9] I. W. Chen and D. K. Friesen. *Parallel algorithms for some dominance problems based on a CREW PRAM.* Proc. 2nd International Symposium on Algorithms, LNCS 557, 1991, pp. 375-384.

[10] R. Cole and U. Vishkin. *The accelerated centroid decomposition technique for optimal parallel tree evaluation in logarithmic time.* Algorithmica, **3** (1988), pp. 329-346.

[11] R. Cole and U. Vishkin. *Approximate parallel scheduling, Part I: The basic technique with applications to optimal parallel list ranking in logarithmic time.* SIAM J. Computing, **17** (1988), pp. 128-142.

[12] A. Datta. *Efficient algorithms for the largest rectangle problem.* Information Sciences, **64** (1992), pp. 121-141.

[13] A. Datta and K. Krithivasan. *Efficient algorithms for the maximum empty rectangle problem in shared memory and other architectures.* Proc. 1990 International Conference on Parallel Processing, 1990, **III**, pp. 344-345.

[14] L. Gewali, M. Keil and S. Ntafos, *On Covering Orthogonal Polygons with Star-Shaped Polygons.* Information Sciences, **65** (1992), pp. 45-63.

[15] M. T. Goodrich. *Intersecting line segments in parallel with an output-sensitive number of processors.* SIAM J. Computing, **20**, (1991), pp. 737-755.

[16] R. Güting, O. Nurmi and T. Ottmann. *Fast algorithms for direct enclosures and direct dominances.* J. Algorithms **10** (1989), pp. 170-186.

[17] J. JáJá. *An Introduction to Parallel Algorithms.* Addison-Wesley, Reading, 1992.

[18] R. M. Karp and V. Ramachandran, *Parallel Algorithms for Shared-Memory Machines*, Handbook of Theoretical Computer Science, Ed. J. van Leeuwen, Vol 1, Elsevier Science Publishers B.V, 1990.

[19] A. Namaad, W. Hsu and D. T. Lee. *On maximum empty rectangle problem.* Discrete Applied Mathematics **8** (1984), pp. 267-277.

[20] M. Orlowski. *A new algorithm for the largest empty rectangle problem.* Algorithmica **5** (1990), pp. 65-73.

[21] M. Overmars and D. Wood. *On rectangular visibility.* J. of Algorithms **9** (1988), pp. 372-390.

[22] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction.* Springer-Verlag, New York, 1985.

[23] B. Schieber and U. Vishkin. *On finding lowest common ancestors: Simplification and Parallelization.* SIAM. J. Computing, **17** (1988), pp. 1253-1262.

[24] R. E. Tarjan and U. Vishkin. *An efficient parallel biconnectivity algorithm.* SIAM J. Computing, **14** (1985), pp. 862-874.

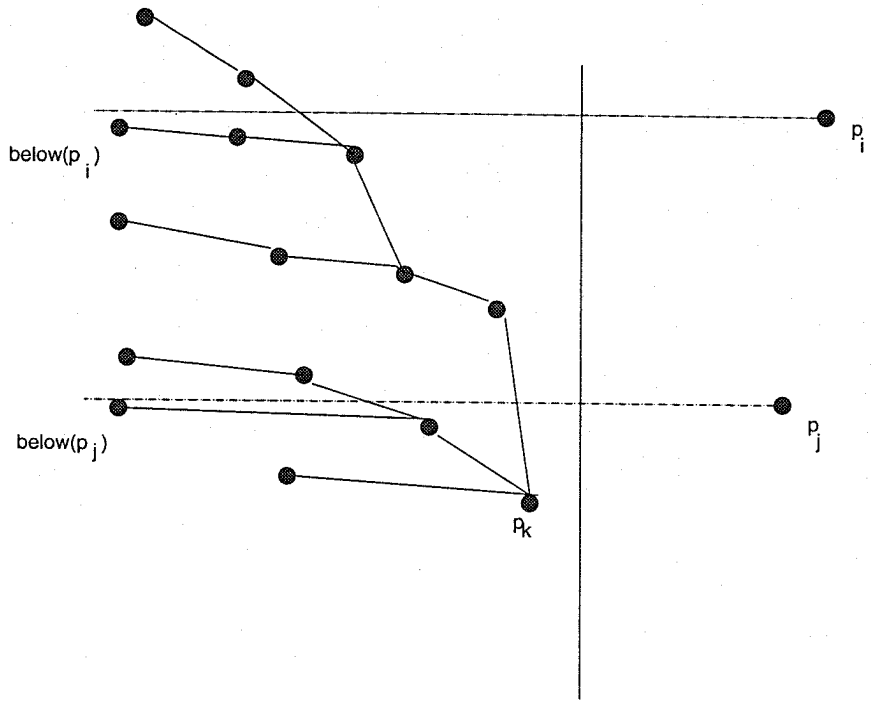

Figure 1: Illustration for Lemma 2.4

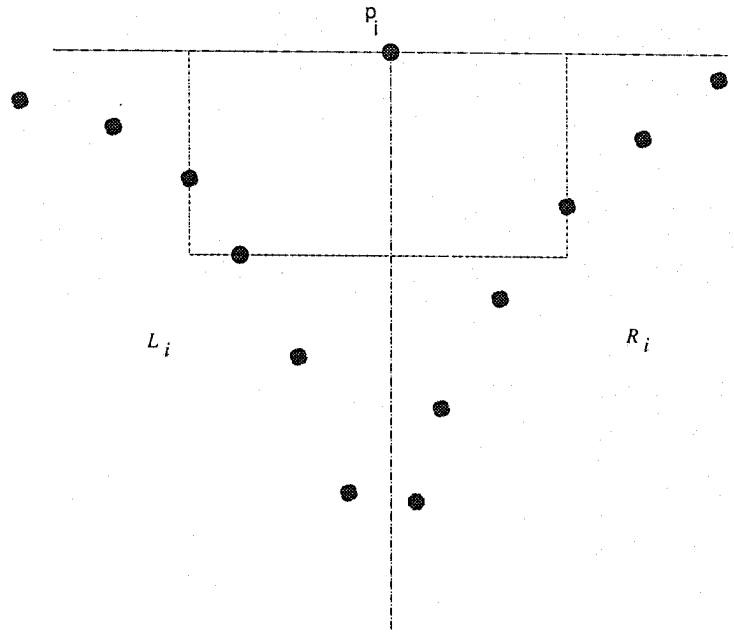

Figure 2: The left and right direct dominance sets for a point. A restricted rectangle is shown by dotted lines.

School of Computer Science, Carleton University
Recent Technical Reports

TR-189 **Probabilistic Estimation of Damage from Fire Spread**
Charles C. Colbourn, Louis D. Nel, T.B. Boffey and D.F. Yates, April 1991

TR-190 **Coordinators: A Mechanism for Monitoring and Controlling Interactions Between Groups of Objects**
Wilf R. LaLonde, Paul White, and Kevin McGuire, April 1991

TR-191 **Towards Decomposable, Reusable Smalltalk Windows**
Kevin McGuire, Paul White, and Wilf R. LaLonde, April 1991

TR-192 **PARASOL: A Simulator for Distributed and/or Parallel Systems**
John E. Neilson, May 1991

TR-193 **Realizing a Spatial Topological Data Model in a Relational Database Management System**
Ekow J. Otoo and M.M. Allam, August 1991

TR-194 **String Editing with Substitution, Insertion, Deletion, Squashing and Expansion Operations**
B John Oommen, September 1991

TR-195 **The Expressiveness of Silence: Optimal Algorithms for Synchronous Communication of Information**
Una-May O'Reilly and Nicola Santoro, October 1991

TR-196 **Lights, Walls and Bricks**
J. Czyzowicz, E. Rivera-Campo, N. Santoro, J. Urrutia and J. Zaks, October 1991

TR-197 **A Brief Survey of Art Gallery Problems in Integer Lattice Systems**
Evangelos Kranakis and Michel Pocchiola, November 1991

TR-198 **On Reconfigurability of Systolic Arrays**
Amiya Nayak, Nicola Santoro, and Richard Tan, November 1991

TR-199 **Constrained Tree Editing**
B. John Oommen and William Lee, December 1991

TR-200 **Industry and Academic Links in Local Economic Development: A Tale of Two Cities**
Helen Lawton Smith and Michael Atkinson, January 1992

TR-201 **Computational Geometry on Analog Neural Circuits**
Frank Dehne, Boris Flach, Jörg-Rüdiger Sack, Natana Valiveti, January 1992

TR-202 **Efficient Construction of Catastrophic Patterns for VLSI Reconfigurable Arrays**
Amiya Nayak, Linda Pagli, Nicola Santoro, February 1992

TR-203 **Numeric Similarity and Dissimilarity Measures Between Two Trees**
B. J. Oommen, K. Zhang and W. Lee, February 1992 (Revised July 1993)

TR-204 **Recognition of Catastrophic Faults in Reconfigurable Arrays with Arbitrary Link Redundancy**
Amiya Nayak, Linda Pagli, Nicola Santoro, March 1992

TR-205 **The Permutational Power of a Priority Queue**
M.D. Atkinson and Murali Thiyagarajah, April 1992

TR-206 **Enumeration Problems Relating to Dirichlet's Theorem**
Evangelos Kranakis and Michel Pocchiola, April 1992

TR-207 **Distributed Computing on Anonymous Hypercubes with Faulty Components**
Evangelos Kranakis and Nicola Santoro, April 1992

TR-208 **Fast Learning Automaton-Based Image Examination and Retrieval**
B. John Oommen and Chris Fothergill, June 1992

TR-209 **On Generating Random Intervals and Hyperrectangles**
Luc Devroye, Peter Epstein and Jörg-Rüdiger Sack, July 1992

TR-210 **Sorting Permutations with Networks of Stacks**
M.D. Atkinson, August 1992

TR-211 **Generating Triangulations at Random**
Peter Epstein and Jörg-Rüdiger Sack, August 1992

TR-212 **Algorithms for Asymptotically Optimal Contained Rectangles and Triangles**
Evangelos Kranakis and Emran Rafique, September 1992

TR-213 **Parallel Algorithms for Rectilinear Link Distance Problems**
Andrzej Lingas, Anil Maheshwari and Jörg-Rüdiger Sack, September 1992

TR-214 **Camera Placement in Integer Lattices**
Evangelos Kranakis and Michel Pocchiola, October 1992

TR-215 **Labeled Versus Unlabeled Distributed Cayley Networks**
Evangelos Kranakis and Danny Krizanc, November 1992

TR-216 **Scalable Parallel Geometric Algorithms for Coarse Grained Multicomputers**
Frank Dehne, Andreas Fabri and Andrew Rau-Chaplin, November 1992

TR-217 **Indexing on Spherical Surfaces Using Semi-Quadcodes**
Ekow J. Otoo and Hongwen Zhu, December 1992

TR-218 **A Time-Randomness Tradeoff for Selection in Parallel**
Danny Krizanc, February 1993

TR-219 **Three Algorithms for Selection on the Reconfigurable Mesh**
Dipak Pravin Doctor and Danny Krizanc, February 1993

TR-220 **On Multi-label Linear Interval Routing Schemes**
Evangelos Kranakis, Danny Krizanc, and S.S. Ravi, March 1993

TR-221 **Note on Systems of Polynomial Equations over Finite Fields**
Vincenzo Acciaro, March 1993

TR-222 **Time-Message Trade-Offs for the Weak Unison Problem**
Amos Israeli, Evangelos Kranakis, Danny Krizanc and Nicola Santoro, March 1993

TR-223 **Anonymous Wireless Rings**
Krzysztof Diks, Evangelos Kranakis, Adam Malinowski, and Andrzej Pelc, April 1993

TR-224 **A consistent model for noisy channels permitting arbitrarily distributed substitutions, insertions and deletions**
B.J. Oommen and R.L. Kashyap, June 1993

TR-225 **Mixture Decomposition for Distributions from the Exponential Family Using a Generalized Method of Moments**
S.T. Sum and B.J. Oommen, June 1993

TR-226 **Switching Models for Non-Stationary Random Environments**
B. John Oommen and Hassan Masum, July 1993

TR-227 **The Probability of Generating Some Common Families of Finite Groups**
Vincenzo Acciaro, September 1993

TR-228 **Power Roots of Polynomials over Arbitrary Fields**
Vincenzo Acciaro, September 1993

TR-229 **Optimal Parallel Algorithms for Direct Dominance Problems**
Amitava Datta, Anil Maheshwari and Jörg-Rüdiger Sack, October 1993

TR-230 **Uniform Generation of Forests of Restricted Height**
M.D. Atkinson and J.-R. Sack