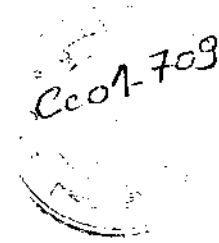
Frank Dehne Jörg-Rüdiger Sack
Nicola Santoro Sue Whitesides (Eds.)



Algorithms and Data Structures

Third Workshop, WADS '93
Montréal, Canada, August 11-13, 1993
Proceedings

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Series Editors

Gerhard Goos
Universität Karlsruhe
Postfach 69 80
Vincenz-Priessnitz-Straße 1
D-76131 Karlsruhe, Germany

Juris Hartmanis
Cornell University
Department of Computer Science
4130 Upson Hall
Ithaca, NY 14853, USA

Volume Editors

Frank Dehne
Jörg-Rüdiger Sack
Nicola Santoro
School of Computer Science, Carleton University
1125 Colonel By Drive, Ottawa, Canada K1S 5B6

Sue Whitesides
School of Computer Science, McGill University
3480 University Street, Montréal PQ, Canada H3A 2A7

CR Subject Classification (1991): F.1-2, E.1, G.2, I.3.5, H.3.3

ISBN 3-540-57155-8 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-57155-8 Springer-Verlag New York Berlin Heidelberg

Printed in Germany

Typesetting: Camera-ready by author
Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
45/3140-543210 - Printed on acid-free paper

PREFACE

The papers in this volume were presented at the Third Workshop on Algorithms and Data Structures (WADS '93). The workshop took place August 11 - 13, 1993, in Montréal, Canada. The workshop alternates with the Scandinavian Workshop on Algorithm Theory (SWAT), continuing the tradition of SWAT '88, WADS '89, SWAT '90, WADS '91, and SWAT '92.

In response to the program committee's call for papers, 165 papers were submitted. From these submissions, the program committee selected 52 for presentation at the workshop. Each paper was evaluated by at least three program committee members, many of whom called upon additional reviewers. In addition to selecting the papers for presentation, the program committee invited the following people to give plenary lectures at the workshop: Mikhail Atallah, Allan Borodin, Richard Cole, Richard Karp, Robert Tarjan, and Andrew Yao.

On behalf of the program committee, we would like to express our appreciation to the six plenary lecturers who accepted our invitation to speak, to all the authors who submitted papers to WADS '93, and to Rosemary Carter of Carleton University for her technical assistance to the program committee. Finally, we would like to express our gratitude to all the people who reviewed papers at the request of program committee members.

August 1993

Frank Dehne
Jörg-Rüdiger Sack
Nicola Santoro
Sue Whitesides

Conference Chair:

S. Whitesides (McGill U.)

Program Committee Chairs:

F. Dehne, J.-R. Sack, and N. Santoro
(Carleton U.)

Program Committee:

M.D. Atkinson (St. Andrew's U.)
H. Attiya (The Technion)
G. Ausiello (U. of Rome)
P. Flajolet (INRIA, Les Chesnay)
Z. Galil (Columbia U.)
S. Hambrusch (Purdue U.)
D. Kirkpatrick (UBC)
M. Klawe (UBC)
R. Kosaraju (Johns Hopkins U.)
J. van Leeuwen (U. of Utrecht)
F. Lombardi (Texas A&M U.)
F. Luccio (U. of Pisa)
J. Matoušek (Charles U.)
I. Munro (U. of Waterloo)
O. Nurmi (U. of Helsinki)
L. Pagli (U. of Pisa)
J. Reif (Duke U.)
R. Seidel (U. of California, Berkeley)
R. Tamassia (Brown U.)
É. Tardos (Cornell U.)
J. Urrutia (U. of Ottawa)
J. Vitter (Duke U.)
D. Wagner (TU Berlin)
S. Whitesides (McGill U.)
P. Widmayer (ETH Zürich)
F. Yao (Xerox PARC)

Invited Speakers:

M. J. Atallah
A. Borodin
R. Cole
R. M. Karp
R. E. Tarjan
A. C. Yao

Sponsored by:

NSERC, Carleton University, and McGill University

ADDITIONAL REFEREES

P. K. Agarwal
E. M. Arkin
D. Avis
A. Bertossi
P. Callahan
R. Canetti
P. Charraresi
J. Chen
Yi.-J. Chiang
R. F. Cohen
L. Devroye
D. Dobkin
H. Edelsbrunner
G. Even
S. Fei
R. Fessler
S. Fogel
G. Frederickson
G. Gambosi
A. Garg
D. Geiger
O. Gerstel
J. Gilbert
M. Goodrich
G. Grahne
R. Grossi
N. V. Hai
X. He
F. Huber
E. Ihler
H. Imai
G. Kant
M. van Kreveld
D. T. Lee
W. Lenhart
S. Leonardi
G. Liotta
E. Lodi
K. Lyons
L. Malmi
L. Margara
R. H. Möhring
C. Montangero
A. Monti
S. Näher
E. Nuutila
P. Orponen
E. Otoo
M.C. Pinotti
K. Pollari-Malmi
R. Ravi
H. Ripphausen-Lipa
K. Romanik
T. Roos
M. Schäffter
P. Scheffler
A. Schuster
Y. Shen
M. Smid
S. Subramanian
S. Suri
S. Tate
I. G. Tollis
J. Vilo
F. Wagner
K. Weihe
M. Wloka

TABLE OF CONTENTS

Invited Presentations

Regular Presentations

Computing the All-Pairs Longest Chains in the Plane*

Mikhail J. Atallah
Dept. of Computer Science
Purdue University
West Lafayette, IN 47907.
E-mail: mja@cs.purdue.edu.

Danny Z. Chen
Department of Computer
Science and Engineering
University of Notre Dame
Notre Dame, IN 46556.
E-mail: chen@cse.nd.edu.

Abstract

Many problems on sequences and on circular-arc graphs involve the computation of longest chains between points in the plane. Given a set S of n points in the plane, we consider the problem of computing the matrix of longest chain lengths between all pairs of points in S, and the matrix of "parent" pointers that describes the n longest chain trees. We present a simple sequential algorithm for computing these matrices. Our algorithm runs in $O(n^2)$ time, and hence is optimal. We also present a rather involved parallel algorithm that computes these matrices in $O(\log^2 n)$ time using $O(n^2/\log n)$ processors in the CREW PRAM model. These matrices enables us to report, in $O(1)$ time, the length of a longest chain between any two points in S by using one processor, and the actual chain by using k processors, where k is the number of points of S on that chain. The space complexity of the algorithms is $O(n^2)$.

1 Introduction

Problems that involve longest increasing subsequences of a given sequence of numbers have attracted a lot of attention in the past. Probably the most studied version is that of the *longest increasing subsequence* (LIS), for which many $O(n \log n)$ time algorithms are known (e.g., [10, 11, 13], and many others). There is also a well-know connection between increasing subsequences and problems on certain specialized classes of graphs such as permutation graphs, circle and circular-arc graphs, and interval graphs (see, e.g., [12]–[19]). This paper considers the all-pairs version of the problem, whose formulation we state precisely next. We have chosen to formulate it as a problem on

*This research was supported by the Leonardo Fibonacci Institute in Trento, Italy, and by the National Science Foundation under Grant CCR-9202807. Part of this research was done while the first author was visiting LIPN, Paris, France.

points in the plane because our solution techniques are drawn from computational geometry; in terms of sequences, the y coordinate of a planar point corresponds to the value of an entry in the sequence, and the position at which that entry occurs in the sequence is determined by the x coordinate.

A point p is said to *dominate* another point q iff $X(p) \geq X(q)$ and $Y(p) \geq Y(q)$, where $X(p)$ and $Y(p)$ respectively denote the x and y coordinates of point p. Let S be a collection of n points in the plane, and let $\sigma = (p_1, \ldots, p_k)$ be a sequence of points such that each p_i is in S. The sequence σ is *increasing* iff p_i dominates p_{i-1} for all $1 < i \leq k$; such a sequence is called a *chain*, and we say that it *begins* at p_1, that it *ends* at p_k, and that its *length* is k (if the points are weighted then the length of σ is the sum of the weights of its points). The chain σ is *longest* if no other chain starting at p_1 and ending at p_k has greater length than σ.

The problem we consider is that of computing the $n \times n$ matrix D of the lengths of longest chains between pairs of points in S; that is, $D(p, q)$ is equal to the length of a longest p-to-q chain. By convention, for $p \neq q$, if q does not dominate p then $D(p, q) = -\infty$. We also compute an $n \times n$ matrix P (shorthand for "parent") such that $P(p, q)$ is the successor of p in some p-to-q longest chain.

We give a simple $O(n^2)$ time sequential algorithm in the unweighted case. Clearly, knowing P allows one processor to trace a longest p-to-q chain in time proportional to its length.

In parallel, we solve the weighted version of the problem in $O((\log n)^2)$ time using $O(n^2/\log n)$ processors in the CREW PRAM model. We also show that a longest p-to-q chain can be obtained in $O(1)$ time by using k CREW PRAM processors, where k is the number of points of S on that chain. The parallel algorithm bears very little resemblance to the sequential one, which seems hard to "parallelize". It also solves a more general (weighted) version of the problem.

An $O(n^2 \log n)$ time sequential algorithm for this problem is quite trivial to obtain, and to the best of our knowledge this was the best previously known bound for this all-pairs version of the problem. There is a published $O(n^2)$ time algorithm [2] for a special case of this problem: that for chains that start in S and end on a set of points that lie on a vertical line V, where V is to the right of S. In parallel, bounds similar to ours were only known for the special case of the layers of maxima problem, which can be viewed as the version of our problem where the chains of interest begin in S but must end at the point $(+\infty, +\infty)$ [1]. It is actually quite easy to use the methods of [1, 3] to solve the version of the problem where the chains of interest begin in S but must end on a set of points on a vertical line V that is to the right of S.

We now briefly discuss how our approach differs from the one for the above-mentioned special version of the problem, in which all chains start in S and end on a set of points on a vertical line V that is to the right of S. That special version of the problem is substantially easier, both sequentially and in parallel, because for a fixed $p \in S$, the collection of longest chains that begin at p and end on V have the following monotonicity property: Two such longest chains that end at (respectively) q' and q'', $Y(q') < Y(q'')$, can always be chosen such that nowhere is the chain to q'

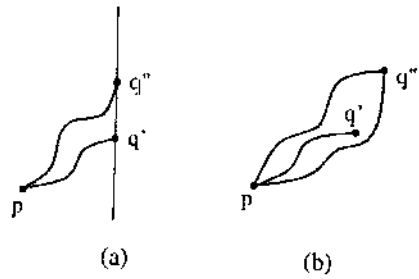


Figure 1: Illustrating (a) how monotonicity holds for some chains, and (b) how it fails to hold for others.

higher (geometrically) than the chain to q'' (intuitively, if it is higher then there is a crossing between the two chains and we can "uncross" them). Figure 1(a) illustrates this. Such a monotonicity property is lacking in the general version of the problem considered here: If q' and q'' do not lie on the same vertical line (see Figure 1(b)) then monotonicity need not hold, in the sense that either one of the two p-to-q'' chains shown could be a unique longest chain to q'', so that such a chain to q'' might go either "above" or "below" a longest p-to-q' chain.

We are unable to obtain an $O(n^2)$ time sequential solution to the weighted version of this problem, and we leave this as an open problem. Our parallel bounds, on the other hand, hold for the weighted version of the problem.

The rest of the paper is organized as follows. Section 2 deals with the sequential algorithm, which is fairly simple. Section 3 gives the parallel algorithm. We have chosen to give the basic terminology and definitions separately for each of the parallel and sequential algorithms, since they have little in common (this way the reader interested in one of the two will not be forced to read material unrelated to her interest). Section 4 concludes by posing some open problems.

2 The Sequential Algorithm

This section gives the $O(n^2)$ time sequential algorithm.

2.1 Preliminaries

The input consists of set S of n points in the plane. For a point $p \in S$, we use $DOM(p)$ to denote the subset of points in S that are dominated by p. A point p of S is a *maximum* in S iff no other point of S dominates p. We use $MAX(S)$ to denote

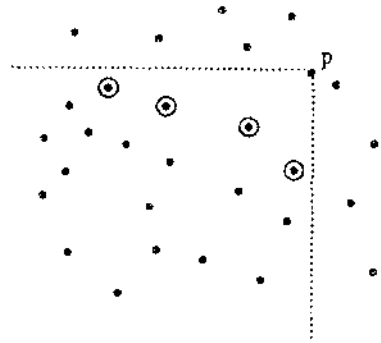


Figure 2: The points of $MD(p)$ are circled.

the set of maxima of S, listed by increasing x coordinates (and hence by decreasing y coordinates). We abbreviate $MAX(DOM(p))$ as $MD(p)$. Figure 2 illustrates the definition of $MD(p)$.

For a point $p \in S$, imagine partitioning $DOM(p) \cup \{p\}$ into k subsets, where $k = \max\{D(q,p) | q \in DOM(p) \cup \{p\}\}$, such that the points in each subset all have longest chains to p of the same length. The subset of $DOM(p) \cup \{p\}$ whose points have a distance j to p is called the j-th *domination layer* of p, denoted by $Layer_j(p)$. For example, $Layer_1(p) = \{p\}$, $Layer_2(p) = MD(p)$, and so on. In general, for each j, $Layer_j(p) = MAX(DOM(p) \cup \{p\} - \cup_{i<j} Layer_i(p))$. We assume that each layer of p is sorted by increasing x-coordinates (hence by decreasing y-coordinates).

It should be clear that, if we were able to compute the domination layers of each $p \in S$, then we would effectively have computed the desired D matrix. Our sequential algorithm will therefore mainly concern itself with the computation of these domination layers and of the P matrix. (The parallel algorithm deals with the weighted version and will use a different approach — in fact most of the definitions given above will not be used in the parallel algorithm.)

2.2 The Algorithm

Below is a high-level description of the sequential algorithm. We are assuming that no two points in S have the same x (resp., y) coordinate, i.e., that if $p, q \in S$ and $p \neq q$ then $X(p) \neq X(q)$ and $Y(p) \neq Y(q)$ (the algorithm can easily be modified for the general case). By convention, walking *forward* (resp., *backward*) along an $MD(p)$ means moving along it by increasing (resp., decreasing) x-coordinates.

Step 1. We first compute $MD(p)$ for every $p \in S$. These $MD(p)$'s can all be easily computed in $O(n^2)$ time as follows. We sort the points by their x coordinates, and then for each $p \in S$ we do the following. From the sorted list we obtain $DOM(p)$, in $O(n)$ time. Then we obtain the maximal elements of $DOM(p)$, also in $O(n)$ time (this

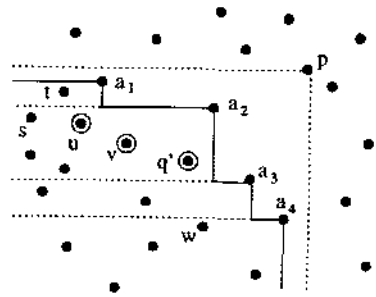


Figure 3: The points of $Interval_{j+1}(p, a_2)$ are shown circled.

is possible since $DOM(p)$ is available sorted). These maximal elements of $DOM(p)$ are, by definition, $MD(p)$.

Step 2. We compute, for each pair $p, q \in S$, the position of $Y(p)$ in the list $Y(MD(q))$, which is the list obtained from $MD(q)$ by replacing every point by its y-coordinate. This is easy to do in $O(n)$ time for a particular $Y(MD(q))$ and all $p \in S$, by merging $Y(MD(q))$ with the sorted list of the y coordinates of the points in S. Doing this once for each $q \in S$ takes a total of $O(n^2)$ time.

Step 3. For each $p \in S$, we obtain the domination layers of p and the column that corresponds to p in the P matrix. We do this in $O(n)$ time for each p, as follows. Clearly, we already have $Layer_1(p)$ $(= \{p\})$ and $Layer_2(p)$ $(= MD(p))$. We obtain $Layer_{j+1}(p)$ from $Layer_j(p)$ in $O(|Layer_j(p)| + |Layer_{j+1}(p)|)$ time, as follows. Let $Layer_j(p) = (a_1, a_2, \ldots, a_k)$, where $X(a_1) < X(a_2) < \cdots < X(a_k)$. We shall walk along the $Layer_j(p)$ list, creating the $Layer_{j+1}(p)$ list as we go along, in left to right order. When we reach a_i while scanning $Layer_j(p)$, we compute the portion of $Layer_{j+1}(p)$ that is in $MD(a_i)$ but not in $DOM(a_{i+1})$; we call this portion $Interval_{j+1}(p, a_i)$ (it forms a contiguous interval of $MD(a_i)$). Figure 3 illustrates the definition of $Interval_{j+1}(p, a_i)$. Note how, in that figure, point w is in $Layer_{j+1}(p) \cap MD(a_2)$ but not in $Interval_{j+1}(p, a_2)$. We shall compute $Interval_{j+1}(p, a_1)$, $Interval_{j+1}(p, a_2)$, $\ldots$, $Interval_{j+1}(p, a_k)$, in that order. While doing this, we maintain a variable called *cutoff* whose significance is that, when we finish processing a_i, *cutoff* contains the rightmost point in $\cup_{1 \le \ell \le i} Interval_{j+1}(p, a_\ell)$; intuitively, *cutoff* is the "dominant" point among those in $\cup_{1 \le \ell \le i} Interval_{j+1}(p, a_\ell)$ as far as the (yet to be computed) lists $Interval_{j+1}(p, a_{i+1}), \ldots, Interval_{j+1}(p, a_k)$ are concerned. In Figure 3, after $Interval_{j+1}(p, a_1)$ is computed, *cutoff* is point t, and after $Interval_{j+1}(p, a_2)$ is computed *cutoff* is point q'.

To determine $Interval_{j+1}(p, a_1)$, we simply start at the beginning of $MD(a_1)$ and walk forward along $MD(a_1)$ until we first reach a point $q \in MD(a_1)$ for which

$Y(q) < Y(a_2)$ (we do not count q as being part of our "walk" along $MD(a_1)$). The (possibly empty) portion of $MD(a_1)$ so traced is obviously equal to $Interval_{j+1}(p, a_1)$. If $Interval_{j+1}(p, a_1)$ is not empty then we set *cutoff* equal to the predecessor of q in $MD(a_1)$, otherwise it remains undefined. For the example shown in Figure 3, $q = u$ and $cutoff = t$. We then proceed to process a_2.

If *cutoff* is undefined (i.e., if $Interval_{j+1}(p, a_1)$ turned out to be empty) then we process a_2 exactly as explained above for a_1. Otherwise we process it as follows. Recall that we already know, from Step 2, the outcome of a hypothetical binary search for $Y(a_3)$ in $Y(MD(a_2))$: Let q' be the predecessor of $Y(a_3)$ in $Y(MD(a_2))$, that is, the lowest point of $MD(a_2)$ whose y-coordinate is larger than $Y(a_3)$. If no such point q' exists on $MD(a_2)$ then surely $Interval_{j+1}(p, a_2)$ is empty and we move on to processing a_3 (leaving *cutoff* unchanged). So suppose that q' exists. If $X(q') < X(cutoff)$ then $Interval_{j+1}(p, a_2)$ is empty and we move on to processing a_3 (leaving *cutoff* unchanged). If $X(q') > X(cutoff)$ then we start at q' and walk backward along $MD(a_2)$ until we reach a point whose x-coordinate is less than $X(cutoff)$; the portion of $MD(a_2)$ so traced is $Interval_{j+1}(p, a_2)$. In Figure 3, the portion so traced is (in that order) q', v, u (point s is not traced because $X(s) < X(t)$). In that case we also update *cutoff* by setting it equal to q' before we proceed to process a_3.

We process $a_3, a_4, \ldots, a_k$ in that order, exactly as explained above except that, when processing a_{i+1}, a_i plays the role of a_1, a_{i+1} plays the role of a_2, and a_{i+2} plays the role of a_3.

Once we have obtained $Layer_{j+1}(p)$ from $Layer_j(p)$, we must compute $P(w, p)$ for every $w \in Layer_{j+1}(p)$ (clearly, such a $P(w, p)$ is in $Layer_j(p)$). This is easily done for all $w \in Layer_{j+1}(p)$ in $O(|Layer_j(p)| + |Layer_{j+1}(p)|)$ time, by merging the two lists $Layer_{j+1}(p)$ and $Layer_j(p)$.

This completes the description of the sequential algorithm. We now turn our attention to the parallel algorithm.

3 The Parallel Algorithm

This section gives the $O((\log n)^2)$ time, $O(n^2/\log n)$ processor algorithm for the weighted version of the problem.

3.1 Preliminaries

The parallel model used is the CREW PRAM, which is the synchronous shared-memory model where concurrent reads are allowed, but no two processors can simultaneously attempt to write in the same memory location (even when they are trying to write the same thing). In what follows, we shall focus on showing that the claimed time complexity can be achieved with an $O(n^2 \log n)$ amount of *work* (= number of operations). This will imply the $O(n^2/\log n)$ processor bound, by Brent's theorem [8]:

Theorem 1 (Brent) *Any synchronous parallel algorithm taking time* T *that consists of a total of* W *operations can be simulated by* P *processors in time* $O((W/P) + T)$.

Remark: There are actually two qualifications to the above Brent's theorem before one can apply it to a PRAM: (i) at the beginning of the i-th parallel step, we must be able to compute the amount of work W_i done by that step, in time $O(W_i/P)$ and with P processors, and (ii) we must know how to assign each processor to its task. Both qualifications (i) and (ii) to the theorem will be easily satisfied in our algorithms, therefore the main difficulty will be how to achieve W operations in time T.

Here as in [4], an important method we use involves multiplying special kinds of matrices. Although the situation depicted in Figure 1(b) implies that the structure that gives rise to such matrices is not always available, the fact that we can deal with the situation in Figure 1(a) will be useful. (This will all be made precise later; for now we are only giving an overview.) All matrix multiplications in the algorithm are in the $(\max, +)$ closed semi-ring, i.e., $(M' * M'')(i,j) = \max_k \{M'(i,k) + M''(k,j)\}$. A matrix M is said to be *Monge* [1] iff for any two successive rows $i, i+1$ and columns $j, j+1$, we have $M(i,j) + M(i+1,j+1) \leq M(i,j+1) + M(i+1,j)$. For two point sets A and B in the plane, let matrix M_{AB} contain the lengths of the longest chains that start in A and end in B (by convention, these chains are allowed to go through any points of S on their way from A to B). Now, consider two point sets X and Y, each totally ordered in some way (so we can talk about the predecessor and successor of a point in X or in Y), and such that the rows (resp., columns) of the matrix M_{XY} are as in the ordering for X (resp., Y). Matrix M_{XY} is *Monge* iff for any two successive points p, p' in X and two successive points q, q' in Y, we have $M_{XY}(p,q) + M_{XY}(p',q') \leq M_{XY}(p,q') + M_{XY}(p',q)$. The next lemma characterizes the Monge matrices of chain lengths used in the algorithm.

Lemma 1 *Let V' (resp., V'') be a vertical line that contains a set X (resp., Y) of points ordered by increasing (resp., decreasing) y-coordinates along V' (resp., V''). (Assume that V' is to the left of V''.) Then the matrix M_{XY} of chain lengths between X and Y is Monge.*

Proof. Obvious. □

The following well-known lemma [3, 1] will be used.

Lemma 2 *Assume that matrices M_{XY} and M_{YZ} are Monge, with $|X| = c_1|Y| \leq c_2|Z|$ for some positive constants c_1 and c_2. Then $M_{XY} * M_{YZ}$ can be computed in $O(\log|Y|)$ time and $O(|X||Z|)$ work in the CREW PRAM model.*

Remark: Since $*$ is a $(\max, +)$ matrix multiplication, $M_{XY} * M_{YZ}$ need not be Monge.

Lemmas 1 and 2 imply the following.

Lemma 3 *Let V (resp., V', V'') be a vertical line that contains a set X (resp., Y, Z) of points ordered by increasing y-coordinates along V (resp., V', V''). Assume that $X(V) < X(V') < X(V'')$, and that $|X| = c_1|Y| \leq c_2|Z|$ for some positive constants*

c_1 and c_2. Suppose that, for every increasing chain C from $p \in X$ to $q \in Z$, there is a p-to-q chain C' that is at least as long as C and goes through some $w \in Y$. Then given the matrices M_{XY} and M_{YZ}, the matrix M_{XZ} can be computed in $O(\log |Y|)$ time and $O(|X||Z|)$ work in the CREW PRAM model.

Proof. Let X' (resp., Y', Z') be the same as X (resp., Y, Z) but sorted by decreasing y coordinates. By Lemma 1, $M_{XY'}$ and $M_{Y'Z}$ are both Monge. By Lemma 2, the matrix $M_{XY'} * M_{Y'Z}$ can be computed in $O(\log |Y|)$ time and $O(|X||Z|)$ work. Now, since by hypothesis all the X-to-Z chains can be modified to go through Y without any decrease in their lengths, it follows that the matrix $M_{XY'} * M_{Y'Z}$ is the desired matrix M_{XZ}. □

3.2 The Algorithm for Chain Lengths

The algorithm given in this subsection concerns itself with the computation of chain lengths only, not of the P matrix that describes the n longest chain trees. Including the computation of P here would have cluttered the exposition. The next subsection will deal with the computation of P. In addition, it is not immediately clear that the availability of P makes possible the reporting of a k-point chain in $O(k)$ work and constant time. This too is postponed until the next subsection.

Let $S = \{p_1, \ldots, p_n\}$ where $X(p_1) < \cdots < X(p_n)$. There is a weight associated with each p_i. Let $V_0, V_1, \ldots, V_n$ be vertical lines such that $X(V_0) < X(p_1)$, $X(p_n) < X(V_n)$, and $X(p_i) < X(V_i) < X(p_{i+1})$ for all $i \in \{1, \ldots, n-1\}$.

Let T be a complete n-leaf binary tree. For each leaf v of T, if v is the i-th leftmost leaf in T, then associate with v the region I_v of the plane that is between V_{i-1} and V_i. For each internal node v of T, associate with v the region I_v consisting of the union of the regions of its children. That is, if v has children u and w, then $I_v = I_u \cup I_w$.

Let v be any node of T. Suppose that the left (resp., right) boundary of I_v is V_i (resp., V_j), and let $S_v = S \cap I_v$, that is, S_v is the subset of the input points that lie in I_v. Observe that if v is at a height of h in T then $j - i = 2^h = |S_v|$ (the height of v is the height of its subtree in T, with leaves being at a height of zero). Let L_v (resp., R_v) be the set of points on V_i (resp., V_j) that are the horizontal projections of S_v on V_i (resp., V_j). The points of L_v and R_v are, of course, disjoint from the input set S, and we assign to each of them a weight of zero. Observe that

$$\sum_{v \in T} |L_v| = \sum_{v \in T} |R_v| = O(n \log n),$$

because for each level of T a $p_i \in S$ appears in exactly one S_v of that level, and hence creates at most two extra points, one in L_v and one in R_v (recall that a level of T is the set of nodes in T that have same distance to the root, so that the root is at level zero, its two children at level 1, etc).

There are two phases for the algorithm: Phase 1 is relatively straightforward, while Phase 2 is the key that made our solution possible.

3.2.1 Phase 1

This phase consists of computing, starting at the leaves and going upward in T, one level at a time, the $M_{L_v R_v}$ matrices, which contain the lengths of all the L_v-to-R_v longest chains (chains that begin on L_v and end on R_v, of course possibly going through points in S_v along the way). This information is trivially available if v is a leaf. So suppose that we have already computed this information for level $\ell+1$, and we want to compute it for level ℓ.

We claim that it suffices to show that the $M_{L_v R_v}$ matrix can be computed in $O(|S_v|^2)$ work and $O(\log n)$ time for each node v at level ℓ. This claim would imply an $O((\log n)^2)$ time, $O(n^2)$ work bound for Phase 1, as follows. That the time bound follows from the claim is obvious (we would spend a logarithmic amount of time per level, and there is a logarithmic number of levels). The work bound would follow from the fact that there are 2^ℓ nodes v at level ℓ, each having $|S_v| = n/2^\ell$, and hence the total work at level ℓ would be

$$O(2^\ell (n/2^\ell)^2) = O(n^2/2^\ell).$$

Summing over all levels ℓ gives $O(n^2)$ total work. We next prove the claim by showing that the $M_{L_v R_v}$ matrix can indeed be computed in $O(|S_v|^2)$ work and logarithmic time.

Let u (resp., w) be the left (resp., right) child of v in T. Let Y denote $R_u \cup L_w$, that is, Y consists of the horizontal projections of the points of S_v on the vertical line V_j that separates the region I_u from the region I_w. Since $M_{L_u R_u}$ is already available at u, we can easily obtain from it $M_{L_u Y}$. Similarly, we obtain $M_{Y R_v}$ from $M_{L_w R_w}$. Now, simply observe that the conditions for Lemma 3 are satisfied, with L_v playing the role of X and R_v the role of Z. That is, we can obtain $M_{L_v R_v}$ from $M_{L_v Y}$ and $M_{Y R_v}$ in $O(|S_v|^2)$ work and logarithmic time. This completes the proof.

Remark: The astute reader may have observed that the above procedure can be modified so as to compute the L_v-to-S_v and S_v-to-R_v chain lengths information. This would involve only a logarithmic factor of additional work, and would exploit the kind of monotonicity depicted in Figure 1(a) by using the lower-dimensional parallel matrix searching algorithm of [5]. However, this would still leave us far from having solved our problem: We would still need something like Phase 2 below, since we cannot afford to multiply "non-square" Monge matrices — as of now, it is not known how to optimally $(\max, +)$-multiply two non-square Monge matrices (for example, the best parallel algorithm for multiplying a $1 \times k$ Monge matrix with a $k \times k$ one in logarithmic time takes $O(k \log k)$ work [5]). Observe how Phase 2 below will satisfy the size requirements of Lemma 3, as expressed in the requirement that $|X| = c_1|Y| \leq c_2|Z|$ for some positive constants c_1 and c_2.

3.2.2 Phase 2

Whereas Phase 1 involved a bottom-up computation in T, Phase 2 will involve a top-down computation, starting at the root and proceeding one level at a time until we reach the leaves. The purpose of the computation at a typical level ℓ is more

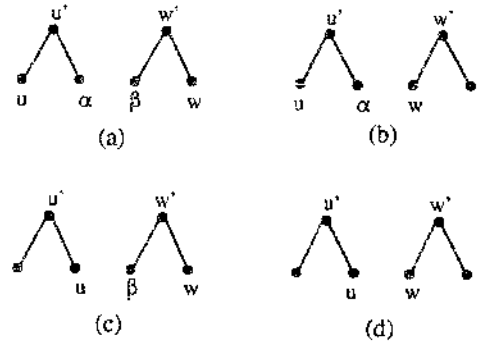


Figure 4: Illustrating the four cases of Phase 2.

ambitious than in Phase 1: We now seek, for every pair of nodes u, w at level ℓ such that u is to the left of w, the computation of the $M_{R_u L_w}$ chain lengths matrix (u is to the left of w iff it is in the subtree of the left child of the lowest common ancestor of u and w). The key idea is to get help from the parents of u and w, which we call u' and (respectively) w'. If $u' = w'$ then the desired information is trivially available, so suppose that $u' \neq w'$. We distinguish four cases, which are illustrated in Figure 4.

Case 1: u is the left child of u', and w is the right child of w'. Let α be the right child of u', β be the left child of w' (see Figure 4(a)). Since Phase 1 computed $M_{L_\alpha R_\alpha}$, we can obtain from it $M_{R_u R_\alpha}$, then $M_{R_u R_{u'}}$. Similarly, we obtain $M_{L_{w'} L_w}$ from $M_{L_\beta R_\beta}$ which was computed in Phase 1. Now, $M_{R_{u'} L_{w'}}$ is already available because Phase 2 is already done with processing the pair u', w' (recall that Phase 2 processes the levels from the root down). We use Lemma 3 to obtain the matrix $M_{R_u L_{w'}}$ from $M_{R_u R_{u'}}$ and $M_{R_{u'} L_{w'}}$, with R_u playing the role of X, $R_{u'}$ playing the role of Y, and $L_{w'}$ playing the role of Z. Finally, we use Lemma 3 again, this time to obtain the desired matrix $M_{R_u L_w}$ from $M_{R_u L_{w'}}$ and $M_{L_{w'} L_w}$.

Case 2: u is the left child of u', and w is the left child of w'. Let α be the right child of u' (see Figure 4(b)). From the $M_{L_\alpha R_\alpha}$ matrix which was computed in Phase 1, we obtain the $M_{R_u R_{u'}}$ matrix. Observe that $M_{R_{u'} L_{w'}}$ was already obtained earlier in Phase 2: get from it the $M_{R_{u'} L_w}$ matrix. We use Lemma 3 to obtain $M_{R_u L_w}$ from the matrices $M_{R_u R_{u'}}$ and $M_{R_{u'} L_w}$.

Case 3: u is the right child of u', and w is the right child of w'. Let β be the left child of w' (see Figure 4(c)). From the $M_{R_{u'} L_{w'}}$ matrix which was computed earlier in Phase 2, obtain the $M_{R_u L_\beta}$ matrix. From the $M_{L_\beta R_\beta}$ matrix which was computed in Phase 1, we obtain the $M_{L_\beta L_w}$ matrix. We use Lemma 3 to obtain $M_{R_u L_w}$ from $M_{R_u L_\beta}$ and $M_{L_\beta L_w}$.

Case 4: u is the right child of u', and w is the left child of w' (see Figure 4(d)). Obtain $M_{R_u L_w}$ from $M_{R_{u'} L_{w'}}$ which was computed earlier in Phase 2.

The time taken by Phase 2 is clearly $O((\log n)^2)$, since we take a logarithmic amount of time at leach level of T. The work done for a particular pair u, w at level ℓ is $O((n/2^\ell)^2)$, and since there are $(2^\ell)^2$ such pairs at level ℓ the total work done at that level is $O(n^2)$. Summing over all levels gives $O(n^2 \log n)$ work for Phase 2. Hence it is Phase 2 that is the bottleneck in the work complexity. The space taken by Phase 2 is still $O(n^2)$ rather than $O(n^2 \log n)$, however, since we do not need to store the matrices for all the levels as Phase 2 proceeds: When we are done with level ℓ, we can discard the matrices for level $\ell - 1$ since level $\ell + 1$ will only need information from level ℓ (recall that in Phase 2 the nodes of T request help only from their parents, not from their grandparents or from higher up in the tree T).

3.3 Computing the Actual Chains

In this subsection, we discuss how to obtain the matrix P which contains the n trees of longest chains, and how to pre-process the longest chain trees, so that each tree can support a longest chain query between any point in S and the point of S at the root of that tree.

First we sketch how the algorithm in the previous subsection can be modified so as to compute the P matrix as well. For each $M_{R_u L_w}$ matrix computed by that algorithm, we compute a companion $P_{R_u L_w}$ matrix whose significance is that, for $p \in R_u$ and $q \in L_w$, $P_{R_u L_w}(p, q)$ is the first point of S that lies on a longest p-to-q chain (it is undefined if no such point of S exists). Note that only points of S can be "parents". It is quite easy to modify the computation of an M_{XZ} so that it also produces P_{XZ}: If M_{XZ} is obtained by using Lemma 3, then P_{XZ} can be obtained from P_{XY} or P_{YZ} as a "byproduct" of this computation. For example, if q dominates p and if $M_{XZ}(p, q) = M_{XY}(p, t) + M_{YZ}(t, q)$, then we distinguish two cases for obtaining $P_{XZ}(p, q)$: If $P_{XY}(p, t)$ is undefined then $P_{XZ}(p, q) = P_{YZ}(t, q)$ (which could also be undefined), otherwise $P_{XZ}(p, q) = P_{XY}(p, t)$. When the modified algorithm finishes computing $P_{R_u L_w}$ for all leaves u, w (at the end of Phase 2), it is easy to obtain the matrix P: If $S_u = \{p_i\}$, $R_u = \{p'_i\}$, $S_w = \{p_j\}$, $L_w = \{p'_j\}$, then we set $P(p_i, p_j)$ equal to $P_{R_u L_w}(p'_i, p'_j)$ if the latter is defined; otherwise, we set $P(p_i, p_j)$ equal to p_j if p_j dominates p_i, and set $P(p_i, p_j)$ to be undefined if p_j does not dominate p_i. From now on, we assume that the matrix P is available. Note that this matrix is a description of n longest chain trees, each rooted at a point of S.

We pre-process each longest chain tree so that the following type of queries can be quickly answered: Given a node p in the tree and a positive integer i, find the i-th node on the path from p to the root of the tree. Such queries are called *level-ancestor queries* by Berkman and Vishkin [6], who gave efficient parallel algorithms for pre-processing rooted trees so that the level-ancestor queries can be answered quickly. The work of Berkman and Vishkin [6, 7] shows (implicitly) that a level-ancestor query can be handled sequentially in constant time, after a logarithmic time and linear work

pre-processing in the CREW PRAM model. The pre-processing of the longest chain trees is done by simply applying the result of Berkman and Vishkin to each of the n trees, in totally $O(\log n)$ time and $O(n^2)$ work.

For the sake of processor assignment in reporting chains, we also need to compute the number of points of S on the actual longest chain which is to be reported. Suppose a longest chain between points p and q in S is to be reported. The number of points of S on such a p-to-q chain can be obtained from the depth of p in the longest chain tree rooted at q; it is known that the depths can be computed within the required complexity bounds by using the Euler Tour technique [20].

To report an actual longest chain between points p and q in S, we do the following (without loss of generality, we assume that q dominates p). First, we go to the longest chain tree rooted at (say) q, and find the number of nodes on the path in that tree from node p to the root q. Let that number be k. The p-to-q path in the tree corresponds to a longest chain from p to q, which we must report. We do so by performing, in parallel, $k-1$ level-ancestor queries, using node p and integers $1, 2, \ldots, k-1$. Each query is handled by one processor in $O(1)$ time. These queries find each point on the p-to-q chain. Finally, we report the k points of that chain in parallel, by assigning to k processors the task of reporting those k points (one point per processor).

4 Further Remarks

The following open problems remain:

- Give an $O(n^2)$ time sequential algorithm for the weighted case.
- Give an $O(n^2)$ time sequential algorithm for the three dimensional version of the problem (unweighted).
- For the three dimensional version of the problem, give an NC parallel algorithm that uses a quadratic (to within a logarithmic factor) number of processors.

Finally, using the methods we developed here in combination with other ideas, we can improve the processor complexity of the layers of maxima problem: We can achieve the same $O((\log n)^2)$ time complexity as in [1] with $O(n^2/(\log n)^3)$ processors, instead of the $O(n^2/\log n)$ processors used in [1].

References

[1] A. Aggarwal and J. Park. "Notes on Searching in Multidimensional Monotone Arrays (Preliminary Version)," *Proc. 29th Annual IEEE Symposium on Foundations of Computer Science*, 1988, pp. 497–512.

[2] A. Apostolico, M. J. Atallah, and S. E. Hambrusch. "New Clique and Independent Set Algorithms for Circle Graphs," *Discrete Appl. Math.*, Vol. 36, 1992, pp. 1–24.