# Finding Shortest Maximal Increasing Subsequences and Domination in Permutation Graphs.

*Mikhail J. Atallah* (*)
*Gleen K. Manacher* (")
*J. Urrutia* (†)

## Abstract.

In this paper we present an algorithm with $O(n \log^2 n)$ complexity to find a shortest maximal increasing subsequence of a sequence of numbers. As a byproduct of this algorithm a $O(n \log^2 n)$ algorithm to find a minimum (weight) independent dominating set in permutation graphs is obtained, this compares favorable with the previous $O(n^3)$ algorithms known to solve this problem .

(*) Computer Science Dept, Purdue University, West Lafayette, IN 47907.

(") Dept. of Mathematics, University of Illinois, Chcago, IL 60614.

(†) Dept. of Computer Science, University of Ottawa, Ottawa, Ontario, Canada.

## 1. Introduction

Given a sequence $\boldsymbol{S} = \{a(1), a(2),...,a(n)\}$ of numbers, a *subsequence* of $\boldsymbol{S}$ is a sequence $\boldsymbol{S}' = \{a(i_1), a(i_2),..., a(i_k)\}$ contained in $\boldsymbol{S}$ such that $i_1 < i_2 <...< i_k$. If in addition $a(i_1) < a(i_2) <...< a(i_k)$, then we say that $\boldsymbol{S}'$ is an increasing subsequence of $\boldsymbol{S}$. An increasing subsequence $\boldsymbol{S}'$ of $\boldsymbol{S}$ is called *maximal* if it is not a proper increasing subsequence of another increasing subsequence of $\boldsymbol{S}$. A *maximum* increasing subsequence is one of maximum length.

Let $\pi$ be a permutation on the set $\boldsymbol{I}_n=\{1,2,...,n\}$. Let $G(\pi)$ be the graph with $V(G)= \boldsymbol{I}_n$ such that i is adjacent to j in $G(\pi)$ if and only if $i<j$ and $\pi^{-1}(i) > \pi^{-1}(j)$. A graph $H$ on n vertices is called a *permutation graph* if $H$ is isomorphic to $G(\pi)$ for a permutation $\pi$ on $\boldsymbol{I}_n$. For a graph $G$ a subset $S$ contained in $V(G)$ is called a dominating set if for every $u \in V(G)$ there exists $v \in \boldsymbol{S}$ such that u is adjacent to v. $\boldsymbol{S}$ is called independent if no two vertices in $\boldsymbol{S}$ are adjacent. In [4] the problem of finding a minimum independent dominating set in permutation graphs (from now on called **MIDS**) was studied. In the same paper an $\boldsymbol{O}(n^3)$ algorithm to solve the **MIDS** was presented.

In this paper we present a $\boldsymbol{O}(n \log^2 n)$ to solve the *shortest maximal increasing subsequence* of a sequence of numbers (from now on called **SMIS**). We show then that the **MIDS** problem in permutation graphs is equivalent to that of finding **SMIS** subsequence of a sequence of numbers. As a consequence an $\boldsymbol{O}(n \log^2 n)$ algorithm to solve the **MIDS** problem in permutation graphs is obtained which compares favorably with the $\boldsymbol{O}(n^3)$ algorithm presented in [4]. It is interesting to notice that the problem of computing the *longest increasing subsequence* (for short **LIS** problem) of a sequence of integers has been widely studied [2,3], while the problem of finding the *shortest* maximal increasing subsequence has not. Moreover, the $\boldsymbol{O}(n \log n)$ algorithms to solve the **LIS** problem can not be modified to solve the **SMIS** problem. So in spite of their apparent similarity, the two problems seem to be quite different. We should point out that our $\boldsymbol{O}(n \log^2 n)$ algorithm uses techniques and data structures that were originally developed to solve vector dominance problems in Computational Geometry. In the weighted versions of the **MIDS** and **SMIS** problems, a non-negative

weight is associated to every element of $\boldsymbol{I}_n$ or $\boldsymbol{S}$. The problems then become those of finding the *minimum weight independent dominating set* (**MWIDS**) and *minimum weight maximum increasing subsequence* (**MWMIS**) respectively. The algorithms we obtain for the unweighted cases can be easily modified to solve the weighted ones, so we shall no longer concern ourselves with the weighted cases.

## 2. Minimum Weight Independent Dominating Sets in Permutation Graphs

The main objective of this section is to prove the following result.

**Theorem 1**. The problem of finding the **SMIS** of a sequence of numbers is equivalent to finding a **MDIS** in a permutation graph.

Before proving Theorem 1, we shall obtain some properties of permutation graphs.

**Lemma 1**. Let $\boldsymbol{I} = \{i_1, i_2, ..., i_k\}$ be a subset of $\boldsymbol{I}_n$; $i_1 < i_2 < ... < i_k$ then $\boldsymbol{I}$ forms an independent set in $G(\pi)$ if and only if $\pi^{-1}(i_1) < \pi^{-1}(i_2) < ... < \pi^{-1}(i_k)$.

**Proof**. By definition, two vertices i, j; i<j; in $G(\pi)$ are adjacent if $\pi^{-1}(i) > \pi^{-1}(j)$. Thus if $\pi^{-1}(i) < \pi^{-1}(j)$, i and j are not adjacent in $G(\pi)$. The result now follows.

QED.

We can now prove:

**Lemma 2**. Let $\boldsymbol{I} = \{i_1, i_2, ..., i_k\}$ be a subset of $\boldsymbol{I}_n$; $i_1 < i_2 < ... < i_k$ such that $\pi^{-1}(i_1) < \pi^{-1}(i_2) < ... < \pi^{-1}(i_k)$. Then $\boldsymbol{I}$ is an independent dominating set in $G(\pi)$ if and only if $\{\pi(i_1), \pi(i_2), ..., \pi(i_k)\}$ forms a maximal increasing subsequence of $\{\pi(1), \pi(2), ..., \pi(n)\}$.

**Proof**. Suppose that $\{\pi(i_1), \pi(i_2), ..., \pi(i_k)\}$ forms a maximal increasing subsequence of $\{\pi(1), \pi(2), ..., \pi(n)\}$. Then $\boldsymbol{I}$ forms a dominating set in $G(\pi)$, otherwise there is a vertex $j \in \boldsymbol{I}_n$ such that $\boldsymbol{I} \cup \{j\}$ is also an independent set. But by

Lemma 1, we can now add $\pi(j)$ to $\{\pi(i_1), \pi(i_2),..., \pi(i_k)\}$ obtaining a new increasing subsequence of $\{\pi(1), \pi(2),..., \pi(n)\}$ which properly contains $\{\pi(i_1), \pi(i_2),..., \pi(i_k)\}$, contradicting the maximality of it.

QED.

**Proof of Theorem 1**. By lemma 2 any independent dominating set in $G(\pi)$ generates a maximum increasing subsequence in $\{\pi(1), \pi(2),..., \pi(n)\}$ . Then finding a *minimum* independent dominating set in $G(\pi)$ is equivalent to finding the *minimun* maximal increasing subsequence in $\{\pi(1), \pi(2),..., \pi(n)\}$.

QED.

### 3. Finding Shortest Maximal Increasing Subsequences.

In this section, we shall present an algorithm to find the **SMIS** of a sequence of numbers. We now introduce some terminology and review some known results which will be needed later.

Let $P$ be a set of points in the plane. We use $X(p)$ and $Y(p)$ to denote the $x$ and (respectively) $y$ coordinates of a point $p$. Point $p_i$ is said to *dominate* $p_j$ iff $X(p_i) > X(p_j)$ and $Y(p_i) > Y(p_j)$. We use $DOM(p_i)$ to denote the subset of points in $P$ that are dominated by point $p_i$; ie., $DOM(p_i)$ contains the points of $P$ that are below and to the left of $p_i$. A point of $P$ is *maximum* if no other point of $P$ dominates it. From now on we use $MAX(P)$ to denote the set of maxima of $P$.

Our algorithm makes use of the following elegant result of Overmars and Van Leeuwen: There exists a data structure for dynamically maintaining the maxima of a set of points in the plane, such that insertions and deletions take time $\boldsymbol{O}(n \log^2 n)$ per operation. Such an *augmented tree structure* (as it is called in [5]) takes $\boldsymbol{O}(n)$ storage space and can initially be created in time $\boldsymbol{O}(n \log n)$. At any time, the maxima are available at the root, in a concatenable queue "attached" to the root. An augmented tree structure can also support *SPLIT* and *CONCATENATE* operations in time $\boldsymbol{O}(\log^2 n)$ per operation (even though this is not mentioned explicitly in [5], it easily follows from it). In other words, if the points are stored in the augmented tree

structure according to (say) their $y$ -coordinate, then a *SPLIT* operation about any horizontal line $y = y_o$ can be implemented in time $\boldsymbol{O}(\log^2 n)$. Such a *SPLIT* operation results in two augmented tree structures; one for the points above the vertical line, and one for those below it.

A *CONCATENATE* operation also takes $\boldsymbol{O}(\log^2 n)$ time and has the reverse effect of a *SPLIT* . In the context of this paper, every point will have a *label* associated with it, and we will need to maintain the smallest- labelled maximum at the root of the augmented tree structure (more precisely, at the root of the concatenable queue attached to the root). It is not hard to show that this can be done without losing the $\boldsymbol{O}(\log^2 n)$ time- per-operation performance (this is done using standard data structure techniques, such as those described in reference [1]).

We now have all the ingredients which we use in our algorithm.

### 3.1. The algorithm

Let $a(1),\ldots, a(n)$ be the input sequence. Let $\square_i$ be a shortest maximal increasing subsequence of $a(1),\ldots, a(i)$ which ends with $a(i)$. Let *label*(i) be the length of $\square(i)$ and let *predecessor*(i) be the index of the predecessor of $a(i)$ in $\square(i)$, i.e. $\square(i)$ ends with $a$ (predecessor(i)), $a(i)$.

**Algorithm MINMAX**

*Input:* Sequence $a(1), \ldots, a(n)$

*Output:* A minimum-length maximal increasing subsequence of $a(1), \ldots, a(n)$

*Method:* The algorithm sets *label*(1):=1 and *predecessor*(1):= $\varnothing$. Next the algorithm creates points $p_1,\ldots, p_n$ in the plane, where $p_i = (i, a(i))$, $1 \le i < n$. Then the algorithm sweeps a vertical line $L$ from left to right, maintaining the maxima of the set of points to the left of $L$ in an augmented tree structure $T$. When the left-to-right sweeping line $L$ encounters a point $p_i$ , the following steps 1-3 are taken:

1) The algorithm splits $T$ about the horizontal line $y = Y(p_i)$, obtaining two augmented tree structures $T_{up}$ and $T_{down}$ . Note that $T_{down}$ contains the set $MAX(DOM(p_i))$ in a concatenable queue attached to its root, and the smallest-labeled point of $MAX\,(DOM\,(p_i))$ is attached to the root of this concatenable queue.

2) If $MAX(DOM(p_i))$ is empty then the algorithm sets $label(i):=1$ and *predecessor*(i):= ∅. Otherwise it sets *predecessor*(i) equal to the index j of the smallest labeled point $p_j$ of $MAX(DOM(p_i))$, then it sets *label*(i):=*label*(j)+1.

3) Rebuild $T$ by concatenating $T_{up}$ and $T_{down}$, then insert $p_i$ in $T$.
After the line $L$ sweeps past $p_n$ (the rightmost of the $p_i$'s), the algorithm chooses a smallest-labeled point in $MAX(\{p_1,\ldots, p_n\})$; let $p_k$ be this point. The algorithm then sets s:= $a_k$ and then, so long as *predecessor*(k)≠ ∅, it does s:=$a_{predecessor}$ is followed by k:=*predecessor*(k). When *predecessor*(k)= ∅ the algorithm outputs □.

**End of Algorithm MINMAX**

That *label*(i) and *predecessor*(i) are computed correctly by the algorithm follows from the definitions of these two functions. That the s produced by the algorithm is the desired subsequence follows from the definitions of the *label* and *predecessor* functions and the observation that any maximal increasing subsequence must end with an $a_k$ such that $a_k \square MAX(\{p_1,\ldots, p_n\})$.

That the algorithm runs in $\boldsymbol{O}(n \log^2 n)$ time is an immediate consequence of the fact that each of the operations *INSERT*, *SPLIT*, and *CONCATENATE* takes $\boldsymbol{O}(\log^2 n)$ time in an augmented tree structure, and that the smallest labeled maximum is readily available at the root.

This completes the proof of the following:

**Theorem 2.** Given a sequence of integers $a_1,\ldots, a_n$, it is possible to find a shortest maximal increasing subsequence in time $\boldsymbol{O}(n \log^2 n)$ and space $\boldsymbol{O}(n)$.

Using theorems 1 and 2, we have:

**Theorem 3**. Finding a minimum independent dominating set in permutation graphs can be achieved in $O(n \log^2 n)$.

**4. Conclusions.**

In this paper we gave an $O(n \log^2 n)$ time algorithm to find a shortest maximal increasing subsequence of a sequence of n numbers. Using this algorithm, we can

find a minimum independent dominating set of a permutation graph in $O(n \log^2 n)$ time. These results can be easily extended to the weighted cases.

**References**.

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

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

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

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

[5]. M.H. Overmars and J. Van Leeuwen, "Maintenance of Configurations in the Plane", *Journal of Computer and Information Sciences*, 1981,pp. 166-204.