ON MAXIMAL LAYERS OF RANDOM ORDERS

by

Indranil Banerjee

A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Master of Science
Computer Science

Committee:

_____	Dr. Dana Richards, Thesis Director
_____	Dr. Zoran Duric, Committee Member
_____	Dr. Walter Morris, Committee Member
_____	Dr. Sanjeev Setia, Chairman, Department of Computer Science
_____	Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering

Date: _____ Fall Semester 2015
George Mason University
Fairfax, VA

On Maximal Layers Of Random Orders

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

By

Indranil Banerjee
Bachelor of Science
National Institute of Technology, Durgapur, India, 2009

Director: Dr. Dana Richards, Professor
Department of Computer Science

Fall Semester 2015
George Mason University
Fairfax, VA

Dedication

I dedicate this thesis to my grandfather, who introduce me to the wonderful world of counting.

Acknowledgments

This thesis would not have been made possible without my advisor Dana Richards' mentorship and encouragement. I also like to thank my committee members Walter Morris and Zoran Duric. They were instrumental in helping me improve and verify the results in this thesis. Lastly, I like to thank Prasun Das and Sanghamitra Bandyopadhyay of ISI, Kolkata for several interesting discussions that led me to work on one of the problem explored in this thesis.

Table of Contents

List of Figures

Abstract

ON MAXIMAL LAYERS OF RANDOM ORDERS

Indranil Banerjee

George Mason University, 2015

Thesis Director: Dr. Dana Richards

In this thesis we investigate the maximal layers of random partial orders. Main contributions are two-fold. In the first half we investigate the expected size of different maximal layers of a random partial order. In particular when the points are in a plane, we give an enumerative formula for the distribution of the size of these maximal sets. Using Monte-Carlo based simulation we extrapolate the results for higher dimensions. In the second part we explore the computational aspect of the problem. To this end we propose a randomized algorithm for computing the maximal layers and analyze its expected runtime. We show that the expected runtime of our proposed algorithm is bounded by $o(kn^2)$ when k is fixed and in the worst case by $O(kn^2)$. This is the first non-trivial algorithm whose run-time remains polynomial whenever k is bounded by some polynomial in n while remaining sub-quadratic in n for constant k. We also extend these results to random orders with arbitrary distributions.

Chapter 1: Introduction

Random partial orders, soon to be defined formally, are one of the more fundamental objects in combinatorics as well as in computational geometry. Unlike their much studied cousins, random graphs, random partial orders has not been studied that extensively. This is partly due to the fact that the partial ordering among elements makes independence assumptions in probabilistic analysis oftentimes elusive. Before formally introducing random orders, we shall first introduce the concept of *maximal sets* of partial orders.

A set P along with a relation $R \subset P \times P$ such that R is antisymmetric, transitive and reflexive is called a partially ordered set (poset). If for all $x, y \in P$ either (x, y) or $(y, x) \in R$ then it is a total order (ordering between each pair of element is known). We will first introduce maximal sets in the geometric setting and then for partial orders. Let P (with $|P| = n$) be a set of points in some k-dimensional space such that in each dimension points are *orderable*[1]. When $k = 2$, we can view P as a set of points in a plane. Given such a set of points we often want to determine the boundary of the set. In one dimensional space, when P is just points on a line [1] the boundary set can be defined unambiguously. That is, it consists of the minimum and the maximum point of P. If P is a multi-set then there may be more than one maximum or minimum point or both. However, we can define the boundary set in several ways if P resides in a plane or in some higher dimension. When P consists of points on a plane one of the most common way to define the boundary of P as the smallest convex set $Q \subset P$, such that if a convex set contain Q then it contains P. This convex set Q is called the convex hull $\mathcal{CH}(P)$ of P. In the plane, the shape of this set is just a convex polygon. For any P the convex hull $\mathcal{CH}(P)$ is unique modulo points that are identical.

[1] We shall assume the finitary setting here: that is objects are finite as well as the dimensions in which they reside. This makes the algorithmic aspect of our study meaningful.

In this thesis we look at a different kind of boundary set, called the maximal set of P. Before we can introduce it, we need to define some terms first. Let P be a set of k-dimensional points. We shall use $x[i]$ to denote the i^{th} coordinate of x.

Definition 1.1. For two points $x, y \in P$ we say $x \succ y$ (x *dominates* y or x is *above* y) if $x[i] \geq y[i]\ \forall i \in [1...k]$.

Definition 1.2. If neither $x \succ y$ nor $y \succ x$ then we say that x and y are incomparable, denoted by $x \| y$.

Definition 1.3. If $\mathcal{ML}(P)$ is the maximal set of P then:

$$\mathcal{ML}(P) = \{x \in P \mid \nexists y \in P \text{ and } y \succ x\}$$

We refer the maximal set and maximal layer interchangeably. Unlike $\mathcal{CH}(P)$, $\mathcal{ML}(P)$ is not unique. It depends on how we define *dominance*. For example if instead use $\leq$ in place of $\geq$ we still get a valid definition. If defined in this way the set is called the *minimal set* of P. In general, we can define the extremal set of P with respect to a orthant in k-space:

Definition 1.4. If $\mathcal{ML}(P, O)$ is the extremal set of P with respect to a orthant O, then:

$$\mathcal{ML}(P, O) = \{x \in P \mid \nexists y \in P \text{ and } y \succ_O x\}$$

and $x \succ_O y$ if $x[i]$ OP $y[i]$ for each $i \in [1...k]$. Where, OP is either $\geq$ or $\leq$ depending on the orthant O.

Thus there are 2^k different extremal sets corresponding to each orthants. In this study we will not use this generalized definition but work with the one defined earlier. The results presented here carries over to this general setting without much difficulty. Next we define the i^{th} maximal set of P:

Definition 1.5. 1. $\mathcal{ML}_1(P) = \mathcal{ML}(P)$

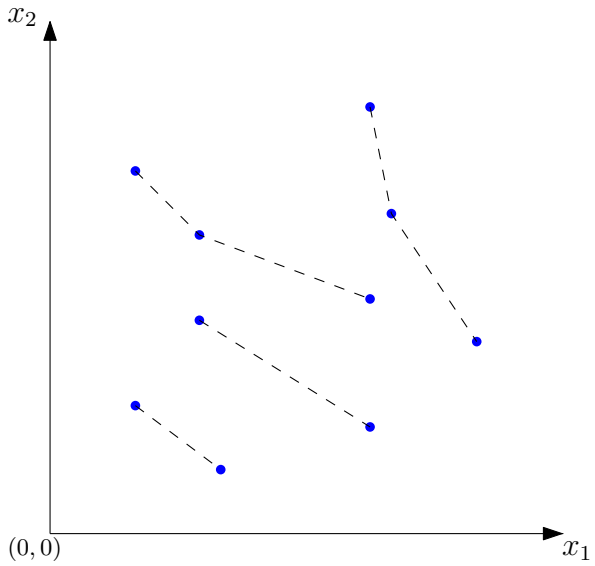

Figure 1.1: Dashed chains represents the maximal layers of the point set.

2. $\mathcal{ML}_i(P) = \mathcal{ML}(P_i)$ for $i > 1$

   Where, $P_i = P \setminus \bigcup_{j=1}^{i-1} \mathcal{ML}_i(P)$.

The i^{th} maximal layers is thus the maximal layer of the remaining set of points after peeling away all of the previous layers from P. If points are in line ($k = 1$) then each layer consist of only one point and the layer ordering is just the ordering of the points in the line. This natural ordering between the layers carries forward to dimensions $k > 1$.

In the combinatorial setting we define maximal layers for the poset $(P, R_{\succ})$. The set of points in P forms the poset $(P, R_{\succ})$ in the obvious way: If $x, y \in P$ and $x \succ y$ then $(x, y) \in R_{\succ}$. Since for the most part the set $R_{\succ}$ will be clear from the context we shall use P to represent the point set as well the poset. The definitions of maximal layers remains the same. One could view the layers of P as levels. The Figure 1.1 highlights the layers of P in the planar setting. Clearly the layers need not be convex. Here are couple of more definitions that we shall use later.

Definition 1.6. We call $C \subset P$ a *chain* if the elements in C and the set $(C \times C) \cap R_{\succ}$ forms a total order.

Definition 1.7. We call $A \subset P$ a *anti-chain* if the elements in A are all mutually incomparable.

Definition 1.8. The length (cardinality) of the largest chain of P is called the *height* of P. It can be defined also as the number of non-empty maximal layers in P. It is denoted as $h_{n,k}(P)$ or simply by h when the context is clear.

Definition 1.9. Similarly the cardinality of the largest anti-chain is called the *width* of P, denoted by $w_{n,k}(P)$ or simply by w when the context is clear.

Definition 1.10. A *chain decomposition* of P is the partition of P into disjoint chains such that each element belongs to exactly one chain.

An *anti-chain decomposition* is defined in an analogous way. Since, points are in the same maximal layer are by definition incomparable and each point belongs to exactly one layer, the maximal layers together constitute an anti-chain decomposition of P.

Definition 1.11. A *linear extension* of P is an ordering (permutation) π of the elements of P such that if $x \succ y$ then $\pi(x) < \pi(y)$. Hence, a linear extension is an ordering which respects the relations between elements of P.

1.1 Random Orders

Now we are ready to formally define random orders. There has been several models of random orders proposed in the literature, for a detail treatise the reader is referred to [1, 2]. Here we shall give the two most common one.

Definition 1.12 (Geometric)**.** Let each $x \in P$ is picked uniformly and independently from $[0,1]^k$, the unit cube equipped with a domination order. Then P is a random order in $[0,1]^k$. We can pick x from $U[0,1]^k$ by picking each component $x[i]$ from $U[0,1]$ independently.

Here, the notation $U[0,1]$ denotes the uniform distribution over the interval $[0,1]$. We shall use the notation $U[S]$ to denote the uniform distribution over the set S. Let π_1 and

π_2 be two permutations of $[1...n]$ not necessarily distinct. We define the intersection of π_1 and π_2 as the poset P such that $P = \{(\pi_1(i), \pi_2(i)) \mid i \in [1...n]\}$. Then '$\succ$' is defined in the obvious way.

Definition 1.13 (Combinatorial)**.** For each $i \in [1...k]$ we pick one permutation π_i uniformly and independently from S_n, the symmetric group. Then the intersection of these permutations forms a random order in k dimensions[2].

The two definitions are equivalent, even though according to the first definition P can be a multi-set. Although the event in which that happens has a null-support. In this thesis we shall use both of these definitions whenever it suits our purpose. In chapter two, where we discuss the distribution of the sizes of $\mathcal{ML}_i(P)$, we will use the combinatorial definition. On the other hand, in chapter 3, we shall use the geometric definition to analyze our randomized algorithm. We shall also need some more definitions to help with our analysis but we will introduce them as they become necessary. From now on we shall refer to maximal layers as just *layers*.

The thesis is organized as follows: In Chapter 2 we discuss the expected size of different layers of P. In section 2.1 we compute this size distribution of different layers when $k = 2$. Next, we study the behavior of this distribution using Monte-Carlo based simulation for higher values of k. In Chapter 3 we begin by introducing some related work on computing maximal sets. Next we describe our proposed algorithm in Section 3.3. In Section 3.4 we analyze its expected runtime which we continue to Section 3.5 in light of a new data-structure. In Section 3.6 we discuss how we can extend our algorithm when P is not a proper random order (as defined above) but some random set of vectors. We summarize and conclude in Chapter 4.

[2]Here by dimension of P we mean the dimension of its elements and not the dimension of the poset P, which is defined in a different way: The dimension of the poset P denoted by $\text{DIM}(P)$, is the minimum number linear extensions of P whose intersecting order gives the partial order P.

Chapter 2: Expected Size of Maximal Layers

Formal study of random partial orders were first investigated in 1980s. This is in contrast to random graphs; the study of which began much earlier [3, 4]. For random orders authors have studied height, width, number of isolated points, number of linear extensions, size of the first maximal layer and other such properties of a random orders[1, 2, 5–7]. Some of the more general aspects of these type of results were encapsulated as first order statements on partial orders [2]. For example we already have a tight upper and lower bound for the height and width of P. Winkler gave the first non-trivial bound for the expected height h as $\Theta(n^{1/k})$ and showed that the expected width w to satisfy: $\frac{1}{e}n^{1-1/k} \leq w(P) \leq (\ln n)n^{1-1/k}$[2]. Later Brightwell tightened the upper bound to $4kn^{1-1/k}$[1].

However, to the best of our knowledge, there has not been any attempt at computing the distribution of points into different layers of a random order. We only have a bound on the size of the first layer, which is $O(\log^{k-1} n)$ [5]. Let $|\mathcal{ML}_m(P)| = w_m(n, k)$ be the expected size of the m^{th} layer ($2 \leq m \leq n$). We want to know how $w_m(n, k)$ behaves as a function of n, k and m. Since n and k are clear from context we shall simply use w_m in place of $w_m(n, k)$. We already have a bound on w_1 as stated earlier. In the ideal scenario we want to get similar bounds for w_m as we have for w and h. Trivially, $w_m \leq w$ for all m. We also note that $w_n = 0$ unless P is a total order. Since, $w = \Theta(n^{1-1/k})$ and $w_1 = O(\log^{k-1} n)$ it can be conjectured that w_m initially grows and then falls as m goes from 1 to n. This is also evident from our empirical results.

In the next section we will describe how we can compute w_m combinatorially (when $k = 2$), before moving on to empirical simulations. In this regard we take an enumerative approach. This is in contrast to a recursive one taken by the authors in [5] to compute $w_1(n, k)$.

2.1 An Enumerative Strategy To Compute $w_m(n, 2)$

The case when $m = 1$ is a special one. We know that an element $x \in P$ belongs to the first layer iff it is not dominated by any element in $P \setminus x$. It does not matter how these elements in $P \setminus x$ are related to each other: that is the sub-poset formed by these elements does not decide the membership of x in $\mathcal{ML}_1$. Only their individual relations with x does. However, this notion cannot be extended to compute the expected size of other layers. From Definition 1.5, $x \in \mathcal{ML}_m$ iff there exists a chain of size $m-1$ above x. Hence, the relationships between elements that dominate x determines which layer x will belong to. For example, the recursive formulation given in [5] fails for $m > 1$ precisely because of this reason.

In this section, using enumeration, we will derive an expression for $w_m(n, 2)$. However, the computation depends on whether we can compute the number of permutations having some increasing subsequence of a given length or greater. Let us define $T(n, l)$ to be the number of permutations (of $[1, ..., n]$) whose largest increasing subsequence has length at least l. The following theorem gives an expression for $w_m(n, 2)$ in terms of $T(n, l)$.

Theorem 2.1. The expected size of $w_m(n, 2)$ is given by:

$$w_m(n, 2) = W_m(n) - W_{m+1}(n) \tag{2.1}$$

where, $W_m(n) =$

$$\left(\frac{1}{n!}\right) \sum_{i=1}^{n} (n-i)! \sum_{l=m-1}^{i-1} \binom{i-1}{l} T(l, m-1) \sum_{j=m}^{n} \binom{j-1}{l} (n-j)_{i-1-l}$$

and $(n)_k = \prod_{i=0}^{k-1} (n-i)$ is the falling factorial.

Proof. We first sort the vectors in P dimension-wise in *descending* order. Relabel the vectors according to their ranks in the sorted order in the first dimension. Now with respect to this labelling, the (sorted) ordering of elements in the other dimension is just some permutation

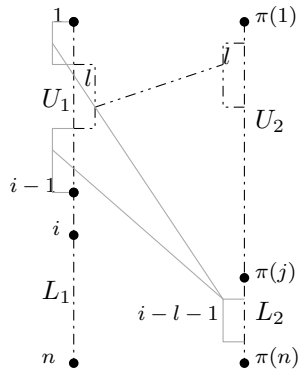

Figure 2.1: Visualization of the four sets U_1, L_1, U_2, L_2

of $[1, ..., n]$. Here, the first dimension is just the identity permutation (see Figure 2.1).

Let us consider the i^{th} vector in this labeling, that is, the vector which has a rank i in dimension 1. Suppose it has the rank j in the other dimension. Our strategy is to compute the probability that the vector with rank i in the first dimension will belong to the m^{th} layer or below. Let this probability be $P(i, n)$. Then,

$$W_m(n) = \sum_{i=1}^{n} P(i, n)$$

will be the expected number of points which are on or below the m^{th} layer. Hence, $w_m(n, 2) = W_m(n) - W_{m+1}(n)$.

We compute $P(i, n)$ by counting the number of instances (of the random order P) in which this event occurs. In the first dimension the element has a rank i and in the second it has a rank j. Given a fixed permutation π, for the element to belong to some layer $\geq m$, there must exists an increasing subsequence of length at-least $m - 1$ in U_2 consisting of elements only from U_1. Here, $U_1 = (1, ..., i - 1)$ and $U_2 = (\pi(1), ..., \pi(j - 1))$. This in turn can be computed by considering each possible subset of U_1 of size $m - 1$ to $i - 1$ and asking how many ways we can map it to U_2. For each l-subset of U_1, we can place it in U_2 in $\binom{j-1}{l}$ ways and whose elements can be permuted among themselves in exactly $T(l, m - 1)$

ways. Now the rest of U_1 (those elements which were not mapped to U_2) is mapped to L_2 in $(n-j)_{i-1-l}$ ways. Lastly, the elements of L_1 can be placed into $U_2 \cup L_2$ in $(n-i)!$ way. We do this for each subset of U_1 of size $\geq m-1$ and each rank j from m to n. Thus yielding,

$$P(i,n) = \left(\frac{1}{n!}\right) \sum_{l=m-1}^{i-1} \binom{i-1}{l} T(l, m-1) \sum_{j=m}^{n} \binom{j-1}{l} (n-j)_{i-1-l}(n-i)! \tag{2.2}$$

This immediately gives the expression for $w_m(n,2)$ as stated in the theorem.

□

In order to compute $w_m(n,2)$ from Theorem 4 we need to compute $T(n,k)$. Authors have found that expressing $T(n,k)$ in terms of some generating function is quite difficult. Baik and others [8] proved that l_n converges in distribution to the Tracy-Widom distribution [9], where, l_n is the size of the largest increasing sequence of a random permutation. Thus it is possible to numerically approximate $T(n,k)$ (and consequently $w_m(n,2)$) by first numerically approximating the cumulative probability $P(l_n \geq k)$ of the Tracy-Widom distribution.

2.2 Experimental Approximation of $w_m(n,k)$

Using Monte-Carlo based simulation we study the behavior of $w_m(n,k)$ and $w^*(n,k) = \max_m w_m(n,k)$. Results of these simulations, averaged over 100 independent runs, are shown in the figures below. We generate random orders using the combinatorial model. We first compute a set of k random permutations of $[1,...,n]$ and determine their product order as described in Chapter 1. Not surprisingly, we see that the expected number of non-empty layers decreases quite sharply, supporting the fact that the expected height is bounded by $O(n^{\frac{1}{k}})$. It is also interesting to note the behavior of $w^*(n,k)$ as k increases. It is evident from Figure 2.3 that $w^*(n,k)$ tends to a linear function of n (for a fixed k). This

is consistent with the fact that $\lim_{k\to\infty} w^*(n,k) = w$, as the number of layers in P decreases and we already know that the expected width w is bounded by $O(n^{1-1/k})$ for fixed k.

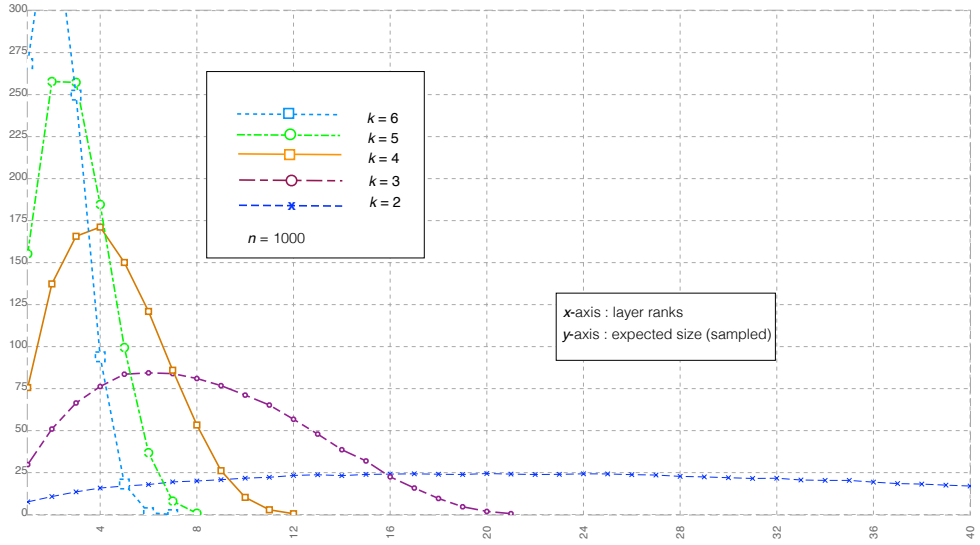

Figure 2.2: Plot showing the behavior of $w_m(n,k)$ for different values of k. Here n is fixed to 1000.

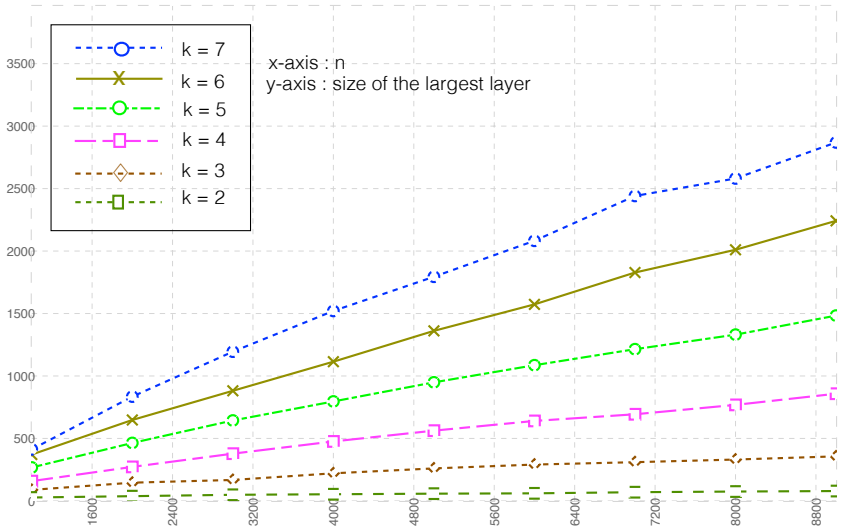

Figure 2.3: Here we see that $w^*(n,k)$ tends to a linear function of n as k increases. Here n is varied from 1000 to 9000.

Chapter 3: Computing Maximal Layers

Computing the set $\mathcal{ML}_1(P)$ is an well studied problem in computational geometry. In this chapter we look at the problem of computing all of the non-empty maximal layers of P, when P is a random order. The chapter is organized as follows. In the next section we review some related work. Section 3.2 describes in detail the proposed algorithm and its run-time analysis is done in Section 3.3. In Section 3.4 we introduce a new data-structure which is then used to further shape the runtime of the proposed algorithm. We conclude with Section 3.5 in which we extend our result for orders that are not-necessarily uniformly random as defined in Chapter 1.

3.1 Related Work

Henceforth we shall denote the the problem of identifying the layers of P having n elements in k dimensions as MaxLayers(n, k). There exists a tight bound for MaxLayers(n, k) when $k \leq 3$, which is $\Theta(n \log n)$ [10–12]. However, we do not have any better lower bound when $k > 3$. For fixed $k > 3$ best known upper-bound is $O(n \log^{k-1} n)$ [13]. Interestingly, the upper bound to find only the first maximal set is $O(n \log^{\max(1,k-2)} n)$. The previous two results hold in worst-case. We see that, for fixed k, these algorithms can be regarded as almost optimal, as they only have a poly-logarithmic overhead over the theoretical lower bound of $\Omega(n \log n)$. Conceptually, they implement multi-dimensional divide and conquer algorithms [14] on input P which introduces the poly-log factor in their runtimes. The set P is partitioned into sub-sets that are both smaller in the number of dimensions as well as the number of points. We shall briefly go over the original algorithm proposed in [15] which computes the first maximal layer of P.

Algorithm 1: Maximal Finding Algorithm as Proposed in [15]

Input : Point set $P(x_1, ..., x_n)$
Output: $\mathcal{ML}_1(P)$

```
1 Sort (x_1[1], ..., x_n[1]) ;        // sort the points in their first coordinates
2 MaximalLayer (P)
3 begin
4     Partition P into two halves according to the sorted ordering (y_1, ..., y_n) in
      dimension 1;
5     Let, L ← (y_1, ..., y_{n/2}) and R ← (y_{n/2+1}, ..., y_n);
6     Recursively find the MaximalLayer (L) and MaximalLayer (R);
7     Find the set S ⊂ MaximalLayer(R) such that no elements in MaximalLayer(L)
      dominates any elements in S;
8     return MaximalLayer(L) ∪ S
```

From the pseudo-code above we can see that both the subsets L and R can solved independent of each other before combining the results. Also, when merging `MaximalLayer`(L) and S we have to loot at only $k-1$ of the dimensions, as the elements are already sorted in one of the dimensions. Analyzing this algorithm leads to a run-time of $O(n \log^{k-2} n)$ in the worst case ($k \geq 4$). Subsequently Jensen [13] extended this algorithm to identify all the layers in $O(n \log^{k-1} n)$ deterministic time.

Things get interesting if k is part of the input. An extreme case when $k = \theta(n)$, these poly-logarithmic upper-bounds above becomes quasi-polynomial (in n). However, there is a trivial algorithm (which compares each point against the other, and keeps track of the computed transitive relations) that requires $O(kn^2)$ time in the worst case. Although, for finding only the first maximal layer, [16] Matousek proposed a deterministic algorithm that runs in $O(n^{(3+\omega)/2})$ when $k = n$, where, $O(n^\omega)$ is the complexity of multiplying two $n \times n$ matrices. Since $\omega > 2$ we see that the algorithm runs in $\omega(n^2)$ time. In a recent paper [17],

Impagliazzo and others show that for determining whether there exists a pair (u, v), where $u \in A$ and $v \in B$ (A and B are both sets of vector of size $O(n)$) such that $u \succ v$, can be done in sub-quadratic time provided $k = O(\log n)$.

3.2 The Iterative Algorithm

3.2.1 Defination

We will use the notation $S \succ p$, where S is a set of incomparable elements, to denote that $\exists q \in S$ such that $q \succ p$. If $S \succ p$ we say that S is *above* p. Furthermore, if $p \succeq q$ then either $p = q$ or $p \succ q$. Let $\mathcal{O} : [0,1]^k \times [0,1]^k \to \{0,1\}^k$, such that $\mathcal{O}(p,q)[j] = 1$ if $p[j] < q[j]$ and 0 otherwise. This definition, which might seem inverted, will make sense when we discuss it in the context of our data structures. We call $\mathcal{O}$ the orthant function as it computes the orthant with origin p in which q resides.

In our analysis we shall use the typical RAM model, where the operation of evaluating $p[j] \geq q[j]$ takes constant time. We take P to be a random order as defined Chapter 1. That is, we build P by picking points uniformly at random from $[0,1]^k$. We shall use MaxPartition(P) as the main procedure for solving an instance of MaxLayers(n, k).

3.2.2 Data Structures

In this section we introduce the framework on which MaxPartition(P) is based. Let B be a self-balancing binary search tree (for example B could be realized as a red-black tree). Let $B(i)$ be the i^{th} node in the in-order traversal of B. Each node of B stores three pointers. One for each of its children (null in place of an empty child) and another pointer which points to an auxiliary data structure. If X is a node in B then left and right children of X are denoted as $l(X)$ and $r(X)$ respectively. By $L(X)$ we denote the auxiliary data structure associate with X. When the context is clear, we shall simply use L in place of $L(X)$.

We also use L as a placeholder for any data structure that can be used to store the set of points of a single layer of P. For example, L could be realized as a linked list. Additionally,

L must support INSERT(L, p) and ABOVE(L, p). The ABOVE(L, p) operation takes a query point p, and answers the query $L \stackrel{?}{\succ} p$. The INSERT(L, p) operation inserts p into L, which assumes p is incomparable to the elements in L. So, we must ensure that L is the correct layer for p before calling INSERT(L, p).

We observe that the layers of P are themselves linearly ordered by their ranks from 1 to h. Thus we can use B to store the layers in sorted order, where each node $B(i)$ would store the corresponding layer $\mathcal{M}_i$ (using $L(B(i))$). We endow B with INSERT(B, p) and SEARCH(B, p) (we do not need deletion) operations. The INSERT(B, p) procedure first calls the SEARCH(B, p) procedure to identify which node $B(i)$ of B the new point p should belong and then calls INSERT$(L(B(i)), p)$. If p does not belong to any layer currently in B then we create a new node in B. The SEARCH(B, p) procedure works as follows: we can think of B as a normal binary search tree, where the usual comparison operator $\geq$ has been replaced by the ABOVE(L, p) procedure. Furthermore, the procedure can only identify whether $L \succ p$ or $L \not\succ p$. This is exactly equivalent to the situation where we have replaced the comparison operator $\geq$ with $>$. So we must determine two successive nodes $B(i)$ and $B(i+1)$ such that $L(B(i)) \succ p$ and $L(B(i+1)) \not\succ p$. If such a pair of nodes does not exist then we return a null node.

3.2.3 MaxPartition(P)

We begin by first computing a linear extension T of P. We initialize B as an empty tree. We iteratively pick points from P in increasing order of their ranks in T and call INSERT(B, p), where p is the current point to be processed. INSERT(B, p) subsequently calls SEARCH(B, p). We have two possibilities:

CASE 1: SEARCH(B, p) returns a non-empty node $B(i)$. We then call INSERT$(L(B(i), p)$.

CASE 2: SEARCH(B, p) returns a null node. Then we create the node $B(m+1)$ in B, where m is the number of nodes currently in B. We first initialize $B(m+1)$ and then call INSERT$(L(B(m+1), p)$ on it. We note that, when we create a new node in B it must always

be the right-most node in the in-order traversal of B. This follows from the order in which we process the points. Since p succeeds a processed point q in the linear extension T, hence $p \not\succ q$. Thus, if p does not belong to any of nodes currently in B then it must be the case that p is below all layers in B. MaxPartition(P) terminates after all points have been processed. At termination $L(B(i))$ stores all of the points in $\mathcal{M}_i$ for $1 \leq i \leq h$. We make a couple of observations here. 1) When a point is inserted into a node $B(i)$ it will never be displaced from it by any point arriving after it. 2) Since, nodes are always added as the right-most node in B, for Search(B, p) to be efficient, B must support self-balancing.

If we assume that Above(L, p) and Insert(L, p) to work correctly, at once we see that Search(B, p) and Insert(B, p) are also correct. Hence, each point is correctly assigned to the layer it belongs to.

Algorithm 2: The MaxPartition Procedure

Input : Point set $P(x_1, ..., x_n)$
Output: Labels each point with the rank of the layer is belongs to.

```
1  begin
2      Y ← LinearExtension (P) ;          // computes a linear extension of P
3      Initialize (B) ;                                  // A self-balancing BST
4      for i ← 1 to n do
5          Insert (B, y_i) ;
```

```
1  Insert (B, y) begin
2      p ← null;
3      Search (B, y, p);
4      Insert (p, y);
```

```
1  Search (B, y, p) begin
2      if L(B) = ∅ then
3          if p = null then
4              Create a new node B* ;      // this will be the right-most node in
               the BST
5              p ← B*(L);
6      if Above (L(B), y) then
7          Search (B.right, y, p)
8      else
9          p ← L(B);
10         Search (B.left, y, p)
```

3.3 Runtime Analysis

Let $\textsc{Above}(L, p)$ take $t_a(|L|)$ time. As mentioned in section 3.2, $|L| \leq w$ for any layer in B. Hence, $t_a(w)$ is an upper bound on the runtime of $\textsc{Above}(L, p)$. Similarly, we bound the runtime of $\textsc{Insert}(L, p)$ with $t_i(w)$. Let p be the next point to be processed. At the time B will have at most h nodes. In order to process p the $\textsc{Insert}(B, p)$ will be invoked, which in turn calls the $\textsc{Search}(B, p)$ as discussed above. But the $\textsc{Search}(B, p)$ will employ a normal binary search on B with the exception that at each node of B it invokes the $\textsc{Above}(L, p)$ instead of doing a standard comparison. Since, B is self-balancing the height of B is bounded by $O(\log h)$. Hence, the number of calls to $\textsc{Above}(L, p)$ is also bounded by $O(\log h)$, each of which takes $t_a(w)$ time. Also, for each point p, $\textsc{Insert}(L, p)$ is called only once. We also assume initializing a node in B takes constant time. So, processing of p takes $O(t_a(w) \log h + t_i(w))$ and this holds for any point.

Lemma 1. We can compute a linear extension T of P in $O(n \log n + kn)$ time in the worst case.

Proof. We shall compute T as follows: Let $\mu(p) = \max_{1 \leq j \leq k} \; p[j]$. Then sorting the points in decreasing order of $\mu(p)$ will give us T. It is trivial to see that T is a linear extension of P. This takes $O(n \log n + kn)$ in the worst case.

□

The reason for computing T in this way will be clear when we get to the analysis of our algorithm. Later we shall see that the time bounds for $\textsc{Above}(L, p)$ and $\textsc{Insert}(L, p)$ will dominate the time it takes to compute T. So we will ignore this term in our run-time analysis. The next lemma trivially follows from the discussion above.

Lemma 2. The procedure $\textsc{MaxPartition}(P)$ takes $O(n(t_a(w) \log h + t_i(w)))$ time and upon termination outputs a data structure consisting of the maximal layers of P in sorted order.

3.4 Realization of L using Half-Space Trees

In this section we introduce a new data structure for implementing L. We shall refer to it as Half-Space Tree (HST).

The function $\mathcal{O}(p, q)$ computes which orthant q belongs to with respect to p as the origin. Clearly, there are 2^k such orthants, each having a unique label in $\{0,1\}^k$. Let $H_j(p)$ be a half space defined as: $H_j(p) = \{q \in [0,1]^k \mid \mathcal{O}(p,q) = \{0,1\}^{j-1}0\{0,1\}^{k-j}\}$ passing through origin p whose normal is parallel to dimension j. Here $\{0,1\}^{j-1}0\{0,1\}^{k-j}$ represents a 0-1 vector for which the j^{th} component is 0. We shall use the notation $h_j(p)$ to denote the extremum orthant of $H_j(p)$ (w.r.t $\succ$), that is, $h_j(p) = 1^{j-1}01^{k-j}$. There are k such half spaces. An orthant whose label contains m 1's lies in the intersection of some $k-m$ such half spaces.

Lemma 3. If $p, q \in P$ and $p \parallel q$ then $\mathcal{O}(p,q) \in \{0,1\}^k \setminus \{0^k, 1^k\}$. That is, q can only belong to orthants which lie in the intersection of at most $k-1$ half spaces.

Proof. Trivially follows from definitions. □

Corollary 1. The above lemma holds if p and q belongs to the same layer. However, the converse of this statement is not true.

3.4.1 Half-Space Tree

We define a k-dimensional HST recursively as follows:

Definition 3.1. (HST).

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 HST.
3. If node q is the j^{th} child of node p then $h_j(p) \succeq \mathcal{O}(p,q)$.

An HST stores points from a single layer. So Corollary 1 tells us that for any node p and a new point q at most $k-1$ of the children nodes satisfy $h_j(p) \succeq \mathcal{O}(p,q)$. Hence, q can be inserted into any one out of these children nodes. Henceforth, we will also use w (the width of $(P,\succ)$) to bound the number of points currently stored inside L. By $L.j$ we denote the j^{th} child node of L.

Above(L,p)

Let us assume that L is realized by an HST. The ABOVE(L,p) works as follows: First we compute $\mathcal{O}(r,p)$. Here, r is the root node. If $\mathcal{O}(r,p) = 0^k$ then we return $L \succ p$. Otherwise we call ABOVE$(L.j,p)$ recursively on each non-empty child node j of root r, such that $h_j(r) \succeq \mathcal{O}(r,p)$. When all calls reach some leaf node, we stop and return $L \not\succ p$.

proof of correctness. CASE 1:$(L \succ p)$ Let q be some point in L such that $q \succ p$, prior to calling ABOVE(L,p). Before reaching the node q, if we find some other node $q' \succ p$ then we are done. So we assume this is not the case. We claim that p will be compared with q. We show this as follows: Let the length of path from root r to q be $i+1$. Let $u_0,...,u_i$ be the sequence of nodes in this path (here $u_0 = r$ and $u_i = q$). Since, $q \succ p$, $\mathcal{O}(u_m,q) \succeq \mathcal{O}(u_m,p)$ for all $0 \leq m < i$. But, u_m is a predecessor node in the path from r to q, hence $h_{j_m}(u_m) \succeq \mathcal{O}(u_m,q)$ where u_{m+1} is the j_m^{th} child of u_m. Which implies $h_{j_m}(u_m) \succeq \mathcal{O}(u_m,p)$ (from transitivity of $\succeq$) for $0 \leq m < i$. Thus we will traverse this path at some point during our search.

CASE 2:$(L \not\succ p)$ Follows trivially from the description of ABOVE(L,p). □

Insert(L,p)

INSERT(L,p) is called with the assumption that $L \not\succ p$. If the root is empty then we make p as the root and stop. Otherwise, we pick one element uniformly at random from the set $S_r = \{j \in \{1,...,k\} \mid h_j(r) \succeq \mathcal{O}(r,p)\}$ and recursively call INSERT$(L.j,p)$.

proof of correctness. It is easy to verify that insert procedure maintains the properties of HST given in definition 2. □

Although the insert procedure is itself quite simple, it is important that we understand the random choices it makes before moving further. These observation will be crucial to our analysis later. Let the current height of L be h_L. By L^* we denote the complete HST of height h_L, clearly L^* has $\Theta(k^{h_l})$ nodes. We color edges of L^* red if both of the nodes it is incident to are present in L, otherwise we color it blue. Unlike ABOVE, we can imagine that the INSERT procedure works with L^* instead of L. Upon reaching a node r in L^* the procedure samples uniformly at random from the set S_r as above. This set may contain edges of either color. If a blue edge have been sampled then we stop and insert p into the empty node incident to the blue edge in L. So we see that, despite not being in L, the nodes incident to blue edges affect the sampling probability equally.

Algorithm 3: HSTSEARCH(H, x)

```
1 begin
2     r ← root (H);
3     if r = null or r ≻ x then
4         return
5     for i ← 1 to k do
6         if x[i] ≤ r[i] then
7             HSTSearch (H.i, x)
```

Algorithm 4: HSTInsert(H, x)

```
1  begin
2      if H = null then
3          Initialize (H, x)// A new layer is initialized with x at the root
           return
4      r ← root(H);
5      S_r ← ∅;
6      for i ← 1 to k do
7          if x[i] ≤ r[i] then
8              S_r ← S_r ∪ i
9      j ← U[S_r] ;        // pick a half-space(child-node) uniformly at random
10     HSTInsert (H.j, x);
```

3.4.2 Runtime Analysis

Here we compute $t_a(w)$ and $t_i(w)$ in expectation over the random order P and the internal randomness of the Insert(L, p) procedure. From the discussion in section 4.1 we clearly see that $t_i(w) = O(t_a(w))$. So it suffices to upper bound $t_a(w)$ in expectation. Furthermore, we only need to consider the case when Above(L, p) returns $L \not\succ p$ as the other case would take fewer number of comparisons. Let this time be $u(w)$. We divide our derivations to compute $u(w)$ into two main steps:

i. Compute the expected number of nodes at depth d of L having w number of nodes.

ii. Use that to put an upper bound on the number of nodes visited during a call to Above(L, p) (when $L \not\succ p$).

We choose to process points according to T as detailed earlier. We denote this ordering by the ordered sequence $(p_1, ..., p_n)$.

Lemma 4. For any two points p, q where p precedes q in T we have the probability that $p[j] > q[j]$ is $\eta_1(k) = 1 - \frac{1}{2}\frac{k-1}{k+1}$. Additionally, if p and q are incomparable then it is $\eta_2(k) = 1 - \frac{1}{k} - \frac{1}{2}\frac{k-2}{k+2}$.

Proof. Recall that T is a linear extension of P. Since p precedes q in T, $\mu(p) > \mu(q)$. Hence, $\exists\ j^{'} \in \{1, ..., k\}$ such that $p[j'] > q[j']$. Let $j' = \text{argmax}_{1 \leq j \leq k}\, p[j]$. We compute the probability $\Pr[p[j] > q[j] \mid \mu(p) > \mu(q)]$ in two parts over the disjoint sets $\{j = j^{'}\}$ and $\{j \neq j^{'}\}$:

$$\begin{aligned}\Pr[p[j] > q[j] \mid \mu(p) > \mu(q)] &= \Pr[p[j] > q[j] \mid j = j^{'}, \mu(p) > \mu(q)] \Pr[j = j^{'}] \\ &\quad + \Pr[p[j] > q[j] \mid j \neq j^{'}, \mu(p) > \mu(q)] \Pr[j \neq j^{'}] \\ &= 1\frac{1}{k} + \left(1 - \frac{1}{2}\frac{\mu(q)}{\mu(p)}\right)\left(1 - \frac{1}{k}\right) \qquad (3.1)\end{aligned}$$

Since,

$$\Pr[p[j] > q[j] \mid j \neq j^{'}, \mu(p) > \mu(q)] = \frac{(\mu(p) - \mu(q))\mu(q) + \frac{\mu(q)^2}{2}}{\mu(p)\mu(q)} = 1 - \frac{1}{2}\frac{\mu(q)}{\mu(p)}$$

This follows from the fact that $p[j]$ and $q[j]$ are independent random variables uniformly distributed over $[0, \mu(p)]$ and $[0, \mu(q)]$ (given $\mu(p) > \mu(q)$) respectively. In the set $\{j = j^{'}\}$ clearly $p[j] > q[j]$. However, in the set $\{j \neq j^{'}\}$ the probability that $p[j] > q[j]$ is $\left(1 - \frac{1}{2}\frac{\mu(q)}{\mu(p)}\right)$. We note that $\mu(p), \mu(q)$ are themselves random variables. More importantly they are i.i.d random variables having the following distribution:

$$\Pr[\mu(p) < t] = t^k$$

on the interval $[0, 1]$. This follows from how points in P are constructed. We take the

expectation of both side of Equation 3.1 over the event space generated by $\mu(p), \mu(q)$ on the set $\{\mu(p) > \mu(q)\}$:

$$\Pr[p[j] > q[j] \mid \mu(p) > \mu(q)] = 1 - \frac{1}{2}\mathbb{E}\left[\frac{\mu(q)}{\mu(p)} \;\mid\; \mu(p) > \mu(q)\right]\left(1 - \frac{1}{k}\right)$$

$$= 1 - \frac{1}{2}\left(\frac{k}{k+1}\right)\left(1 - \frac{1}{k}\right) = 1 - \frac{1}{2}\left(\frac{k-1}{k+1}\right)$$

A similar argument can be used to prove the second claim. □

Theorem 3.1. After w number of insertions the expected number of nodes at depth d in L is given by:

$$k^d\left(1 - \sum_{i=1}^{d} \frac{(1 - \frac{1}{k^i})^{w-1}}{\prod_{j=1, j\neq i}^{d}(1 - \frac{1}{k^{i-j}})}\right)$$

Proof. Let $X_{w,d}$ be the number of nodes at depth d of L after w insertions. Due to the second assertion of Lemma 3 we know that any new point to be inserted can belong to any of the k half-spaces with probability $\eta_2(k)$, which is constant over the half-spaces. The insert procedure selects one of these candidate half-spaces uniformly at random. Thus it follows from symmetry that a particular half-space will be chosen for insertion with probability $\frac{1}{k}$. If the subtree is non-empty then we do these recursively. We define an indicator random variable for the event that the t^{th} insertion adds a node at depth d as $I_{t,d}$. Then,

$$X_{w,d} = \sum_{t=1}^{w} I_{t,d}$$

Taking expectation on both side we get,

$$\mathbb{E}[X_{w,d}] = \sum_{t=1}^{w} \Pr[I_{t,d}]$$

Trivially, $\mathbb{E}[X_{w,0}] = 1$ for $t > 0$. When $d = 1$ and $t \geq 2$ then $\Pr[I_{t,1}] = 1 - \frac{X_{t-1,1}}{k}$. This is because there are $X_{t-1,1}$ nodes at depth 1 (nodes directly connected to the root) hence there are $k - X_{t-1,1}$ empty slots for the node to get inserted at depth 1, otherwise it will be recursively inserted to some deeper node. Hence we have,

$$\mathbb{E}[X_{w,1}] = \sum_{t=2}^{w} \left(1 - \frac{X_{t-1,1}}{k}\right)$$

For $d = 2$, we can similarly argue that the probability of insertion at depth 2 for some $t \geq 3$ is equal to probability of reaching a node at depth 1 times the probability of being inserted at depth 2. It is not difficult to see that this equals: $\left(\frac{X_{t-1,1}}{k}\right)\left(1 - \frac{X_{t-1,2}}{kX_{t-1,1}}\right)$. Hence,

$$\mathbb{E}[X_{w,2}] = \sum_{t=3}^{w} \left(\frac{X_{t-1,1}}{k}\right)\left(1 - \frac{X_{t-1,2}}{kX_{t-1,1}}\right)$$

Proceeding in this way we see that,

$$\mathbb{E}[X_{w,d}] = \sum_{t=1}^{w} \left(\frac{\mathbb{E}[X_{t-1,d-1}]}{k^{d-1}} - \frac{\mathbb{E}[X_{t-1,d}]}{k^{d}}\right)$$

Here we again take expectation on both sides and simplify the expression so that the sum starts from $t = 1$ since the terms $\mathbb{E}[X_{t,d}] = 0$ when $t \leq d$.

Let $a(w, d) = \mathbb{E}[X_{w,d}]$, we can then simplify the above equation to get the following recurrence,

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

with $a(w, d) = 0$ for $w \leq d$. The solution to this can be found by choosing a ordinary generating function $G_d(z)$ with parameter d, such that $G_d(z) = \sum_{t=0}^{\infty} a(t, d) z^t$. To simplify

our calculations we modify the recurrence slightly: With $a(w,d) = k^d b(w,d)$, the recurrence equation becomes,

$$b(w,d) = \frac{1}{k^d} b(w-1,d-1) + (1 - \frac{1}{k^d}) b(w-1,d)$$

Let, $G_d(z) = \sum_{w=0}^{\infty} b(w,d) z^w$. We note that $b(w,d) = 0$ when $w \leq d$. Then we have,

$$\begin{aligned} G_d(z) &= \frac{z}{k^d} G_{d-1}(z) + z(1 - \frac{1}{k^d}) G_d(z) \\ &= \frac{z}{k^d(1 - (1 - \frac{1}{k^d})z)} G_{d-1}(z) \\ &\ldots \\ &= \frac{z^d}{\prod_{i=1}^{d} k^i (1 - (1 - \frac{1}{k^i})z)} G_0(z) \end{aligned}$$

But, $G_0(z) = \sum_{w=0}^{\infty} b(w,0) z^w = \sum_{w=1}^{\infty} z^w = \frac{z}{1-z}$ as $b(w,0) = a(w,0) = 1$ when $w \geq 1$. Hence,

$$\begin{aligned} b(w,d) &= k^{-d(d+1)/2} [z^{w-d-1}] G_d(z) \\ &= k^{-d(d+1)/2} [z^{w-d-1}] \frac{1}{(1-z)\prod_{i=1}^{d} (1 - (1 - k^{-i})z)} \end{aligned} \tag{3.2}$$

Where the notation $[z^i]p(z)$ means the coefficient of z^i in the polynomial $p(z)$ as usual. Using partial fractions: Let,

$$\frac{1}{(1-z)\prod_{i=1}^{d} (1 - (1 - k^{-i})z)} \equiv \frac{\beta_0}{1-z} + \sum_{i=1}^{d} \frac{\beta_i}{(1 - (1 - k^{-i})z)}$$

For which we get the following solution,

$$\beta_0 = k^{d(d+1)/2}$$

$$\beta_i = \frac{k^{d(d+1)/2}(1-k^{-i})^d}{\prod_{j\neq i, j\geq 1}^{d}(1-k^{j-i})}$$

Substituting these in Equation 4 above we get, $b(w,d) = 1 - \sum_{i=1}^{d} \frac{(1-k^{-i})^{w-1}}{\prod_{j=1,j\neq i}^{d}(1-k^{j-i})}$, which gives us the desired result for $a(w,d)$.

□

Before moving on to the main theorem we need another lemma:

Lemma 5. If $B = (b_0, b_1, ..., b_n)$ is a sequence such that $b_r \geq b_{r+1} \geq ... \geq b_n$, then the sum $S = \sum_{i=0}^{n} b_i m^i \leq \sum_{i=0}^{r} b_i m^i + \frac{b_{r+1}m^{r+1}}{(1-m)}$ where $m < 1$.

Proof. We have, $S = \sum_{i=0}^{n} b_i m^i$, where $b_r \geq b_{r+1} \geq ... \geq b_n$. But then,

$$S = \sum_{i=0}^{r} b_i m^i + \sum_{i=r+1}^{n} b_i m^i \leq \sum_{i=0}^{r} b_i m^i + \sum_{i=r+1}^{n} b_{r+1} m^i$$

$$= \sum_{i=0}^{r} b_i m^i + \frac{b_{r+1}m^{r+1}}{1-m}$$

□

Corollary 2. If $m = 1 - \frac{1}{2}\frac{k-1}{k+1}$ and $k \geq 4$ then , $S \leq \sum_{i=0}^{r} b_i m^i + \frac{7}{3} b_{r+1} m^r$.

Theorem 3.2. Expected number of nodes visited during an unsuccessful search $u(w)$ is bounded by $O\left(w^{1-\frac{1}{\log k} + \log_k\left(1+\frac{2}{k+1}\right)}\right)$.

Proof. Before proving this we make the following observation. If for any $d = d_0$, the sequence $a(w,d)$ becomes decreasing, that is, $a(w,d_0) \geq a(w,d_0-1)$ and $a(w,d_0) > a(w,d_0+1)$,

then afterwards it will stay decreasing. This is clear from the fact that $a(w,d)$ represents the expected number of nodes at depth d after w insertions. So the sequence $a(w,d)$ is unimodal since $a(w,0) \leq a(w,1)$ trivially for $w \geq 2$. Let d_0 be the value that maximizes $a(w,d)$.

Let us compute the probability of visiting a node at depth d during a call to ABOVE when the query point is not below L. Let q be the current node being checked and p be the query point. According to Lemma 3 the probability $\Pr[p \in H_j(q)]$ is same for any j and is not dependent on the rank of q in T. Hence it is also not dependent on the depth of q in L. Furthermore, this probability is $\eta_1 = 1 - \frac{1}{2}\frac{k-1}{k+1}$, again from Lemma 3.

Thus the probability of visiting a node at depth d is the result of d independent moves each having probability η_1, hence it is η_1^d. Now we can find the expression for the expected number of nodes visited:

$$\begin{aligned} u(w) &= \sum_{d=0}^{w-1} \eta_1^d a(w,d) \\ &\leq \sum_{d=0}^{d_0} \eta_1^d a(w,d) + \frac{7}{3}\eta_1^{d_0} a(w,d_0+1) \end{aligned} \tag{3.2}$$

Here we use Theorem 2, Lemma 4 and its corollary and the fact that the sequence $a(w,d)$ is unimodal; to bound $u(w)$. Also note that $a(w,d) \leq k^d$. Now we need to upper bound d_0.

We shall denote d_0 as $d_0(w,k)$ as it is a function of both w and k. Hence, $d_0(w,k)$ maximizes $a(w,d)$ as d varies from 0 to $w-1$. Since we are interested in an upper bound on $d_0(w,k)$, we may think of d being fixed and we vary w from 0 to ∞[1]. If we then lower bound w, when $a(w,d)$ maximum, we will be able get a corresponding upper bound for $d_0(w,k)$. This makes our analysis simpler as the number of terms in the expression for $a(w,d)$ is fixed

[1] actually from $d+1$ as terms below it are 0, but this does not affect our analysis

for a fixed d.

Let

$$a'(w,d) = a(w,d-1) - a(w,d)$$

$$= k^{d-1}\left(1 - \sum_{i=1}^{d-1} \frac{(1-\frac{1}{k^i})^{w-1}}{\prod_{j=1,j\neq i}^{d-1}(1-\frac{1}{k^{i-j}})}\right) - k^d\left(1 - \sum_{i=1}^{d} \frac{(1-\frac{1}{k^i})^{w-1}}{\prod_{j=1,j\neq i}^{d}(1-\frac{1}{k^{i-j}})}\right)$$

Letting $\alpha_i = 1 - \frac{1}{k^i}$ we get,

$$a'(w,d) = -k^{d-1}(k-1) + k^{d-1}\sum_{i=1}^{d} \frac{{\alpha_i}^{w-1}(k-\alpha_{i-d})}{\prod_{j=1,j\neq i}^{d}\alpha_{i-j}}$$

Since we wish to compute $d_0(w,k)$ or at least get an upper bound, we assume that $a'(w,d) < 0$. Hence,

$$\sum_{i=1}^{d} \frac{{\alpha_i}^{w-1}(k-\alpha_{i-d})}{\prod_{j=1,j\neq i}^{d}\alpha_{i-j}} < k-1$$

Since, $\prod_{j=1,j\neq i}^{d}\alpha_{i-j} = \prod_{j=1}^{i-1}\alpha_{i-j}\prod_{j=1}^{d-i}(1-k^j) = P(i-1)\prod_{j=1}^{d-i}(1-k^j)$, where, $P(i) = \prod_{j=1}^{i-1}\alpha_{i-j}$.

Let $A = k\sum_{i=1}^{d}\frac{{\alpha_i}^{w-1}}{P(i-1)\prod_{j=1}^{d-i}1-k^j}$ and $B = \sum_{i=1}^{d}\frac{{\alpha_i}^{w-1}\alpha_{i-d}}{P(i-1)\prod_{j=1}^{d-i}1-k^j}$. So according to our assumption, $A + B < k-1$.

However, writing out the terms in the expression for A yields:

$$A = k\left(\frac{\alpha_d^{w-1}}{P(d-1)} - \frac{\alpha_{d-1}^{w-1}}{(k-1)P(d-2)} + \frac{\alpha_{d-2}^{w-1}}{(k-1)(k^2-1)P(d-3)} - ...\right)$$

$$= k\left(\frac{\alpha_d^{w-1}}{P(d-1)} - \frac{\alpha_{d-1}^{w-1}}{(k-1)P(d-2)} + o(\frac{1}{k^2}) - ...\right)$$

$$\geq k\left(\frac{\alpha_d^{w-1}}{P(d-1)} - \frac{\alpha_{d-1}^{w-1}}{(k-1)P(d-2)}\right)$$

It is not difficult to see that $\frac{\alpha_{d-2}^{w-1}}{(k-1)(k^2-1)P(d-3)} - ... \leq o(\frac{1}{k^2})$, since, $P(i) < P(j)$ when $i > j$ and α_i^{w-1} decreases as $i \to 0$. Similarly, we can show that,

$$B \geq -\frac{\alpha_{d-1}^{w-1}}{P(d-2)}$$

Thus we get,

$$k\frac{\alpha_d^{w-1}}{P(d-1)} - \frac{\alpha_{d-1}^{w-1}}{P(d-2)}\left(1 + \frac{k}{k-1}\right) < k-1$$

Since we want to get an upper bound for $d_0(w,k)$ we assume that w is sufficiently large. More precisely, we let $w \geq ck^{d-1} + 1$ where $0 < c < 1$. But then,

$$\alpha_{d-1}^{w-1} = \left(1 - \frac{1}{k^{d-1}}\right)^{w-1} \leq \left(1 - \frac{1}{k^{d-1}}\right)^{ck^{d-1}}$$

$$\leq e^{-c} \approx \frac{3}{5}$$

where we take $c = \frac{1}{2}$. Putting this value for α_{d-1}^{w-1} in the main equation and dividing both

sides by k we get,

$$\frac{\alpha_d^{w-1}}{P(d-1)} < 1 - \frac{1}{k} + \frac{3}{5}\frac{2k-1}{k(k-1)P(d-1)},$$

since $P(d-1) < P(d-2)$. But,

$$\begin{aligned} P(d-1) > P(d) &= \left(1-\frac{1}{k}\right)\left(1-\frac{1}{k^2}\right)...\left(1-\frac{1}{k^{d-1}}\right) \\ &= 1 - \frac{1}{k} - \frac{1}{k^2} + \frac{1}{k^5} + \frac{1}{k^7} + ... \geq 1 - \frac{1}{k} - \frac{1}{k^2} \end{aligned}$$

And $P(d-1) < 1 - \frac{1}{k}$. Substituting these upper and lower bound of $P(d-1)$ to the LHS and RHS of the expression respectively we get,

$$k\frac{\alpha_d^{w-1}}{k-1} < 1 - \frac{1}{k} + \frac{3}{5}\frac{(2k-1)k}{(k-1)(k^2-k-1)} \leq 1 + \frac{\gamma(k)}{k}$$

Where $0 < \gamma(k) < 1$ for $k \geq 4$. So we have,

$$1 - \frac{w}{k^d} < \left(1-\frac{1}{k^d}\right)^w < \alpha_d^{w-1} < \left(1+\frac{\gamma(k)}{k}\right)\left(1-\frac{1}{k}\right) < 1 - \frac{1}{k^2}$$

So we get, $d < \log_k w + 2$. This is the upper bound on $d_0(w,k)$ that we have sought.

Let $d_0 \leq \beta + 3$, where $\beta = \lfloor \log_k w \rfloor$. Hence, for $k \geq 4$,

$$
\begin{aligned}
u(w) &= \sum_{d=0}^{w-1} \eta_1^d a(w,d) \\
&\leq \sum_{d=0}^{d_0} \eta_1^d a(w,d) + \frac{7}{3}\eta_1^{d_0} a(w,d_0+1) \\
&\leq \sum_{d=0}^{\beta} \eta_1^d k^d + {\eta_1}^{\beta+1}(a(w,\beta+1) + a(w,\beta+2) + a(w,\beta+3)) + \frac{7}{3}\eta_1^{\beta+3} a(w,\beta+4)
\end{aligned}
$$

Here we observe that, $a(w,\beta+i) \leq w$ for $1 \geq i \geq 4$. Substituting these bounds we get,

$$
u(w) \leq \frac{(\eta_1 k)^{\beta+1} - 1}{\eta_1 k - 1} + c_1 w \eta_1^{\beta+1}
$$

where $c_1 < 5$ is a constant. Thus

$$
\begin{aligned}
u(w) &\leq \frac{(\eta_1 k)^{\beta}(\eta_1 k)}{\eta_1 k - 1} + c_1 w \eta_1^{\beta+1} \\
&\leq c_2 (\eta_1 k)^{\beta} + c_1 w \eta_1^{\beta+1} \\
&\leq c_2 (\eta_1 k)^{\lfloor \log_k w \rfloor} + c_1 w \eta_1^{\lfloor \log_k w \rfloor + 1} \\
&\leq c_2 (\eta_1 k)^{\log_k w} + c_1 w \eta_1^{\log_k w} \\
&\leq (c_1 + c_2) w \eta_1^{\log_k w} \leq c_3 w^{1 - \frac{1}{\log k} + \log_k (1 + \frac{2}{k+1})}
\end{aligned}
$$

where, $c_2 \leq \frac{14}{9}$ and c_3 are constants. Which leads to

$$u(w) \leq O\left(w^{1-\frac{1}{\log k}+\log_k\left(1+\frac{2}{k+1}\right)}\right).$$

This proves Theorem 3. □

Corollary 3. MaxPartition runs in $O\left(kn^{2-\frac{1}{\log k}+\log_k\left(1+\frac{2}{k+1}\right)}\log n\right)$ in expectation.

Proof. From Theorem 1 and the first paragraph of Section 4.2 we see that the runtime of MaxPartition is $O(knu(w)\log h)$. Since computing $\mathcal{O}(p,q)$ between pairs of vectors takes $O(k)$ time. Using the upper-bound of $u(w)$ and the fact that $w, h \leq n$ we get the runtime of MaxPartition as claimed above.

□

3.4.3 Extension of MaxPartition to Other Distributions

It is quite easy to modify the analysis in the previous section for points sampled from an arbitrary distributions. That is, for each point $x \in P$, we pick $x[i]$ from $D[0,1]$ for all $1 \leq i \leq k$. Where, $D[0,1]$ is a distribution function on $[0,1]$. For such a P we can extend our run-time analysis as follows: first we look at the probability $\eta_2(k)$ in this new setting. We note that we do not use the value of $\eta_2(k)$ explicitly, just the fact that its same for each dimension. Clearly, this probability still remains independent of the dimension j in the new setting. This applies to $\eta_1(k)$ also, as can be seen below:

$$\Pr[p[j] > q[j] \mid \mu(p) > \mu(q)] = \int_0^{\mu(p)} \int_0^{s} dF_{[0,\mu(q)]}(t) dF_{[0,\mu(p)]}(s)$$

Where, $F_{[a,b]}(t)$ is the c.d.f of the distribution $D[a,b]$ in the interval $[a,b]$. Clearly, this probability is independent of j. The same can be said about $\eta_2(k)$. In fact we can forgo the assumption that each point is identically distributed. That is for each $x \in P$ we

pick $x[i]$ from $D_x[0,1]$ for all i. Even in this setting both $\eta_1(k,p,q)$ and $\eta_2(k,p,q)$ will be invariant under change of dimensions even though they will now depend also on the pair p,q. Furthermore, Theorem 2 still holds under this new setting without needing to make any changes. The expected run-time of the MaxPartition procedure, which depends on η_1, will change and would depend on the distributions $D_x[0,1]$. However, we observed that $\eta_1(k,p,q) \leq 1 - 1/k$ otherwise $p \succ q$. Hence, using this upper bound on $\eta_1(k,p,q)$ we can determine an upper bound on the expected run-time by substituting the appropriate value in place of η_1 in the proof of Theorem 3. Which leads to a run-time of $O(k^2 n^{1+\log_k(k-1)})$ in this new setting.

3.4.4 A Summary of Results

We summarize the main results as follows:

i. *k is a constant.* From Corollary 3 we can easily verify that MaxPartition has a runtime that is bounded by $O(n^{2-\delta(k)})$ where $\delta(k) > 0$.

ii. *k is some function of n.* For any k the runtime of our algorithm is bounded by $O(kn^2)$ in the worst case. This bound does not hold for the divide-and-conquer algorithm in [13]. Also, The proposed algorithm never admits a quasi-polynomial runtime unlike any of the previously proposed non-trivial algorithms.

iii. *When points in P have arbitrary distributions.* For this case we get a run-time of $O(k^2 n^{1+\log_k(k-1)})$ in expectation, regardless of the underlying distribution. It can be shown that the run-time complexity grows slower than that in [13] whenever $k = \Omega(\log n)$.

Chapter 4: Concluding Remarks

In this thesis we revisit random partials orders as introduced in [2]. We study a fundamental property of random orders: the expected size distribution of different maximal layers. For the case when the random order consists of points in a plane we formulate an expression for the size distribution of the maximal sets. It turns out that the expected size of different maximal layers depends on computing the number of permutations having increasing subsequence of a certain size. Since the latter problem does not yet have any closed form, it makes enumerative formulation of the size distribution of maximal sets difficult. Our experimental simulations shows that the size distribution follows a bell shape pattern. It is likely that shape (in terms of the layers) of a random partial order is like an "onion"; size of the latter layers decreases sharply after some "fat" layer. It can be conjectured that the maximum size of any layer has a similar bound to that of the width of the poset.

In the second half of this thesis we explore the computational aspect of the problem. That is we investigate the problem of identifying the different maximal layers of a random order. In particular we propose a randomized algorithm whose expected runtime is better than current best known, whenever the number of dimensions is large ($\Omega(\log n/\log\log n)$). Although the bonds provided here hold only in expectation, we note that in the worst case the algorithm only takes $O(kn^2)$ time. This is in contrast to existing deterministic algorithms that have super-polynomial run-times (in n) whenever k is $\Omega(\log n/\log\log n)$.

It would be interesting to see if the proposed algorithm can be made deterministic without going over the $o(n^2)$ upper-bound for constant k. Another area of interest is to see if the algorithm can be extended to inputs that are not necessarily random orders. For example in Section 3.4.4 we show how we can use it for the case when the input set of vectors were chosen with some arbitrary distribution and not uniformly at random. Ideally,

one would hope to extend it for arbitrary posets (of vectors). Lastly, it is worth looking into the application of HST for other similar problems in computational geometry. In particular it would be interesting to know whether HST or some modification of it can be used to store points within a convex layer instead of a maximal layer. This may lead to a faster algorithm for the convex layers problem in higher dimension.

Appendix A: Publications

List of publications arising out of materials from this thesis, in reverse chronological order.

1. Indranil B., Dana Richards, "Computing Maximal Layers Of Points in $E^{f(n)}$". Accepted in Latin American Theoretical Informatics Symposium (LATIN) (2016).

2. Indranil B., Dana Richards, "Distribution Of Maximal Layers Of Random Orders", Congressus Numerantium 225, 211-216 (2015).

3. Indranil B., Dana Richards, "On Maximal Layers Problem". Forty-Sixth Southeastern International Conference on Combinatorics, Graph Theory, and Computing (Talk) (2015).

Bibliography

[1] G. Brightwell, "Random k-dimensional orders: Width and number of linear extensions," *Order*, vol. 9, no. 4, pp. 333–342, 1992.

[2] P. Winkler, "Random orders," *Order*, vol. 1, no. 4, pp. 317–331, 1985.

[3] P. Erdös and A. Rényi, "On random graphs I," *Publ. Math. Debrecen*, vol. 6, pp. 290–297, 1959.

[4] M. Karoński and A. Ruciński, "The origins of the theory of random graphs," in *The Mathematics of Paul Erdös I.* Springer, 1997, pp. 311–336.

[5] J. L. Bentley, H. Kung, M. Schkolnick, and C. D. Thompson, "On the average number of maxima in a set of vectors and applications," *Journal of the ACM (JACM)*, vol. 25, no. 4, pp. 536–543, 1978.

[6] M. J. Golin, "How many maxima can there be?" *Computational Geometry*, vol. 2, no. 6, pp. 335–353, 1993.

[7] G. Brightwell, "Models of random partial orders," *Surveys in combinatorics*, pp. 53–83, 1993.

[8] J. Baik, P. Deift, and K. Johansson, "On the distribution of the length of the longest increasing subsequence of random permutations," *Journal of the American Mathematical Society*, vol. 12, no. 4, pp. 1119–1178, 1999.

[9] C. A. Tracy and H. Widom, "Level-spacing distributions and the airy kernel," *Communications in Mathematical Physics*, vol. 159, no. 1, pp. 151–174, 1994.

[10] F. F. Yao, *On finding the maximal elements in a set of plane vectors.* Department of Computer Sciences, University of Illinois at Urbana-Champaign, 1974.

[11] F. Nielsen, "Output-sensitive peeling of convex and maximal layers," *Information Processing Letters*, vol. 59, no. 5, pp. 255–259, 1996.

[12] H. Edelsbrunner, *Algorithms in combinatorial geometry.* Springer Science & Business Media, 2012, vol. 10.

[13] M. T. Jensen, "Reducing the run-time complexity of multiobjective eas: The nsga-ii and other algorithms," *Evolutionary Computation, IEEE Transactions on*, vol. 7, no. 5, pp. 503–515, 2003.

[14] J. L. Bentley, "Multidimensional divide-and-conquer," *Communications of the ACM*, vol. 23, no. 4, pp. 214–229, 1980.

[15] H.-T. Kung, F. Luccio, and F. P. Preparata, "On finding the maxima of a set of vectors," *Journal of the ACM (JACM)*, vol. 22, no. 4, pp. 469–476, 1975.

[16] J. Matousek, "Computing dominances in en," *Information processing letters*, vol. 38, no. 5, pp. 277–278, 1991.

[17] R. Impagliazzo, S. Lovett, R. Paturi, and S. Schneider, "0-1 integer linear programming with a linear number of constraints," *arXiv preprint arXiv:1401.5512*, 2014.

Curriculum Vitae

Indranil Banerjee graduated with honors from Bengal Engineering College & Model School, Howrah, India in 2004. He went on to study Electrical Engineering at National Institute of Technology, Durgapur, India, where he was awarded a Bachelor of Technology (Hons.) in 2009. After graduation he worked primarily in the field of computer vision. He was employed at Videonetics Technology Pvt. Ltd. and subsequently to TechBLA Corp. from 2009 to 2012 in the role of a software engineer. During this time he worked with Dr. Prasun Das (SQC-OR) and Dr. Sanghamitra Bandyopadhyay (Machine Learning Unit) at Indian Statistical Institute, Kolkata, on evolutionary multi-objective optimization. He began his graduate studies at George Mason University in fall 2012.