1986

An Efficient Algorithm for Maxdominance, with Applications

Mikhail J. Atallah
Purdue University, mja@cs.purdue.edu

S. Rao Kosaraju

Report Number:

86-641

AN EFFICIENT ALGORITHM FOR MAXDOMINANCE, WITH APPLICATIONS

Mikhail J. Atallah
S. Rao Kosaraju

CSD-TR-641
November 1986
Revised May 1987

AN EFFICIENT ALGORITHM FOR MAXDOMINANCE, WITH APPLICATIONS

Mikhail J. Atallah†

S. Rao Kosaraju*

Dept. of Computer Science
Purdue University
West Lafayette, IN 47907

Abstract. Given a planar set S of n points, *maxdominance* problems consist of computing, for every $p \in S$, some function of the maxima of the subset of S that is dominated by p. A number of geometric and graph-theoretic problems can be formulated as maxdominance problems, including the problem of computing a minimum independent dominating set in a permutation graph, the related problem of finding the shortest maximal increasing subsequence, the problem of computing a maximum independent set in an overlap (and hence circle) graph, the problem of enumerating restricted empty rectangles, and the related problem of computing the largest empty rectangle. We give an algorithm for optimally solving a class of maxdominance problems. A straightforward application of our algorithm yields improved time bounds for the above-mentioned problems. The techniques used in the algorithm are of independent interest, and include a linear-time tree computation that is likely to arise in other contexts.

† This research was supported by the Office of Naval Research under Grants N00014-84-K-0502 and N00014-86-K-0689, and the National Science Foundation under Grant DCR-8451393, with matching funds from AT&T.

* This research was supported by the National Science Foundation under Grant DCR-856361

1. Introduction

A point p is said to *dominate* a point q iff $X(p) \geq X(q)$, $Y(p) \geq Y(q)$, and $p \neq q$, where $X(p)$ and $Y(p)$ respectively denote the x and y coordinates of point p. If S is a set of points and p is a point, we use $DOM_S(p)$ to denote the subset of points in S that are dominated by point p. A point of S is a *maximum in S* iff no other point of S dominates it. We use $MAX(S)$ to denote the set of maxima of S, listed by increasing x coordinates (and hence by decreasing y coordinates). We abbreviate $MAX(DOM_S(p))$ as $MD_S(p)$. A number of geometric and graph-theoretic problems can be formulated as one of the following two *maxdominance* problems **P1** and **P2** (problem **P2** being substantially more difficult than **P1**).

Problem P1. Given a set S of n points in the plane, compute $MD_S(p)$ for every $p \in S$.

We solve the above problem in $O(n\log n + t)$ time where t is the size of the output, i.e. $t = \sum_{p \in S} |MD_S(p))|$.

Problem P2. For a set S of points in the plane with a real weight $w(p)$ associated with every $p \in S$, the problem is to compute the *label* and *predecessor* of every point in S, where the *label* function is defined as follows:

$$label(p) = w(p) \quad \text{if } DOM_S(p) = \emptyset,$$

$$label(p) = w(p) + Min\{label(q) : q \in MD_S(p)\} \quad \text{otherwise.}$$

The *predecessor* of point p is any one of the points which gave p its label, i.e. it is a point $q \in MD_S(p)$ such that $label(p) = w(p) + label(q)$ (if $DOM_S(p) = \emptyset$ then p has no predecessor).

We solve problem **P2** in $O(n\log n)$ time and $O(n)$ space, which is optimal since sorting is a trivial special case of **P2**.

It is the algorithm for **P2** that is the main contribution of this paper (**P1** is solved by a much simplified version of the algorithm for **P2**).

The paper is organized as follows. Section 2 establishes some preliminary results, and Section 3 gives a result on tree computations which is needed in our solution to **P2** (it is also of independent interest). Section 4 gives our $O(n\log n)$ time, $O(n)$ space algorithm for problem **P2**. Section 5 gives an $O(n\log n+t)$ algorithm for problem **P1**. Section 6 lists problems for which improved complexity bounds follow from our results, and Section 7 concludes.

2. Preliminaries

Throughout this section, L and R are two planar sets of points separated by a vertical line and such that L is to the left of R; S denotes $L \cup R$. To simplify the exposition, we assume that no two points have same x coordinate (similarly for y coordinates).

Recall that in the list $MD_S(p)$, the points are in increasing x coordinate value. For every $p \in S$, $leader_S(p)$ denotes the leftmost (i.e. highest) point in $MD_S(p)$ (if $MD_S(p)=\varnothing$ then $leader_S(p)=\varnothing$). In Figure 1, $MD_R(p)=\{u,v,w\}$, $MD_S(p)=\{b,e,d,c,u,v,w\}$, $leader_R(p)=u$, and $leader_S(p)=b$.

For every $p \in R$, $Strip_L(p,R)$ denotes the points of L that are below p and above $leader_R(p)$; $Begin_L(p,R)$ and $End_L(p,R)$ denote the leftmost (i.e. highest) and rightmost (i.e. lowest) points on $MAX(Strip_L(p,R))$, respectively (if $Strip_L(p,R)=\varnothing$ then $Begin_L(p,R)=End_L(p,R)=\varnothing$). For example, in Figure 1, $Strip_L(p,R)=\{a,b,c,d,e,f\}$, $MAX(Strip_L(p,R))=\{b,e,d,c\}$, $Begin_L(p,R)=b$, and $End_L(p,R)=c$. Observe that for every $p \in R$, the list $MD_S(p)$ is the concatenation of $MAX(Strip_L(p,R))$ with $MD_R(p)$.

We define $G(S)$ as the directed acyclic graph whose vertex set is S and such that (p,q) is an edge in $G(S)$ iff there exists a point $w \in S$ such that q immediately follows p on the list $MD_S(w)$, in which case we say that edge (p,q) is *caused by* w. An edge may be caused by more than one point, but $G(S)$ has a single copy of such an edge. In Figure 1, edge (u,v) is in $G(S)$ and is caused by points k,p,g and h. Note also that (u,w) is not an edge of $G(S)$. Let $E(L,R)$ be the subset of edges of $G(S)$ that have both ends in L and are caused by at least one point in R. That is,

$E(L,R)=\{(p,q) : p \in L, q \in L, (p,q)$ is caused by some $w \in R\}$.

Observation 1. The graph $(L,E(L,R))$ is a forest.

Proof. A node in this graph has out-degree at most one. □

Note that for every $p \in R$, $MAX(Strip_L(p,R))$ is the path in the forest $(L,E(L,R))$ from $Begin_L(p,R)$ to $End_L(p,R)$.

Let $CROSS(L,R)$ be the subset of edges of $G(S)$ that have one endpoint in L and one in R.

Observation 2. $|CROSS(L,R)| \le |R|$.

Proof. An edge in $CROSS(L,R)$ can only be caused by a point in R. Moreover, a point in R can cause at most one edge in $CROSS(L,R)$. Thus $|CROSS(L,R)| \le |R|$. □

Note that if $p \in R$ causes the edge $(c,u) \in CROSS(L,R)$, then $c=End_L(p,R)$ (see Figure 1).

Two points p and q are *comparable* iff one of them dominates the other. A set of points forms a *chain* iff every two points in it are comparable. $MAXREV(S)$ denotes the subset of S such that $p \in MAXREV(S)$ iff no other point of S is both above p and to its left. We assume that the elements of $MAXREV(S)$ are listed by increasing x coordinates (and hence by increasing y coordinates, since they form a chain). In Figure 1, $MAXREV(R)=\{l,u,k\}$.

Lemma 1. Given the lists Q_L and Q_R containing the points of L and R, respectively, sorted by increasing y coordinates, $E(L,R)$ and $CROSS(L,R)$ can be computed in $O(|L|+|R|)$ time. In addition, for all $p \in R$, $Begin_L(p,R)$ and $End_L(p,R)$ can also be computed in $O(|L|+|R|)$ time.

Proof. Let $Q_R=(q_1, \cdots, q_{|R|})$, $Y(q_1) < \cdots < Y(q_{|R|})$. Initialize $E(L,R)$ and $CROSS(L,R)$ to $\emptyset$. We compute the edges in $E(L,R)$ by scanning the list Q_R, maintaining on a stack $STACK$ the $MAXREV$ of the subset of R encountered so far by the scan; i.e. when we are at q_i, $STACK$ contains the elements of $MAXREV(\{q_1, \cdots, q_i\})$ stored by increasing y coordinates. Note that q_i is the highest point in $\{q_1, \cdots, q_i\}$ and hence it belongs to $MAXREV(\{q_1, \cdots, q_i\})$ and is at the top of $STACK$. When the scan advances from q_i to q_{i+1}, we do the following: we add to $E(L,R)$ and $CROSS(L,R)$ the edges that are caused by q_{i+1} and are not caused by any of $\{q_1, \cdots, q_i\}$ (i.e. the "new" edges), update the contents of *STACK* so that it contains $MAXREV(\{q_1, \cdots, q_{i+1}\})$, and compute $Begin_L(q_{i+1},R)$ and $End_L(q_{i+1},R)$. The details are as follows.

(1) Obtain the elements of $Strip_L(q_1,R)$ in sorted order. This takes $O(|Strip_L(q_1,R)|)$ time by scanning Q_L until a point of L higher than q_1 is reached. (*Note.* Since $leader_R(q_1)=\varnothing$, $Strip_L(q_1,R)=DOM_L(q_1)$.) Compute $MAX(Strip_L(q_1,R))$; since the points in $Strip_L(q_1,R)$ are already sorted, this takes $O(|Strip_L(q_1,R)|)$ time [OV]. Add $|MAX(Strip_L(q_1,R))|-1$ edges to $E(L,R)$, one for each pair of adjacent points in $MAX(Strip_L(q_1,R))$; i.e. if q immediately follows p in $MAX(Strip_L(q_1,R))$ then we add edge (p,q) to $E(L,R)$. If $Strip_L(q_1,R)\neq\varnothing$ then set $Begin_L(q_1,R)$ and $End_L(q_1,R)$ to be the leftmost and rightmost points on $MAX(Strip_L(q_1,R))$, respectively. If $Strip_L(q_1,R)=\varnothing$ then set $Begin_L(q_1,R)$ and $End_L(q_1,R)$ to be $\varnothing$.

Set $i=1$ and repeat the following Steps (2)-(5) until $i>|R|$:

(2) Advance along Q_L until a point of L higher than q_{i+1} is reached. The sequence of points encountered, excluding the last one, yields the subset H of points in L that are above q_i and below q_{i+1}, sorted by their y components. Compute $MAX(H)$; since the points in H are already sorted, this takes $O(|H|)$ time [OV]. Add to $E(L,R)$ an edge for each consecutive pair of points in the list $MAX(H)$ (these edges of $E(L,R)$ are caused by q_{i+1} and not caused by any of $\{q_1, \cdots, q_i\}$). If q_{i+1} dominates q_i then go to Step (3), otherwise go to Step (4).

(3) Since q_{i+1} dominates q_i, q_i is $leader_R(q_{i+1})$ and therefore $H=Strip_L(q_{i+1},R)$ and all the new edges of $E(L,R)$ caused by q_{i+1} were already added in Step (2). If $H\neq\varnothing$ then set $Begin_L(q_{i+1},R)$ (resp. $End_L(q_{i+1},R)$) to be the leftmost (resp. rightmost) point on $MAX(H)$, then add to $CROSS(L,R)$ the edge $(End_L(q_{i+1},R),q_i)$. If $H=\varnothing$ then set $Begin_L(q_{i+1},R)=End_L(q_{i+1},R)=\varnothing$. Go to Step (5).

(4) Since q_{i+1} does not dominate q_i, q_{i+1} is above and to the left of q_i: Pop from *STACK* all the points that are below and to the right of q_{i+1}, and let $\beta_1, \cdots, \beta_k$ be the sequence of points so popped (see Figure 2). Note that $\beta_1=q_i$, and that the β_j's form a chain and are the top k

points on $MAXREV(\{q_1, \cdots, q_i\})$. Let U_0 denote $MAX(H)$, and let U_j denote $MAX(Strip_L(\beta_j, R))$ $(1 \le j \le k)$. Sub-step (4.1) below computes $Begin_L(q_{i+1}, R)$ and $End_L(q_{i+1}, R)$, while sub-step (4.2) finds any additional edges of $E(L, R)$ that are caused by q_{i+1} (for example, an edge between the rightmost point of U_j and the point immediately to its right on U_{j+1}). We do not need to add to $CROSS(L, R)$ the edge (if there is one) caused by q_{i+1}, because such an edge would also be caused by β_k and thus would already have been added when processing β_k.

(4.1) If $\bigcup_{j=0}^{k} U_j = \varnothing$ then set $Begin_L(q_{i+1}, R)$ and $End_L(q_{i+1}, R)$ to be $\varnothing$. Otherwise set $Begin_L(q_{i+1}, R)$ to be the highest point on the highest nonempty U_j $(0 \le j \le k)$, and set $End_L(q_{i+1}, R)$ to be the rightmost point on $\bigcup_{j=0}^{k} U_j$. That this sub-step takes $O(k)$ time can be seen by noting that we already know $Begin_L(\beta_j, R)$ and $End_L(\beta_j, R)$, and hence testing whether $U_j = \varnothing$ takes constant time (by testing whether $Begin_L(\beta_j, R) = \varnothing$).

(4.2) If $\bigcup_{j=0}^{k} U_j = \varnothing$ then go to Step (5). Otherwise let U_α be the highest nonempty U_j $(1 \le j \le k)$.

Repeat the following (i)-(iii):

(i) Let v be the rightmost point of U_α. Let U_γ be the highest nonempty U_j that is below U_α (i.e. $\alpha < \gamma \le k$) and has its rightmost point to the right of v; if no such U_γ exists then go to Step (5). Locating U_γ can clearly be done in $O(\gamma - \alpha)$ time.

(ii) Start at the leftmost point of U_γ and trace it left-to-right until the first point (say, w) to the right of v is reached: stop the scan of U_γ at w and add edge (v, w) to $E(L, R)$. We "charge" the cost of tracing the portion of U_γ that is to the left of v to the points so traced (one unit per point traced).

(iii) Set $\alpha := \gamma$ and go to (i).

Note: Sub-steps (4.1) and (4.2) can be combined; we chose to keep them separate for ease of exposition.

The cost of sub-step (4.2) is $O(k)$ plus the cost of the "charges" done in (ii). Let us count the overall cost of the charges done in (ii). A point (say, u) that gets charged one unit in (ii) will never get charged again in the future, because when executing Step (4) for a future q_{j+1} $(i<j)$, u will be "shielded" by v; i.e. u will not belong to the $MAX(Strip_L(\beta,R))$ of any β in $MAXREV(\{q_1, \cdots, q_j\})$. Thus the cost of all "charges" done in (ii) is $O(|L|)$.

(5) Push q_{i+1} on $STACK$, then set $i:=i+1$.

To analyze the time complexity of the above procedure, simply observe that $q_i \in R$ gets pushed on $STACK$ exactly once (once such a point q_i is removed from $STACK$, it cannot belong to the $MAXREV$ of the subset of points of Q_R already scanned, since at least one point of this subset is above it and to its left). Thus the total time taken by the above procedure is $O(|L|+|R|)$. □

Corollary 1. $G(S)$ has $O(n\log n)$ edges and can be built in $O(n\log n)$ time, where $n=|S|$.

Proof. Choose $|L|=|R|=n/2$, and let $f(n)$ denote the maximum number of edges that $G(S)$ can have. The edge set of $G(S)$ consists of the (not necessarily disjoint) union of $E(L,R)$, $CROSS(L,R)$, and the edge sets of $G(L)$ and $G(R)$. The number of edges in each of $G(L)$ and $G(R)$ is at most $f(n/2)$. By observations 1 and 2, $E(L,R)$ and $CROSS(L,R)$ have at most $n/2-1$ and $n/2$ edges, respectively. Therefore $f(n) \le 2f(n/2)+n-1$, and hence $f(n)=O(n\log n)$. The $O(n\log n)$ time bound for constructing $G(S)$ is by a straightforward divide and conquer, with Lemma 1 giving the needed linear time conquer step. □

Observation 3. There exists an S such that $G(S)$ has $\Omega(n\log n)$ edges.

Proof. Let $g(n)$ denote the number of edges that $G(S)$ has by our construction. Construct three identical sets of $n/3$ points each (call them S_1, S_2, S_3), each of which individually gives rise to a $G(S_i)$ that has $g(n/3)$ edges. Now, stack S_1, S_2, S_3 on top of one another so that the lowest point in S_1 is higher than the highest point in S_2, the lowest point in S_2 is higher than the highest point in S_3, and each point of S_1 has same x-coordinate as the corresponding point of S_2 or S_3. Now, disturb the above situation as follows: shift every point of S_1 to the right by an extremely small amount ε, and simultaneously shift every point of S_2 to the left by the same amount ε (the points in S_3 don't move). Let S be the set of points consisting of the union of the new (shifted) S_1, the new S_2, and S_3. The slight shifting of S_1 to the right and S_2 to the left means that for each point x_1 of S_1, the corresponding point of S_2 (call it x_2) is to its left by a 2ε amount, and the corresponding point of S_3 (call it x_3) is to its left by an ε amount. Thus in $G(S)$, each x_1 causes the edge (x_2,x_3) to be present. Thus $G(S)$ has at least $3g(n/3)+n/3$ edges, and hence $g(n)\geq 3g(n/3)+n/3$, resulting in $g(n)=\Omega(n\log n)$. □

Let the *label* of a point $p\in S$ with respect to set S (henceforth denoted $label_S(p)$) be as in the definition of problem **P2**.

Let S be partitioned into four subsets A_1,A_2,A_3,A_4, where A_i is to the left of A_{i+1}. For every $p\in A_{i+1}$, let $Left_{A_i}(p,A_{i+1})$ be the smallest $label_S(q)$ over all q that are on the portion of $MD_S(p)$ that lies in A_i; that is,

$$Left_{A_i}(p,A_{i+1})=Min\{label_S(q) : q\in MAX(Strip_{A_i}(p,A_{i+1}))\} \quad \text{if } Strip_{A_i}(p,A_{i+1})\neq\emptyset,$$

$$Left_{A_i}(p,A_{i+1})=\infty \quad \text{otherwise.}$$

Observation 4. Let $p\in A_2$. If $DOM_S(p)\neq\emptyset$, then

$label_S(p)=w(p)+Min\{Left_{A_1}(p,A_2),Min\{label_S(q):q\in MD_{A_2}(p)\}\}$.

Proof. An immediate consequence of the definitions and the fact that $MD_S(p)$ is the concatenation of $MAX(Strip_{A_1}(p,A_2))$ with $MD_{A_2}(p)$. □

Observation 5. For every $p\in A_3$, we have

$$Left_{A_1\cup A_2}(p,A_3)=Min\{Left_{A_1}(p,A_2\cup A_3),Left_{A_2}(p,A_3)\}.$$

Proof. An immediate consequence of the fact that $MAX(Strip_{A_1\cup A_2}(p,A_3))$ is the concatenation of $MAX(Strip_{A_1}(p,A_2\cup A_3))$ with $MAX(Strip_{A_2}(p,A_3))$. □

3. A Special Class of Tree Computations

Chazelle [C] has given a general technique which, given any n paths on a free tree that has a real label associated with each node, computes the smallest label on each of these n paths in $O(n\log n)$ time. In our algorithm for solving problem **P2** (given in the next section), we will need a similar computation on a rooted tree in which the n paths have a *nested property* (defined below). In Lemma 2, we establish that this can be done in $O(n)$ time.

Definition 1. Let $C=(P_1, \cdots, P_l)$ be a sequence of descendent-to-ancestor paths in a rooted tree T; path P_i begins at u_i and ends at w_i, where w_i is an ancestor of u_i. We say that C *has the nested property* iff

(i) $i<j$ and $P_i \cap P_j \neq \emptyset$ imply that w_j is ancestor of w_i, and

(ii) $i<j<k$ and $P_i \cap P_j \cap P_k \neq \emptyset$ imply that $P_i \cap P_k \subseteq P_j \cap P_k$.

For example, in the tree shown in Figure 3, if $P_1=a,b,c$, $P_2=u,v,b,c,d$, $P_3=a,b,c,d,e$, and $P_4=w,v,b,c,d,e,f$, then (P_1,P_2,P_4) has the nested property but (P_2,P_3,P_4) does not.

Lemma 2. Let T be an n-node rooted tree represented by *parent* pointers. In addition to *parent*(v), each node v also has a real label $l(v)$ associated with it. Let $C=(P_1, \cdots, P_n)$ be a sequence of descendent-to-ancestor paths in T. Let $f(i)$ be the smallest $l(v)$ over all v on path P_i. If C has the nested property, then $f(1), f(2), \cdots f(n)$ can be computed in $O(n)$ time.

Proof. We use the path compression technique previously used to solve the UNION-FIND problem [AHU]; the nested property will be crucial in proving that the algorithm actually runs in linear time. Assign to each node p of T a temporary label $Temp(p)$, initially set to $l(p)$; the significance of these $Temp$ labels is that as we do path compression on T, the $f(i)$ of every P_i yet to be traced equals the smallest $Temp$ label on it (this is certainly true initially, and will be maintained as we do path compressions). In what follows, we use T_0 to denote the initial (i.e. unmodified) tree T, and we view a path P_i as being defined by its two endpoints u_i and w_i rather than by a sequence of nodes in T_0 (path compression on T may shorten a path in T but does not change its endpoints).

We process the n paths in the order $P_1, \cdots, P_n$. To process P_i, we first trace it on T and compute the smallest $Temp(q)$ over all q on it, which is $f(i)$. Then we modify T by doing path compression along the path just traced, as follows. First, by tracing P_i once in the backward direction (from w_i to u_i), we compute for all $p \in P_i$, the quantity $g(p)$=$Min\{Temp(q)$:q is on the path from w_i to $p\}$. Once this is done, we modify T by making every $p \in P_i - \{w_i\}$ a child of w_i, and changing its temporary label by doing $Temp(p)$:=$Min\{Temp(p), g(p)\}$. Figure 4 illustrates the effect of this on T if P_i=a,b,c,d (in that figure, the numbers between parentheses are $Temp$ values).

A P_j yet to be processed (i.e. one with $j>i$) may have been "shortened" by the path compression made along P_i; however, because of property (i) (of Definition 1) and because of the the way the $Temp$ labels are updated, $f(j)$ is still the smallest $Temp$ on the u_j-to-w_j path in the modified tree T. This modification of T maintains the nested property for the sequence of paths yet to be processed, i.e. for the sequence $(P_{i+1}, \cdots, P_n)$ where every P_j $(j>i)$ is the u_j-to-w_j path in the modified tree T; to see this, observe that every such P_j $(j>i)$ ends at a w_j whose *parent* pointer is the same as the one in T_0 (because of property (i) and the order in which we are processing the P_i's). We now must show that the sum of the lengths of all the P_i's traced in this manner is $O(n)$. We say that an edge e of T_0 is *first traced by* P_j iff e belongs to P_j but not to any other P_k with $k<j$. When we trace P_i in the path-compressed tree T that resulted from processing paths $P_1, \cdots, P_{i-1}$, we partition the cost of tracing P_i into two components: The *strict cost* is that of traversing the edges first traced by P_i, and the *extra cost* is that of tracing the other edges (the latter may include edges first traced by P_j's with $j<i$ as well as edges previously added by the path compression process). The sum of the strict costs of all the P_i's is trivially $O(n)$. We now prove that the total extra cost is also $O(n)$. Let C_i denote the set of paths that were processed before P_i and have a nonempty intersection with P_i in T_0, i.e. C_i=$\{P_j : j<i$ and $P_j \cap P_i \neq \emptyset$ in $T_0\}$. Let P_a and P_b be paths in C_i; we say that P_a *beats* P_b iff $a>b$ and $P_a \cap P_b \neq \emptyset$ in T_0. Note that if P_a beats P_b in C_i, then the nested property implies that $P_b \cap P_i \subseteq P_a \cap P_i$ in T_0. For every $P_j \in C_i$, let C_{ij} denote the subset of C_i each of whose elements has a nonempty intersection with P_j in T_0 (see Figure 5). The nested property implies that, for every $P_k \in C_{ij}$, $P_{max(j,k)}$ beats $P_{min(j,k)}$. Path P_j is said to be a *chief in* C_i iff it beats every $P_k \in C_{ij}$, i.e. iff $j > \max\{k : P_k \in C_{ij}\}$. In Figure 5, the chiefs in C_i are P_a and P_c. Let D_i be the subset of C_i that contains only the chiefs. The extra cost of tracing P_i in T is equal to $|D_i|$, because the path compression that was done after processing each chief in C_i has reduced the intersection of that chief with P_i to exactly one edge; we "charge" a unit of this extra cost to each chief. A chief in C_i (say, P_a) will be prevented by P_i from ever being chief in a subsequent C_j $(j>i)$; to see this, note that if such a P_j $(j>i)$ intersects P_a in T_0 then it must also intersect P_i in T_0 (because $a<i<j$), and therefore P_i will belong to C_{ja} and will beat P_a in C_j. Hence the overall extra cost is at most n. □

4. Computing the $\text{label}_S(p)$'s

In this section we give an $O(n \log n)$ time, $O(n)$ space algorithm for solving problem **P2**.

Let $S=\{p_1, \cdots, p_n\}$ be the set of input points whose $label_S(p)$'s we wish to compute. To simplify the notation, we assume that the p_i's are given already sorted by increasing x coordinates, i.e. $X(p_1)<X(p_2)< \cdots <X(p_n)$. The algorithm that follows omits the computation of $predecessors_S(p)$ (including it would have unnecessarily cluttered the exposition). The interested reader can easily modify the algorithm so that it computes $predecessors_S(p)$ as well as $label_S(p)$ for all $p \in S$. The algorithm is initially called with $R=S$ and $Left_{\emptyset}(p,S)=-\infty$ for all $p \in S$, and it returns with $label_S(p)$ computed for all $p \in S$.

Algorithm MAXDOM(R)

Input: A contiguous m-subset R of S, i.e. $R=\{p_r, \cdots, p_{r+m-1}\}$; for every $p \in R$, $Left_L(p,R)$, where $L=\{p_1, \cdots, p_{r-1}\}$. In addition, the input includes the list Q_R containing the points of R sorted by increasing y coordinates.

Output: The labels $label_S(p_r), \cdots, label_S(p_{r+m-1})$.

Overview of Algorithm: The algorithm partitions R into subsets A and B such that $|A|=|B|=m/2$ and A is to the left of B. Since $Left_L(p,A)$ is given for all $p \in A$ (it equals

$Left_L(p,R)$), the algorithm can recursively call itself for set A, obtaining $label_S(p)$ for every $p \in A$. Then, using the labels so computed, the algorithm computes $Left_{L \cup A}(p,B)$ for every $p \in B$, in linear time. After that, the algorithm recursively calls itself for set B (it can do so because it now knows $Left_{L \cup A}(p,B)$ for all $p \in B$). The trick is how to compute $Left_{L \cup A}(p,B)$ for all $p \in B$ in linear time, knowing $Left_L(p,R)$ for every $p \in R$ and $label_S(p)$ for every $p \in A$; lemmas 1 and 2 are used for achieving this.

Step 1. If $m=1$ then set $label_S(p_r):=w(p_r)+Left_L(p_r,R)$ if $Left_L(p_r,R) \neq \infty$; set $label_S(p_r):=w(p_r)$ if $Left_L(p_r,R)=\infty$. Then return. If $m>1$ then proceed to Step 2.

Step 2. Let $A=\{p_r, \cdots, p_{r+m/2-1}\}$, $B=\{p_{r+m/2}, \cdots, p_{r+m-1}\}$. Extract from Q_R the lists Q_A and Q_B containing the points of A and B, respectively, sorted by increasing y components.

Step 2 takes $O(m)$ time.

Step 3. Since we have Q_A and $Left_L(p,A)$ for every $p \in A$ (it equals $Left_L(p,R)$), we can recursively solve the problem for the set A by doing $MAXDOM(A)$. This recursive call returns $label_S(p_r), \cdots, label_S(p_{r+m/2-1})$.

Step 4. This step computes the forest $F=(A,E(A,B))$ together with $Begin_A(p,B)$ and $End_A(p,B)$ for every $p \in B$. By Lemma 1, this can be done in in $O(m)$ time.

Step 5. Let $Q_B=(b_1, \cdots, b_{m/2})$ where $Y(b_1)< \cdots <Y(b_{m/2})$, and let $Path(b_i)$ denote the path from $Begin_A(b_i,B)$ to $End_A(b_i,B)$ in F. Use the forest $F=(A,E(A,B))$ created by the previous step to compute, for every $p \in B$ such that $Begin_A(p,B) \neq \emptyset$, the quantity $Left_A(p,B)= Min\{label_S(q): q \in Path(b_i)\}$. Lemma 3 (given at the end of this section) shows that the sequence of paths $Path(b_1), \cdots, Path(b_{m/2})$ has the nested property. This and Lemma 2 imply that Step 5 can be done in $O(m)$ time.

Step 6. For every $p \in B$, set $Left_{L \cup A}(p,B):=Min\{Left_L(p,R), Left_A(p,B)\}$.

This step takes $O(m)$ time, and its correctness follows from Observation 5 (in Section 2).

Step 7. Recursively solve the problem for set B by doing $MAXDOM(B)$. This returns $label_S(p_{r+m/2}), \cdots ,label_S(p_{r+m-1})$.

(End of Algorithm)

Theorem 1. *MAXDOM(S)* returns $label_S(p)$ for every $p \in S$ (and thus solves problem **P2**) in $O(n \log n)$ time and $O(n)$ space.

Proof. The running time $T(m)$ of procedure *MAXDOM* satisfies the recurrence $T(m) \leq 2T(m/2)+O(m)$ and hence $T(m)=O(m \log m)$. The space $S(m)$ satisfies the recurrence $S(m) \leq S(m/2)+O(m)$, and thus $S(m)=O(m)$. Correctness is easily established by induction on $|R|$, using observations 4 and 5. □

Lemma 3. The sequence $Path(b_1), \cdots ,Path(b_{m/2})$ of descendent-to-ancestor paths in F has the nested property.

Proof. We first prove property (i) of Definition 1. Let $i<j$ and assume that $Path(b_i) \cap Path(b_j) \neq \emptyset$. Since $j>i$, b_j is above b_i. If b_j were to the right of b_i then the intersection of $Path(b_i)$ with $Path(b_j)$ would be empty, hence b_j must be to the left of b_i. Therefore $leader_B(b_j)$ is not above $leader_B(b_i)$. This, and the fact that $Path(b_i) \cap Path(b_j) \neq \emptyset$, imply that $Y(End_A(b_j,B)) \leq Y(End_A(b_i,B))$. Hence $End_A(b_j,B)$ is an ancestor of $End_A(b_i,B)$. We now prove that property (ii) of Definition 1 also holds. Let $i<j<k$ and assume that $Path(b_i) \cap Path(b_j) \cap Path(b_k) \neq \emptyset$. Property (i) implies that $End_A(b_k,B)$ is ancestor of $End_A(b_j,B)$, which is itself ancestor of $End_A(b_i,B)$. Because b_i is below b_j, which is below b_k, we also have

$$Y(Begin_A(b_i,B)) \leq Y(Begin_A(b_j,B)) \leq Y(Begin_A(b_k,B)).$$

This implies that the first (i.e. geometrically highest) point on $Path(b_i) \cap Path(b_k)$ is an ancestor of the first point on $Path(b_j) \cap Path(b_k)$. □

5. Computing the $MD_S(p)$'s

In this section we briefly sketch how the algorithm of the previous section can be modified to solve problem **P1**. This problem is considerably easier than **P2**, and the algorithm (given below) correspondingly simpler.

Algorithm MD_LIST

Input: A set S containing the points $p_1, \cdots, p_n$ where $X(p_1) < X(p_2) < \cdots < X(p_n)$.

Output: The lists $MD_S(p_1), \cdots, MD_S(p_n)$, together with the list Q_S containing the points of S sorted by increasing y coordinates.

Step 1. If $n=1$ then output $MD_S(p_1)=\emptyset$ and return. If $n>1$ then proceed to Step 2.

Step 2. Recursively solve the problem for the set $A=\{p_1, \cdots, p_{n/2}\}$. This recursive call returns $MD_S(p_1), \cdots, MD_S(p_{n/2})$, together with the list Q_A containing the points of A sorted by increasing y coordinates.

Step 3. Recursively solve the problem for the set $B=\{p_{n/2+1}, \cdots, p_n\}$. This recursive call returns $MD_B(p_{n/2+1}), \cdots, MD_B(p_n)$, together with the list Q_B containing the points of B sorted by increasing y coordinates.

Note. For every $p \in B$, the list $MD_S(p)$ is the concatenation of $MAX(Strip_A(p,B))$ with the already computed list $MD_B(p)$. $MAX(Strip_A(p,B))$ is the path from $Begin_A(p,B)$ to $End_A(p,B)$ in the forest $F=(A,E(A,B))$.

Step 4. Construct the forest F, together with $Begin_A(p,B)$ and $End_A(p,B)$ for every $p \in B$. This is done in $O(n)$ time (by Lemma 1).

Step 5. Use the forest F created by the previous step to compute, for every $p \in B$, the list $MAX(Strip_A(p,B))$. This list is obtained by simply tracing the path in F from $Begin_A(p,B)$ to $End_A(p,B)$ (no path compression is needed since we are interested in the paths themselves rather than in some function of them).

Step 6. For every $p \in B$, compute $MD_S(p)$ by concatenating $MAX(Strip_A(p,B))$ with $MD_B(p)$. This takes constant time per concatenation, for a total of $O(n)$ time.

Step 6. Merge Q_A and Q_B into Q_S and return. This takes $O(n)$ time.

(End of Algorithm)

Correctness of the above algorithm is easily established by induction on n. We analyze its time complexity by charging some of the time to the output, and using $T(n)$ to denote the time not charged to the output. Thus the total time will be $O(T(n)+t)$ where $t=\sum_{p\in S}|MD_S(p)|$. The cost of Step 5 is completely charged to the output, since every $MAX(Strip_A(p,B))$ is part of $MD_S(p)$. Since the cost charged to $T(n)$ includes $2T(n/2)$ plus an additional $O(n)$ time, we have $T(n)=O(n\log n)$. Thus we have established the following.

Theorem 2. Algorithm *MD_LIST* correctly solves problem **P1**, and runs in time $O(n\log n+t)$.

6. Applications

In this section we discuss some problems for which improved algorithms follow from our solution to the maxdominance problems **P1** and **P2**.

5.1. Permutation graphs and subsequence problems

For any undirected graph $G=(V,E)$, a subset H of the vertex set V is called a *dominating set* iff for every $u\in V$ there exists $v\in H$ such that u is adjacent to v. Set H is *independent* iff no two vertices in H are adjacent. The problem of finding a minimum independent dominating set (MIDS for short) is NP-hard for general graphs, however for the class of *permutation graphs* an $O(n^3)$ time solution was given in [FK], later improved to $O(n(\log n)^2)$ in [AMU]. We now briefly point out how our solution to problem **P2** implies an $O(n\log n)$ time solution for the MIDS problem.

In [AMU] the MIDS problem is reduced to that of computing a particular subsequence of a sequence of length n. Given a sequence $\alpha=a_1a_2\cdots a_n$ of numbers, a *subsequence* of α is a sequence $\beta=a_{i_1}a_{i_2}\cdots a_{i_k}$ such that $i_1<i_2<...<i_k$. If, in addition, $a_{i_1}<a_{i_2}<...<a_{i_k}$, then we say that β is an *increasing subsequence of* α. An increasing subsequence of α is *maximal* iff it is not a

proper increasing subsequence of any increasing subsequence of α. A *maximum* increasing subsequence is one of maximum length. Note that a maximum increasing subsequence is also maximal, but that a maximal increasing subsequence may not be maximum. For example, in the sequence 2,1,4,5,3 the increasing subsequence 1,3 is maximal but not maximum (for this example the length of a maximum increasing subsequence is three, e.g. 2,4,5). In [AMU] it was pointed out that MIDS can be reduced to the problem of computing a *shortest maximal increasing subsequence* (from now on called SMIS) of a sequence of n numbers. We now point out how our solution to problem **P2** implies an $O(n \log n)$ time solution to the SMIS (and hence MIDS) problem. For the sake of generality, we consider the weighted version of the problem, i.e. where every element a_i has an associated weight w_i, and the problem is then to compute a minimum-weight maximal increasing subsequence of the input sequence $\alpha = a_1 \cdots a_n$. This is done as follows: create a set of points $S = \{p_1, \cdots, p_n\}$ where $p_i = (i, a_i)$, and let the weight $w(p_i)$ of point p_i be w_i. Let the *label* of every point in S be defined as follows:

$$label(p) = w(p) \quad \text{if } DOM_S(p) = \varnothing,$$

$$label(p) = w(p) + Min\{label(q) : q \in MD_S(p)\} \quad \text{otherwise.}$$

As in **P2**, the predecessor of point p is any of the points which gave p its label. It is not hard to see that (i) the minimum-weight shortest maximal increasing subsequence of α has a weight equal to $Min\{label(p) : p \in MAX(S)\}$, and (ii) the corresponding subsequence of α can be retrieved by beginning at the smallest-labeled point in $MAX(S)$ and following the chain of predecessor pointers. These observations imply that our solution to problem **P2** implies a solution to SMIS (and hence MIDS) having complexity $O(n \log n)$ time and $O(n)$ space.

The known $O(n \log n)$ time solutions to the well studied problem of computing a maximum increasing subsequence [D,DMS] cannot be modified to solve the SMIS problem, which is considerably more difficult in spite of the apparent similarity.

5.2. Empty rectangle problem

Given a rectangle R and a set S of n points in R, a *valid rectangle* is one which is contained in R, has its sides parallel to those of R, and does not contain any of the points in S. Consider the problem of enumerating all the *restricted rectangles*, where a *restricted rectangle* (RR for short) is a valid rectangle such that each of its four edges either contains a point of S or coincides with an edge of R. Let s denote the number of RR's, i.e. the size of the output. Naamad et al. [NLH] prove that $s=O(n^2)$ and give an example in which $s=\Theta(n^2)$. They also show that when the points are drawn from a uniform distribution, the expected value of s is $O(n\log n)$.

In [AF] it was shown that any $O(T(n)+t)$ time algorithm for problem **P1** would immediately imply an $O(T(n)+s)$ time algorithm for enumerating all the RR's (recall that t is the size of the output to **P1**). The solution that was given in [AF] had $T(n)=n(\log n)^2$. Since our solution to **P1** has $T(n)=n\log n$, it automatically implies an $O(n\log n+s)$ time solution to the problem of enumerating all RR's. This is an improvement over the $O(n(\log n)^2+s)$ time algorithm given in [AF] and over the $O(\min(n^2,s\log n))$ time algorithm given in [NLH].

Since the expected value of s is $O(n\log n)$, our result implies an improvement by a factor of $\log n$ in the best known average case time complexity for the related problem of computing the largest (i.e. maximum area) RR. Similar bounds (using a different method) were recently independently established in [BE,O,PR]. A worst-case time bound of $O(n(\log n)^3)$ for finding the largest RR was given in [CDL], recently improved to $O(n(\log n)^2)$ in [AS].

5.3. Independent Sets in Overlap or Circle Graphs

Given n intervals $I_1,\cdots,I_n$ on the line, their corresponding *overlap graph* is the undirected graph having the I_i's as vertices, and such that there is an edge between intervals I_i and I_j iff these two intervals overlap but neither one contains the other. The problem of computing a maximum-weight independent set for such graphs (which are the same as circle graphs) was considered in [AH] and an algorithm of time complexity $O(nd)$ was given, where $d\leq n$ is a quantity whose expected value is proportional to n. Thus the average case time complexity of the algorithm given in [AH] is $O(n^2)$. Our solution to problem **P1** makes possible an $O(n\log n+t)$ time implementation of the same algorithm that was given in [AH], where t can still be quadratic in the worst case but has expected value $O(n \log n)$ (the algorithm is essentially the same as that of [AH] and we therefore refer the interested reader to that paper).

7. Conclusion

We gave asymptotically optimal algorithms for two maxdominance problems. These in turn implied improvements in the time complexities of a number of graph-theoretic and geometric problems. The techniques we used are of independent interest, and we have reason to believe they will be useful for solving other problems as well.

Acknowledgements. The authors are grateful to the referees for many useful comments. In particular, one of the referees pointed out a flaw in an earlier proof of Lemma 1.

References

[AF] M.J. Atallah and G.N. Frederickson, "A Note on Finding a Maximum Empty Rectangle," *Discrete Applied Mathematics*, Vol. 13, No. 1, January 1986, pp. 87-91.

[AH] A. Apostolico and S. E. Hambrusch, "New Clique and Independent Set Algorithms for Circle Graphs," Purdue CS Tech. Rept. # 608, june 1986.

[AHU] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

[AMU] M.J. Atallah, G.K. Manacher and J. Urrutia, "Finding a Minimum Independent Dominating Set in a Permutation Graph", Purdue C.S. Tech. Rept.

[AS] A. Aggarwal and S. Suri, "Fast Algorithms for Computing the Largest Empty Rectangle," to appear in *Proc. of 3rd ACM Symposium on Computational Geometry*, June 87.

[BE] B. Bhattacharja and H. ElGindy, "Fast Algorithms for the Maximum Empty Rectangle Problem," U. of Penn. and Simon Fraser U. Tech. Repts., March 1987.

[C] B. Chazelle, "Computing on a Free Tree Via Complexity-Preserving Mappings", *Proc. 25th Annual IEEE Symposium on Foundations of Computer Science*, pp. 358-368, 1984.

[CDL] B. Chazelle, R.L. Drysdale and D.T. Lee, "Computing the Largest Empty Rectangle", *SIAM J. on Computing*, Vol. 15, No. 1, pp. 300-315 (1986).

[DMS] R.B.K. Dewar, S.M. Merritt and M. Sharir, "Some Modified Algorithms for Dijkstra's Longest Upsequence Problem", *Acta Informatica*, Vol. 18, No. 1, pp.1-15 (1982).

[D] E.W. Dijkstra, "Some Beautiful Arguments Using Mathematical Induction", *Acta Informatica*, Vol.13, No. 1, pp. 1-8 (1980).

[FK] M. Farber and J. Mark Keil, "Domination in Permutation Graphs", *Journal of Algorithms*, 6, pp. 309-321 (1985).

[NLH] A. Naamad, D.T. Lee and W.-L. Hsu, "On the Maximum Empty Rectangle Problem", *Discrete Applied Mathematics*, 8, pp. 267-277 (1984).

[O] M. Orlowski, "A New Algorithm for the Largest Empty Rectangle Problem," manuscript.

[OV] M.H. Overmars and J. Van Leeuwen, "Maintenance of Configurations in the Plane", Journal of Computer and Systems Sciences, Vol. 23, pp. 166-204 (1981).

[PR] T. H. k. Prasad and C. P. Rangan "An Improved Algorithm for the Maximum Empty Rectangle Problem," IIT Madras Tech Rept., March 1987.

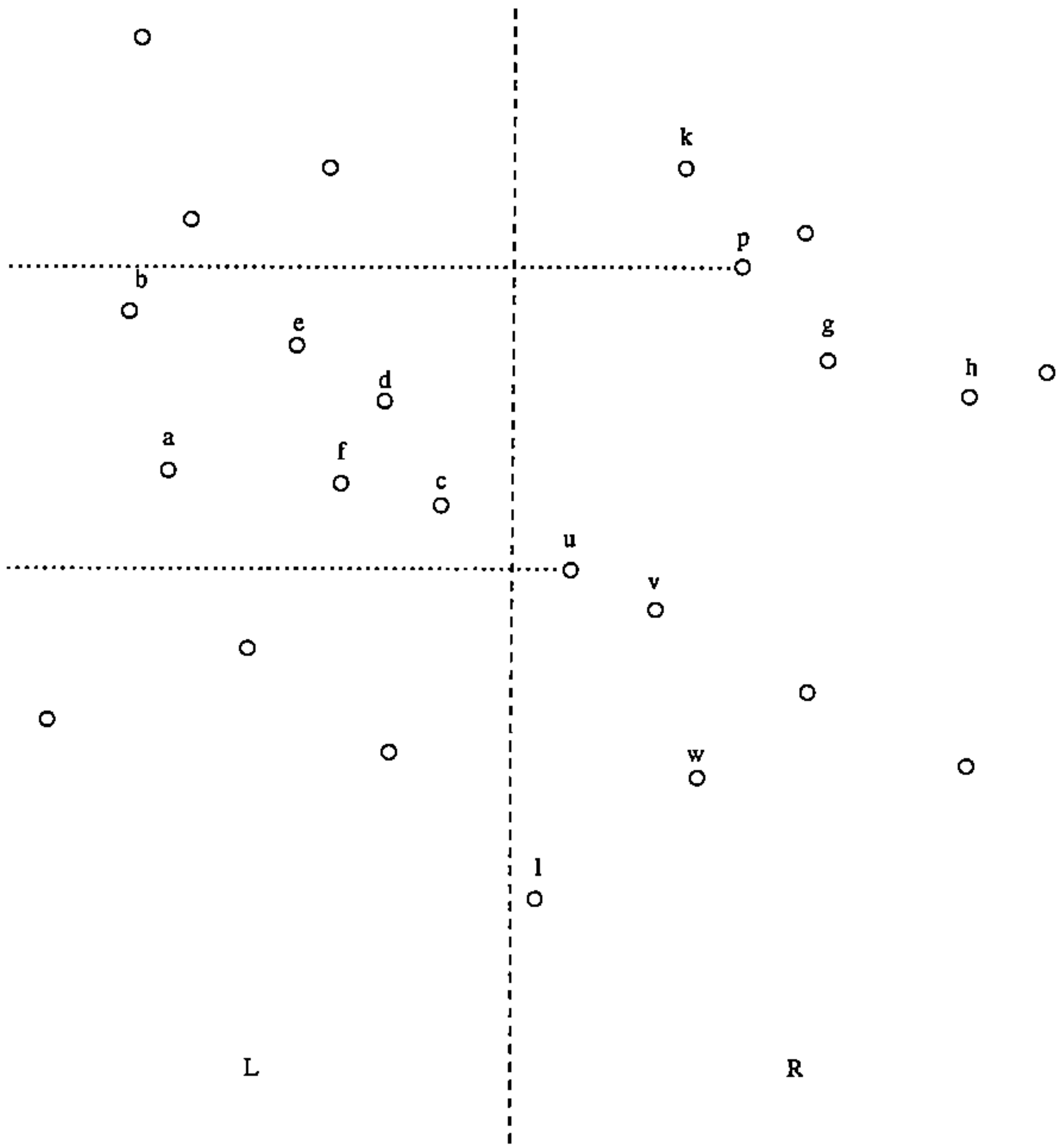

Figure 1. Illustrating the basic definitions

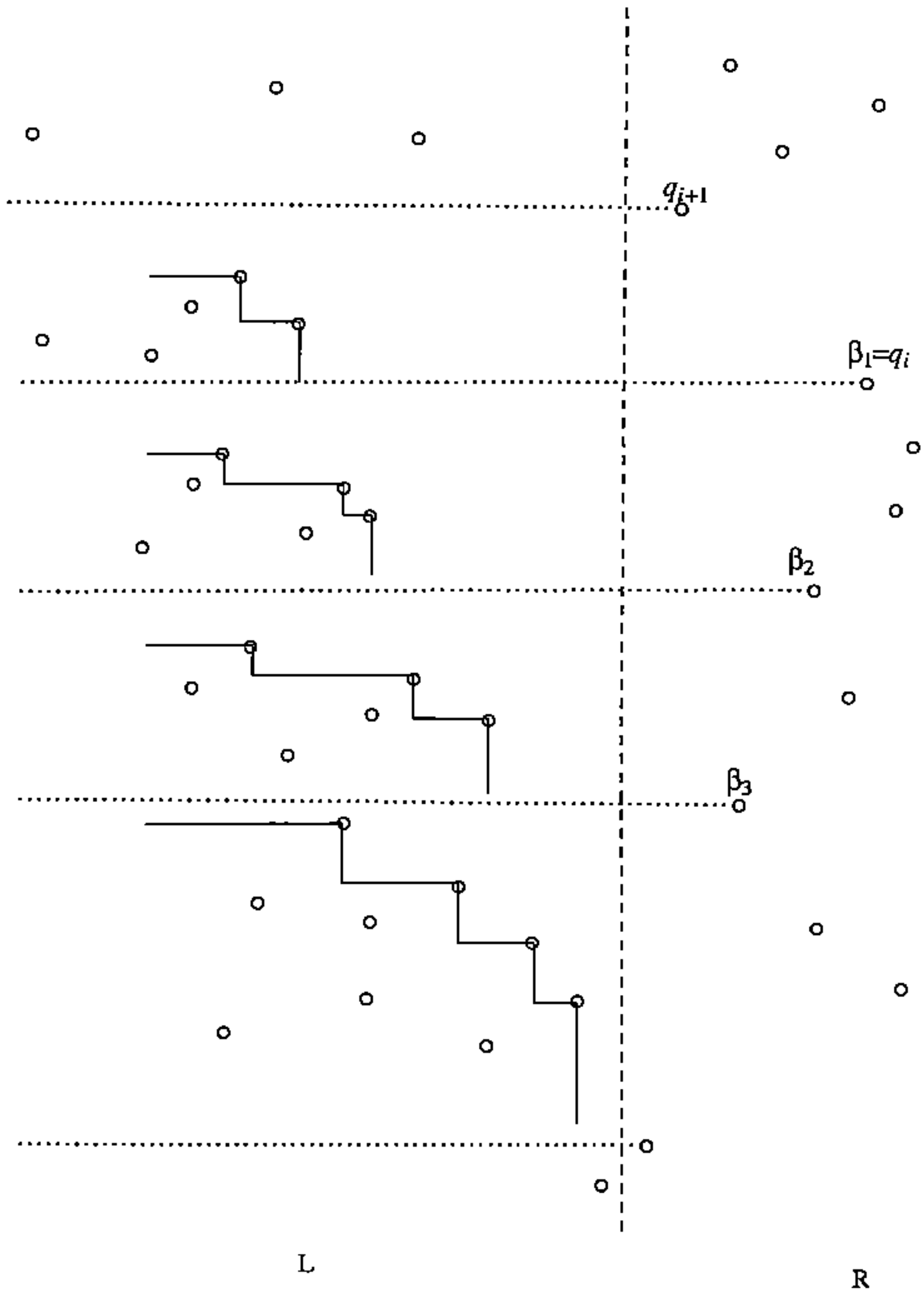

Figure 2. Illustrating Step 4 of the proof of Lemma 1.

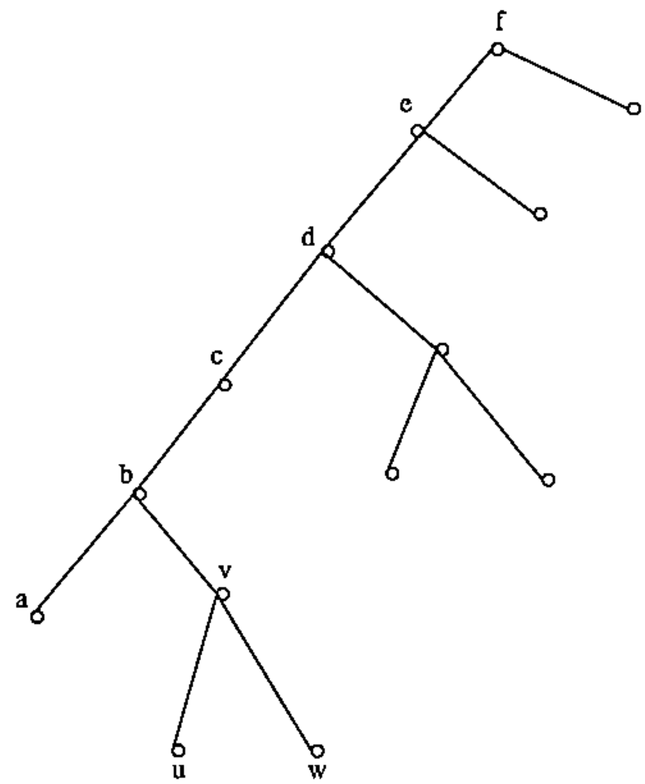

Figure 3. Illustrating Definition 1.

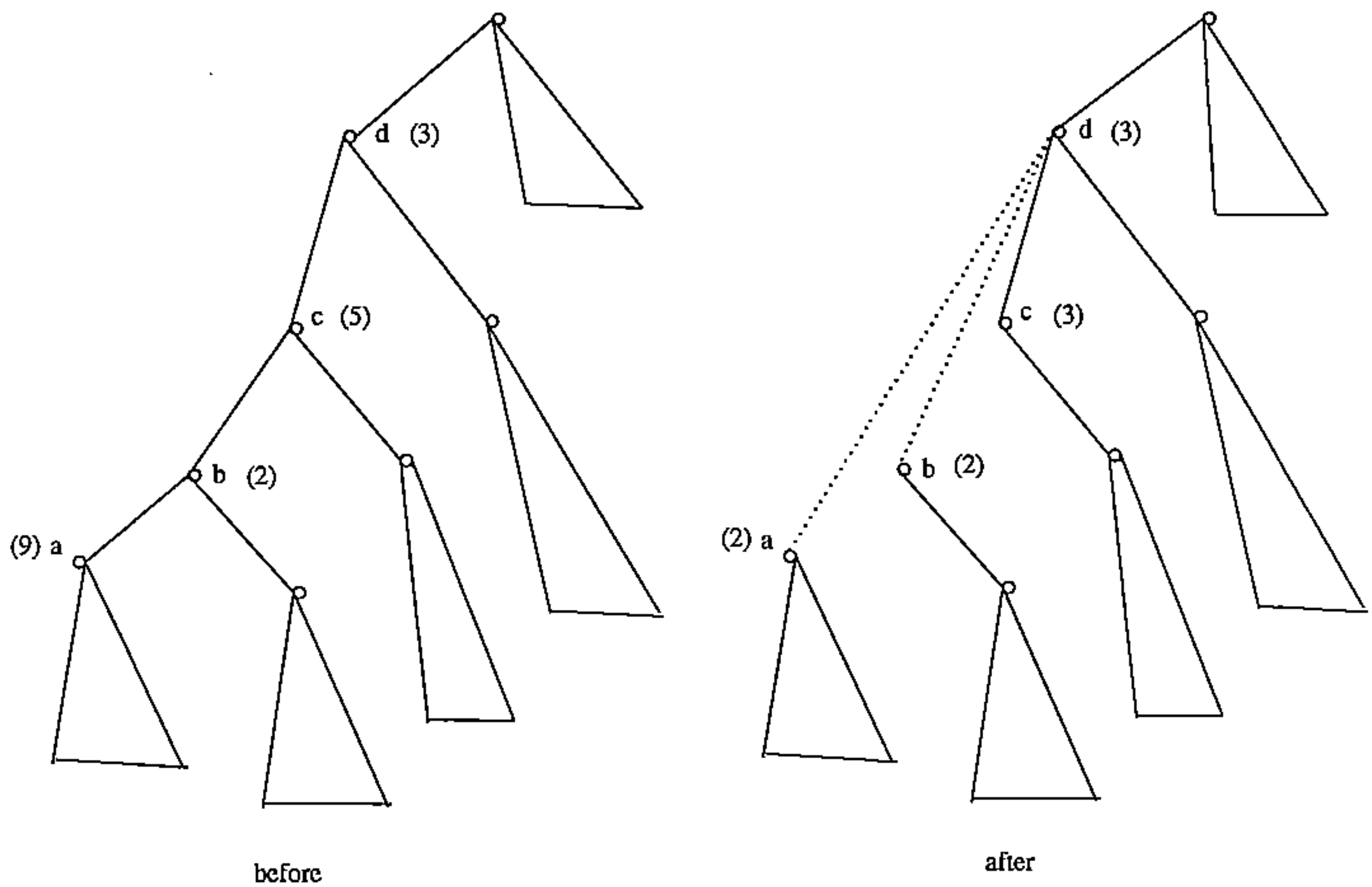

Figure 4. Illustrating how the *Temp* labels are updated.

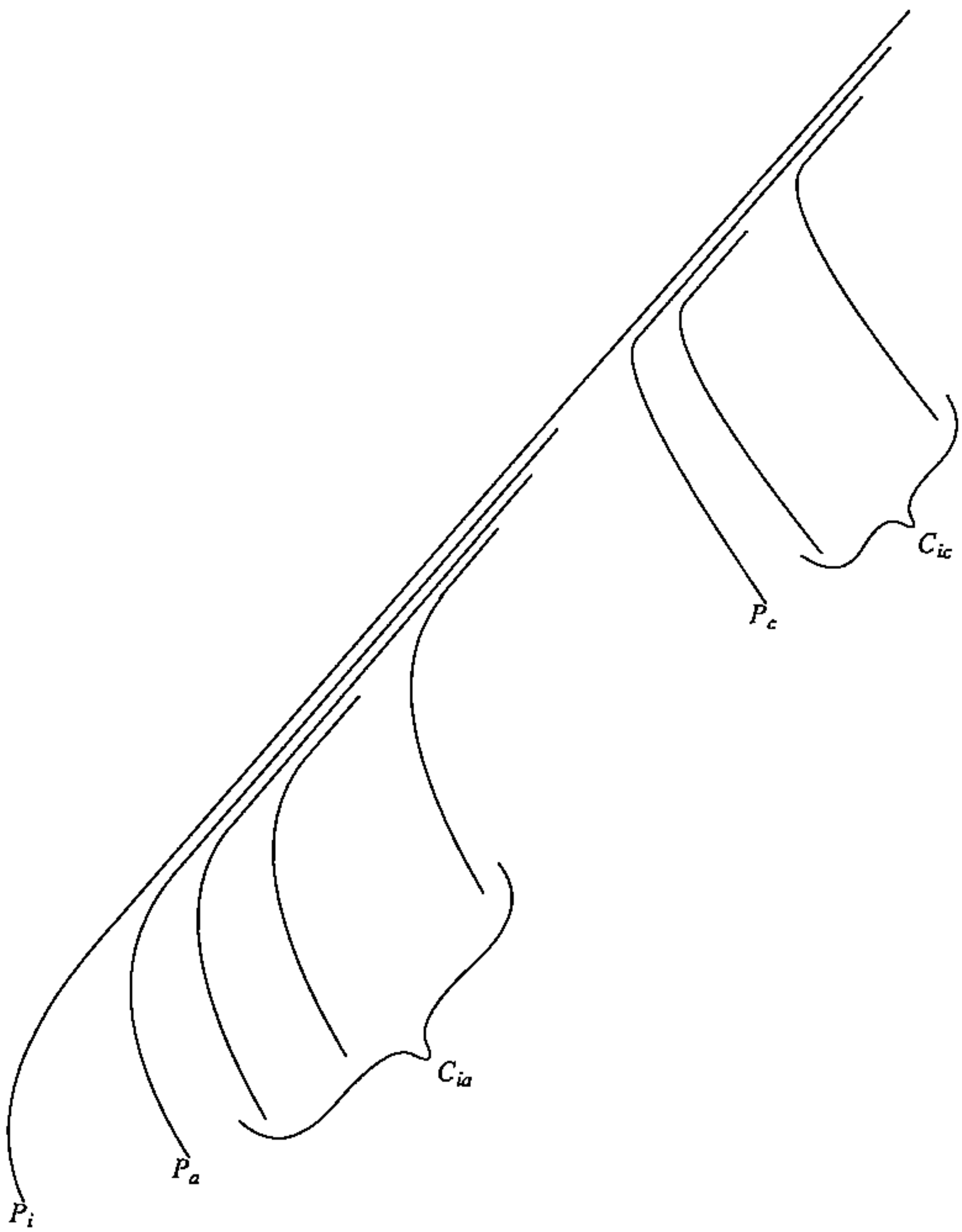

Figure 5. Here C_i contains seven paths. All paths shown are in T_0.