Computing Maximal Layers Of Points In $E^{f(n)}$

Indranil Banerjee, Dana Richards

George Mason University

ibanerje@cs.gmu.edu

April 11, 2016

Definitions

($p \succ q$)

For two points $p, q \in E^k$ we say p dominates q or p is above q iff $p[i] \geq q[i]\ \forall i \in [k]$.

Let $\boldsymbol{P = \{p_1, ..., p_n\}}$ with $p_i \in E^k$, we define the **First Maximal Layer** $M_1(P)$ of P as the set of points, that are not dominated by any other points.

The $\boldsymbol{h^{th}}$ **maximal layer** is defined recursively:

$$M_h(P) = M_1\left(P \setminus \bigcup_{i=1}^{h-1} M_i(P)\right)$$

Definitions

The **height** h **of** P is the number of non-empty (maximal) layers in P and the **width** w is the size of the largest **anti-chain** in $(P, \succ)$.

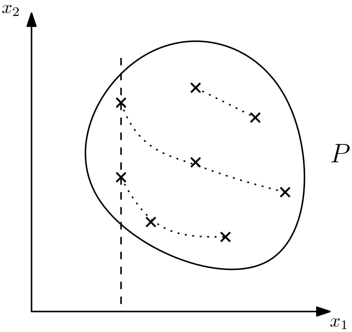


Figure: Dotted chains represent the maximal layers of the point set P.

Definitions
Random Orders

P is a random order if P is built by picking points uniformly at random from the unit k-cube $[0, 1]^k$.

Alternatively P is picked uniformly at random from S_n^k, set of all k-tuple of n-permutations

It can be shown that they are equivalent

Definitions

Random Orders

1. The expected height $\bar{h} = O(n^{1/k})$ and width $\bar{w} = O(n^{1-1/k})$
2. It is known that $M_1(P) = O((\log n)^{k-1})$ (for fixed k)
3. There is no known asymptotic formula for the size distribution of layers beyond the first layer.
4. We show (in ongoing work) the problem is closely related to determining the **longest increasing subsequence** of a random permutation.
5. For a random order $\boldsymbol{Prob[p[i] = q[i]] = 0}$.

Previous work

Deterministic Setting:(k is fixed)

1. In a seminal paper [Kung et al, 1975] proposed an algorithm for finding the first maximal layer in $O(n(\log n)^{k-2})$ time. This uses multi-dimensional divide-and-conquer approach.
2. [Jensen, 2003] extended this to compute all the layers in $O(n(\log n)^{k-1})$ time.

($k = n$)
[Matoušek, 1991] gave a $O(n^{1.5+\omega/2})$ algorithm for determining the first maximal layer.

Randomized Algorithm When P Is A Random Order

1. [Bentley, 1975] gave expected $O(n)$ algorithm for a fixed k

Our Results

For Random Orders

1. We introduce a new iterative framework for the maximal layers problem.
2. We assume that the dimension k may not be constant,
3. The proposed algorithm has an expected run-time of $\boldsymbol{O(kn^{2-\delta(k)})}$ $(\delta(k) > 0)$.
4. The factor k accounts for the fact that checking dominance requires $O(k)$ time
5. Later it is extended for an arbitrary P

A Simple Iterative Algorithm
For Random Orders

Main Steps:

1. Sort the points in descending order according to the L_∞ norm. (T)
2. We pick from T one element at a time in order.
3. Insert() the element in to the layer it belongs to.
4. End when T is empty.

Step 3 is achieved through a new data structure Hierarchical Search Tree (HST), for each layer.

A Simple Iterative Algorithm
For Random Orders

If p **precedes** q in T then q **cannot dominate** p.

Expected runtime $= n \times t_i + t_n$.
t_i = the expected time to insert an element in to its HST. (Line 3)
t_n = time to compute T

T can be computed in $O(kn + \log n)$, which we can ignore

A Simple Iterative Algorithm
Step-3: Insert()

We maintain each layer using an HST

1. Layers are totally ordered.
2. Hence, we can use a binary search tree (B), to maintain the sorted order
3. During Insert() we traverse B, querying corresponding HST to see if the current point is dominated by some point in that layers/HST
4. If no suitable HST is found, then we create an empty HST in B and insert the point

Hierarchal Search Tree (HST)

Definition

An HST Is Defined Recursively

1. A singleton node (root) storing a point p.
2. A root has a number of non-empty children nodes (up to k) each of which is a root of an HST.
3. If node q is in the j^{th} subtree of node p then $p[j] \geq q[j]$

HST is similar in principal to space partition trees.

Hierarchal Search Tree (HST)

Queries

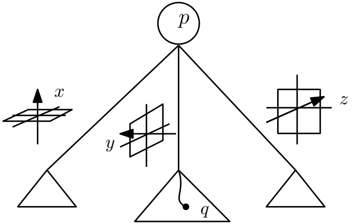


Figure: Example of an HST for the 3-dimensional case.

Above(): Checks whether the query point is dominated by some point inside the layer

Add(): Inserts the query point into the HST

Hierarchal Search Tree (HST)

Queries

Above()

1. Let q be the query point, let p be the point at the root
2. **if** $p \succeq q$ **then** stop (the current layer dominates q)
   else
   recursively search a subtree for each each half-space labeled j for which $p[j] \geq q[j]$

Hierarchal Search Tree (HST)
Queries

Add()

1. We start at the root, and proceed as we did in the case of Above()
2. If there are more than one choice of subtrees then we choose one uniformly at random
3. We stop and insert the point when we reach an empty leaf node.

Analysis

Lemma:
$\eta = Prob\{\ p[j] \geq q[j] \mid p \text{ precedes } q \text{ in } T\} = 1 - \frac{k-1}{2(k+1)}$

1. We see η does not depend on the relative ranks in T of the elements
2. $t_i = t_{search} + t_{add}$
3. t_{search} = the expected time it takes to search B to find the correct layer
4. t_{add} = the cost adding the element to an HST
5. $t_{search} \geq t_{add}$

Analysis

Computing t_{search}

1. First we compute $a_{m,d}$ the expected number of nodes at **depth d** of a HST with m nodes
2. Let $b_{m,k}$ = the number of nodes visited during Above()
3. We use $a_{m,d}$ to bound the expected value of $b_{m,k}$
4. Then $t_{search} = b_{m,k} \times \log h$

Analysis

Computing t_{search}

1. Recall $\eta = 1 - \frac{k-1}{2(k+1)}$ is the probability that a node p, already in some HST, dominates the query point q. Then,

$$b_{m,k} = \sum_{i=0}^{m-1} a_{m,i} \eta^i$$

2. We compute $a_{m,d}$ using the following recurrence:

$$a_{m,d} = \left(\frac{1}{k^{d-1}}\right) a_{m-1,d-1} + \left(1 - \frac{1}{k^d}\right) a_{m-1,d}$$

3. It is unlikely that the above recurrence has a closed form as the recurrence for the binomial coefficient is a special form of it.

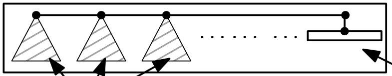
Analysis

Computing t_{search}

$$a_{m,d} = \left(\frac{1}{k^{d-1}}\right) a_{m-1,d-1} + \left(1 - \frac{1}{k^d}\right) a_{m-1,d}$$

1. Fortunately $a_{m,d}$ is unimodal and is bell-shaped with respect to d when w and k are fixed
2. Using bounds on the maxima of $a_{m,d}$ we show that
$$b_{m,k} = O(m^{1-\frac{1}{\log k}+\log_k(1+\frac{1}{k})})$$
3. $m, h \leq n$
4. $t_{search} = b_{m,k} \times \log h = O(kn^{1-\delta(k)} \log n)$
5. Total runtime $= O(kn^{2-\delta'(k)})$

When P Is Not Random
A List Of HSTs



1. The Above() is modified to iterate over this list of HSTs instead of a single one and the list R
2. When adding to the List-HST, we first check if R is full or not. If R is full we create an HST and add it to the list. Otherwise we just add the point to the R.

The General Case
Using A List Of HSTs

Creating an HST

We randomly permute the points in R before creating a new HST

So we can carry the same probabilistic assumptions from the previous case

1. It can be shown that SEARCH on the list of HSTs takes $O(kn^{1/2+(\log_k(k-1))/2})$
2. Which gives the total runtime of $O(k^2 n^{3/2+(\log_k(k-1))/2} \log n) = O(k^2 n^{2-\xi(k)})$
3. $0 < \xi(k) < 1/2$

Open Problems
Using A List Of HSTs

1. In the deterministic setting, can we reduce the upper bound to $o(n^2)$ independent of k.
2. Alternatively improve the trivial lower bound of $(n \log n)$
3. Given a random order, determine the expected value of $M_h(P)$.

Questions?

References

Winkler, P. (1985)
Random Orders
Order 1(4), 317 – 331.

Jensen M. T. (2003)
Reducing the Run-Time Complexity of Multiobjective EAs: The NSGA-II and Other Algorithms
IEEE TEC 7(5), 503 – 515.

Kung H. T., Luccio F., Preparata F. P. (1975)
On Finding the maxima of a Set of Vectors
Journal of ACM 22(4), 469 – 476.

Bentley J. L. (1980)
Multidimensional Divide-and-conquer
Commun. ACM 23(4), 214 – 229.

Matoušek, J. (1991)
Computing dominance in E^n
Information Processing Letters 38, 277 – 278 .