1 Overview

In this lecture, Prof. Matthew J.Katz presented the work on “Parametric Searching”, Megiddo 1983. He defined the problem, gave various solutions with analyses of them.

2 Introduction

Consider, e.g., the 2-center problem:

Let \( P \) be a set of \( n \) points in the plane. Find the minimum radius \( r^* \) such that \( P \) can be covered by two disks of radius \( r^* \).

Observation: \( r^* \) is determined by either 2 or 3 points in \( P \).

The corresponding decision problem: For a given value \( r \), determine whether \( P \) can be covered by two disks of radius \( r \). If yes then \( r^* \leq r \), and if no then \( r^* > r \).
Assuming we have an efficient solution to the decision problem, we would like to use it to find \( r^* \) in the set of all potential values. But this set is too large to generate explicitly !!!!

We present the parametric searching technique through an example.

**Problem:** Let \( Y_1, \ldots, Y_n \) be \( n \) lines in the plane, where \( Y_i = Y_i(\delta) = a_i \delta + b_i \), and \( a_i > 0 \), for \( i = 1, \ldots, n \). Define \( F(\delta) = \text{median} \{ Y_1(\delta), \ldots, Y_n(\delta) \} \), for \( \delta \in IR \). \( F \) is a piece-wise linear, increasing function with \( O(n^2) \) turns. Find the value \( \delta^* \), for which \( F(\delta) = 0 \), i.e., find the root of the equation \( F(\delta) = 0 \).

3 Different Solutions

3.1 A trivial solution

Find the median among the roots of the \( n \) functions \( Y_1, \ldots, Y_n \).

We shall find the function \( Y \) (among \( Y_1, \ldots, Y_n \)) that determines \( F \) at \( \delta^* \). Once we know \( Y \), we compute its root to obtain \( \delta^* \). There is a “slight” problem—\( \delta^* \) is unknown, so we do not have the values \( Y_1(\delta^*), \ldots, Y_n(\delta^*) \).
3.2 A first solution

We apply the algorithm of Blum et al. for finding the median number in a set of numbers (in our case \( \{ Y_1(\delta), ..., Y_n(\delta) \} \)). This algorithm is based on comparisons between pairs of numbers from the underlying set.

In order to compare between \( Y_i(\delta^*) \) and \( Y_j(\delta^*) \), i.e., to decide which of them is greater, we only need to determine the location of \( \delta^* \) with respect to the intersection point between the lines \( Y_i \) and \( Y_j \).

Assuming \( a_i > a_j \). If \( \delta^* \) lies to the left (resp. to the right) of \( \delta_{i,j} \), then \( Y_i(\delta^*) < Y_j(\delta^*) \) (resp. \( Y_i(\delta^*) > Y_j(\delta^*) \)).

But, how do we determine the location of \( \delta^* \) with respect to the intersection point \( \delta_{i,j} \)?

We compute \( F(\delta_{i,j}) \) by applying the median finding algorithm to the set \( \{ Y_1(\delta_{i,j}), ..., Y_n(\delta_{i,j}) \} \).

Now, since \( F \) is a monotone increasing function, if \( F(\delta_{i,j}) > 0 \), then \( \delta^* \) lies to the left of \( \delta_{i,j} \), and, if \( F(\delta_{i,j}) < 0 \), then \( \delta^* \) lies to the right of \( \delta_{i,j} \).

**Analysis:** The median finding algorithm performs \( O(n) \) comparisons (of the form \( Y_i(\delta^*) : Y_j(\delta^*) \)), and each of these comparisons is resolved by a call to the median finding algorithm with a concrete set of values (i.e., \( \{ Y_1(\delta_{i,j}), ..., Y_n(\delta_{i,j}) \} \)). Thus the total running time is \( O(n^2) \).
3.3 An improved solution

We replace the main median finding algorithm (that attempts to find the median line at \( \delta^* \)) with a parallel version of this algorithm.

Actually we apply a parallel sorting algorithm (to sort the lines at \( \delta^* \)), that uses \( n \) processors and sorts in \( O(\log n) \) (parallel) time.

In the first (parallel) step of this algorithm each of the \( n \) processors performs a comparison (of the form \( Y_i(\delta^*) : Y_j(\delta^*) \)).

We simulate this step sequentially. But, instead of calling the median finding algorithm for each of the \( n \) comparisons, we proceed as follows.

We first compute the \( n \) intersection points \( \delta_{i,j} \) corresponding to the \( n \) comparisons. Let \( \delta_1 < \delta_2 < \ldots < \delta_n \) be these intersection points.

We now (binary) search for \( \delta^* \) in the sorted list \( \delta_1, \delta_2 \ldots \delta_n \). Each comparison of the form \( \delta^* : \delta_i \) is resolved by a call to the median finding algorithm.

Once we have located \( \delta^* \) in the sorted list \( \delta_1, \delta_2 \ldots \delta_n \), we can easily resolve all \( n \) comparisons assigned to the \( n \) processors, and proceed to the next parallel step.

Notice that in each step we further restrict the range in which \( \delta^* \) is known to lie.

At the end we obtain the lines sorted by their value at \( \delta^* \); we compute the root of the median line in this list to obtain \( \delta^* \).

**Analysis:** For each parallel step we perform \( O(\log n) \) “expensive” (i.e., linear-time) comparisons. Since there are \( O(\log n) \) parallel steps, the total time required for all “expensive” comparisons is \( O(n\log^2 n) \). The additional time required for the simulation of the parallel algorithm is \( O(n\log n) \).

4 Other Problems

4.1 A formal description of parametric searching

A problem that receives as input \( n \) data items and a real-valued parameter \( \delta \). (\( P(\delta) \equiv F(\delta) \)).

We need to find a value \( \delta \) for which \( P(\delta) \) is “special”. E.g., the output of \( P(\delta) \) is a real
number and \( P(\delta) = 0 \) or \( P(\delta) \) is an extreme value.

Assume we have an efficient sequential algorithm \( A_s \) for solving \( P(\delta) \) when given \( \delta \). Further assume that \( A_s \) can determine whether a given \( \delta \) is smaller than, larger than, or equal to \( \delta^* \). \( (A_s \equiv \text{the median finding algorithm applied to } \{Y_1(\delta), \ldots, Y_n(\delta)\}. ) \)

We also assume that the flow of \( A_s \) depends on comparisons, and that each such comparison can be resolved by checking the sign of a small-degree polynomial in the data items and \( \delta \).

\( \text{(The comparison } Y_i(\delta) : Y_j(\delta) \text{ is resolved by checking the sign of the polynomial } a_i\delta + b_i - (a_j\delta + b_j).) \)

Let \( A_p \) denote a parallel version of \( A_s \) (or more generally a parallel algorithm that solves \( P(\delta) \)), and let \( P \) denote the number of processors used by \( A_p \). \( (A_p \equiv \text{parallel sorting algorithm}) \)

Then \( \delta^* \) can be found in time \( O(T_p(P + T_s\log P)) \), where \( T_s \) is the running time of \( A_s \) and \( T_p \) is the (parallel) running time of \( T_p \).

**Remark:** Notice that the final algorithm is sequential. We use a parallel version of \( A_s \) only to ensure a small number of batches of independent comparisons.

### 4.2 Slope selection

**Problem:** Given a set \( H \) of \( n \) lines in the plane, and a number \( 1 \leq k \leq (\binom{n}{2}) \), find the \( k \)th vertex (from the left) of the arrangement \( A(H) \).

Let \( t_1 < t_2 < \ldots < t_{\binom{n}{2}} \) be the \( x \)-coordinates of the vertices of the arrangement \( A(H) \).

We shall use a parallel sorting algorithm \( A_p \) to sort the lines along a vertical line just to the right of \( t_k \).

In order to resolve a comparison between two lines, we first compute the \( x \)-coord \( x \) of their intersection point, and then compute the index of \( x \) (so that we know whether \( t_k \) lies to the left or to the right of \( x \)). The latter step is done in \( O(n\log n) \) time by counting the number of inversions in the permutation obtained by sorting the lines along a vertical line just to the right of \( x \).

The overall running time is therefore \( O(n\log^3 n) \). Can be improved to \( O(n\log n) \) using several tricks.
4.3 Alternative techniques

**Randomized halving** described by Matousek.

The technique of Frederickson and Johnson uses **sorted matrices**.

An **expander-based** technique (Katz and Sharir).