Maxima-finding algorithms for multidimensional samples: A two-phase approach

Wei-Mei Chen[a], Hsien-Kuei Hwang[b], Tsung-Hsi Tsai[b]

[a]*Department of Electronic Engineering, National Taiwan University of Science and Technology, Taipei 106, Taiwan*
[b]*Institute of Statistical Science, Academia Sinica, Taipei 115, Taiwan*

Abstract

Simple, two-pahse algorithms are devised for finding the maxima of multidimensional point samples, one of the very first problems studied in computational geometry. The algorithms are easily coded and modified for practical needs. The expected complexity of some measures related to the performance of the algorithms is analyzed. We also compare the efficiency of the algorithms with a few major ones used in practice, and apply our algorithms to find the maximal layers and the longest common subsequences of multiple sequences.

Key words: maximal points, computational geometry, Pareto optimality, sieve algorithms, dominance, multiobjective optimization, skyline, average-case analysis of algorithms.

1. Introduction

A point $\mathbf{p} \in \mathbb{R}^d$ is said to *dominate* another point $\mathbf{q} \in \mathbb{R}^d$ if the coordinatewise difference $\mathbf{p} - \mathbf{q}$ has only nonnegative coordinates and $\mathbf{p} - \mathbf{q}$ is not identically a zero vector, where the dimensionality $d \geq 1$. For convenience, we write $\mathbf{q} \prec \mathbf{p}$ or $\mathbf{p} \succ \mathbf{q}$. The non-dominated points in a sample are called the *maxima* or *maximal points* of the sample. Note that there may be two identical points that are both maxima according to our definition of dominance. Since there is no total order for multidimensional points when $d > 1$, such a dominance relation among points has been one of the simplest and widely used partial orders. We can define dually the corresponding *minima* of the sample by reversing the direction of the dominance relation.

1.1. Maxima in diverse scientific disciplines

Daily lives are full of tradeoffs or multi-objective decision problems with often conflicting factors; the numerous terms appeared in different scientific fields reveal the importance and popularity of maxima in theory, algorithms, applications and practice: maxima (or vector maxima) are sometimes referred to as *nondominance*, *records*, *outer layers*, *efficiency*, or *noninferiority* but are more frequently known as *Pareto optimality* or *Pareto efficiency* (with the natural derivative *Pareto front*) in econometrics, engineering, multi-objective optimization, decision making, etc. Other terms used with essentially the same denotation include *admissibility* in statistics, *Pareto front* (and the corresponding notion of *elitism*) in evolutionary algorithms, and *skyline* in database language; see [2, 16, 23, 24] and the references therein and the books [20, 21, 27] for more information. They also proved useful in many computer algorithms and are closely related to several well-known problems, including convex hulls, top-k queries, nearest-neighbor search, largest empty rectangles, minimum independent dominating set in permutation graphs, enclosure problem for rectilinear d-gon, polygon decomposition, visibility and illumination, shortest path problem, finding empty slimplices, geometric containment problem, data swapping, grid placement problem, and multiple longest common subsequence problem to which we will apply our algorithms later; see [16, 50] for more references.

We describe briefly here the use of maxima in the contexts of database language and multi-objective optimization problems using evolutionary algorithms.

Skylines in database queries are nothing but minima. A typical situation where the skyline operator arises is as follows; see [14] for details. Travelers are searching over the Internet for cheap hotels near the beach. Since the two criteria "lower price" and "shorter distance" are generally conflicting with each other and since there are often too many hotels to choose from, one is often interested in those hotels that are non-dominated according to the two criteria; here dominance is defined using minima. Much time will be saved if the search or sort engine can automatically do this and filter out those that are

Email addresses: wmchen@mail.ntust.edu.tw (Wei-Mei Chen), hkhwang@stat.sinica.edu.tw (Hsien-Kuei Hwang), chonghi@stat.sinica.edu.tw (Tsung-Hsi Tsai)

dominated for database queriers (by, say clicking at the skyline operator). On the other hand, frequent spreadsheet users would also appreciate such an operator, which can find the maxima, minima or skyline of multidimensional data by simple clicks.

In view of these and many other natural applications such as e-commerce, multivariate sorting and data visualization, the skylines have been widely and extensively addressed in recent database literature, notably for low- and moderate-dimensional data, following the pioneering paper [14]. In addition to devising efficient skyline-finding algorithms, other interesting issues include top-k representatives, progressiveness, absence of false hits, fairness, incorporation of preference, and universality. A large number of skyline-finding algorithms have been proposed for various needs; see, for example, [5, 14, 32, 44, 47, 49, 54] and the references therein.

On the other hand, the last decade has witnessed a tremendous growth of interest in the study of multi-objective evolutionary algorithms (MOEAs), where the idea of maxima also appeared naturally in the form of non-dominated solutions (or elites). MOEAs provide a popular approach for multi-objective optimization, which identify the most feasible solutions lying on the Pareto front under various (often conflicting) constraints by repeatedly finding non-dominated solutions based on biological evolutionary mechanisms. These algorithms have turned out to be extremely fruitful in diverse engineering, industrial and scientific areas, as can be witnessed by the huge number of citations many papers on MOEA have received so far. Some popular schemes in this context suggested the maintenance of an explicit archive/elite for all non-dominated solutions found so far; see below and [28, 42, 46, 55, 56] and the references therein. See also [19] for an interesting historical overview.

Finally, maxima also arises in a random model for river networks (see [3, 10]) and in an interesting statistical estimate called "layered nearest neighbor estimate" (see [11]).

1.2. Maxima, maximal layers and related notions

Maxima are often used for some ranking purposes or used as a component problem for more sophisticated situations. Whatever the use, one can easily associate such a notion to define multidimensional sorting procedures. One of the most natural ways is to "peel off" the current maxima, regarded as the first-layer maxima, and then finding the maxima of the remaining points, regarded then as the second-layer maxima, and so on until no point is left. The total number of such layers gives rise to a natural notion of *depth*, which is referred to as the *height* of the corresponding random, partially ordered sets in [13]. Such a maximal-layer depth is nothing but the length of the longest increasing subsequences in random permutations when the points are selected uniformly and independently from the unit square, a problem having attracted widespread interests, following the major breakthrough paper [4].

On the other hand, the maximal layers are closely connected to chains (all elements comparable) and antichains (all elements incomparable) of partially ordered set in order theory, an interesting result worthy of mention is the following dual version of Dilworth's theorem, which states that the size of the largest chain in a finite partial order is equal to the smallest number of antichains into which the partial order may be partitioned; see, for example, [40] for some applications.

In addition to these aspects, *maximal layers* have also been widely employed in multi-objective optimization applications since the concept was first suggested in Goldberg's book [33]. Based on identifying the maximal layer one after another, Srinivas and Deb [52] proposed the non-dominated sorting genetic algorithm (NSGA) to simultaneously find multiple Pareto-optimal points, which was later on further improved in [22], reducing the time complexity from $O(dn^3)$ to $O(dn^2)$. Jensen [39] then gave a divide-and-conquer algorithm to find the maximal layers with time complexity $(n(\log n)^{d-1})$; see Section 5 for more details.

In the contexts of multi-objective optimization problems, elitism usually refers to the mechanism of storing some obtained non-dominated solutions into an external archive during the process of MOEAs because a non-dominated solution with respect to its current data is not necessarily non-dominated with respect to the whole feasible solutions. The idea of elitism was first introduced in [56] and is regarded as a milestone in the development of of MOEAs [19]. Since the effectiveness of this mechanism relies on the size of the external non-dominated set, an elite archive with limited size was suggested to store the truncated non-dominated sets [42, 56], so as to avoid the computational costs of maintaining all non-dominated sets. Nevertheless, restricting the size of archive reduces the quality of solutions; more efficient storages and algorithms are thus studied for unconstrained elite archives; see for example [28, 39, 48].

1.3. Aim and organization of this paper

Due to the importance of maxima, a large number of algorithms for finding them in a given sample of points have been proposed and extensively studied in the literature, and many different design paradigms were introduced including divide-and-conquer, sequential, bucket or indexing, selection, and sieving; see [9, 8, 30, 36, 41, 45, 50, 51] and the survey [16] for more information. Quite naturally, practical algorithms often merge more than one of the design paradigms for better performance.

Despite the large number of algorithms proposed in the literature, there is still need of simpler and practically efficient algorithms whose performance does not deteriorate too quickly in massive point samples as the number of maximal points grows, a property which we simply refer to as "scalable". This is an increasingly important property as nowadays massive data sets or data streams are becoming ubiquitous in diverse areas.

Although for most practical ranking and selecting purposes, the notion of maxima is most useful when the number of maxima is not too large compared with the sample size, often there is no a priori information on the number of maxima before computing them. Furthermore, the number of maxima may be very close to n when the dimension d grows; see [1].

Also a general-purposed algorithm may in practice face the situation of data samples with very large standard deviation for their maxima. From known probabilistic theory of maxima (see [1] and the references therein), the expected number of maxima and the corresponding variance can in two typical random models grow either in $O((\log n)^{d-1})$ when the coordinates are roughly independent or in $O(n^{1-1/d})$ when the coordinates are roughly negatively dependent, both O-terms here referring to large n, the sample size, and fixed d, the dimensionality. In particular, in the planar case, there can be $\sqrt{n}$ number of maxima on average for roughly negatively correlated coordinates, in contrast to $\log n$ for independent coordinates; see also [6, 34] for the "gap theorem" and [25] for a similar $\sqrt{n}$ vs $\log n$ effect (reflecting dependence or independence) on random Cartesian trees. Since the maximal points can be very abundant with large standard deviations, more efficient and more uniformly scalable algorithms are needed.

We propose in this paper two simple techniques to achieve scalability: the first technique is to reduce the maxima-finding to a two-phase records-finding procedure, giving rise to a no-deletion algorithm, which largely simplifies the design and maintenance of the data structure used. The second technique is the introduction of bounding box in the corresponding tree structure for storing the current maxima, which reduces significantly the deterioration of efficiency in higher dimensions. The combined use of both techniques on k-d trees turns out to be very efficient, easily coded and outperforms many known efficient algorithms under reasonable random models. Some preliminary results on the use of k-d trees for finding maxima of appeared in [17].

This paper is organized as follows. In the next section, we briefly describe some existing algorithms proposed in the diverse literature, focusing on the two most popular and representative paradigms: divide-and-conquer and sequential. Section 3 gives details of the proposed techniques, implementation on k-d trees, and diverse aspects of further improvements. A comparative discussion will also be given with major known algorithms. Analytic and empirical aspects of the performance of the algorithms will be discussed in Section 4. Finally, we apply our algorithm to the problems of finding maximal layers and that of finding multiple longest common subsequence in Section 5, where the efficiency of our algorithm is tested on several data sets.

Throughout this paper, $\mathbf{Max}(\mathbf{P})$ *always denotes the maxima of the sequence of points* $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$.

2. Known maxima-finding algorithms—a brief account

In view of the large amount of algorithms with varying characters appeared in the literature, it is beyond the scope of this paper to provide a full description of all existing algorithms. Instead, we give a brief account here on divide-and-conquer and sequential algorithms; see [16] and the references there for other algorithms.

2.1. Divide-and-conquer algorithms

Divide-and-conquer algorithms were first proposed by Kung et al. [45] with the worst-case time complexity of order $n(\log n)^{d-2+\delta_{d,2}}$ for dimensionality $d \geq 2$, where n is the number of points and $\delta_{a,b}$ denotes the Kronecker delta function. Bentley [8] schematized a multidimensional divide-and-conquer paradigm, which in particular is applicable to the maxima-finding problem with the same worst-case complexity. Gabow et al. [30] later improved the complexity to $O(n(\log n)^{d-3}\log\log n)$ for $d \geq 4$ by scaling techniques. Output-sensitive algorithms with complexity of order $n(\log(M+1))^{d-2+\delta_{d,2}}$ were devised in [41], where M denotes the number of maxima.

The typical pattern of most of these algorithms is as follow.

Algorithm Divide-and-conquer
//Input: A sequence of points $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ in $\mathbb{R}^d$
//Output: $\mathbf{Max}(\mathbf{P})$
begin
  if $n \leq 1$ **then return**$(\{\mathbf{p}_1, \ldots, \mathbf{p}_n\})$
  else return Filter-out-false-maxima(Divide-and-conquer$(\{\mathbf{p}_1, \ldots, \mathbf{p}_{\lfloor n/2 \rfloor}\})$,
    Divide-and-conquer$(\{\mathbf{p}_{\lfloor n/2 \rfloor+1}, \ldots, \mathbf{p}_n\})$
end

Here Filter-out-false-maxima($\mathbf{P}, \mathbf{Q}$) drops maxima in $\mathbf{Q}$ (in $\mathbf{P}$) dominated by maxima in $\mathbf{P}$ (in $\mathbf{Q}$).

These divide-and-conquer algorithms are generally characterized by their good theoretic complexity in the worst case, simple structural decompositions in concept but low competitiveness in practical and typical situations with sequential algorithms, although it is known that most divide-and-conquer algorithms have linear expected-time performance under the usual hypercube random model, or more generally when the expected number of maxima is of order $o(n^{1-\varepsilon})$; see [23, 29]. Variants of them have however been adapted in the skyline and evolutionary computation contexts; see for example [49] for skylines and [39] for MOEAs.

2.2. Sequential algorithms

The most widely-used procedure for finding non-dominated points in multidimensional samples has the following incremental, on-line, one-loop pattern (see [9, 45]).

Algorithm Sequential
//Input: A sequence of points $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ in $\mathbb{R}^d$
//Output: Max($\mathbf{P}$)
begin
    $\mathbf{M} := \{\mathbf{p}_1\}$ //$\mathbf{M}$: a data structure for storing the current maxima
    for $i := 2$ **to** n **do**
        if no point in $\mathbf{M}$ dominates $\mathbf{p}_i$ **then** //updating $\mathbf{M}$
            delete $\{\mathbf{q} \in \mathbf{M} : \mathbf{q} \prec \mathbf{p}_i\}$ from $\mathbf{M}$
            insert $\mathbf{p}_i$ into $\mathbf{M}$
end

The algorithm is a natural adaptation of the one-dimensional maximum-finding loop, which represents the very first algorithm analyzed in details in Knuth's *Art of Computer Programming* books [43]. It runs by comparing points one after another with elements in the data structure $\mathbf{M}$, which stores the maxima of all elements seen so far, called *left-to-right maxima* or *records*; it moves on to the next point $\mathbf{p}_{i+1}$ if the new point $\mathbf{p}_i$ is dominated by some element in $\mathbf{M}$, or it removes elements in $\mathbf{M}$ dominated by the new point $\mathbf{p}_i$ and accepts the new point $\mathbf{p}_i$ into $\mathbf{M}$.

For dimensions $d \geq 2$, such a simple design paradigm was first proposed in [45] (with an additional pre-sorting stage with respect to one of the coordinates) and the complexity was analyzed for $d = 2$ and $d = 3$. To achieve optimal worst-case complexity for $d = 3$, they used AVL-tree (a simple, balanced variant of binary search tree). The simpler implementation using a linear list (and without any pre-sorting procedure) was discussed first in the little known paper [36] and later in greater detail in [9], in particular with the use of move-to-front self-adjusting heuristic.

The Sequential algorithm, also known as block-nested-loop algorithm [49], is most efficient when the number of maxima is a small function of n such as powers of logarithm (see [26, 35]), but deteriorates rapidly when the number of maxima is large. In addition to list employed in [9] to store the maxima for sequential algorithms, many varieties of tree structures were also proposed in the literature: quad trees in [36, 48], R-trees in [44], and d-ary trees in [51]; see also [49]. But these algorithms become less efficient (in time bound and in space utilization) as the dimensionality of data increases, also the maintenance is more complicated. We will see that the use of k-d trees is preferable in most cases; see also [16] for the use of binary search trees for $d = 2$.

3. A two-phase sequential algorithm based on k-d trees using bounding boxes

We present in this section our algorithm based on the ideas of multidimensional non-dominated records, bounding boxes, and k-d trees. Further refinements of the algorithm will also be discussed. We then compare our algorithm with a few major ones discussed in the literature.

3.1. The design techniques

We introduce in this subsection multidimensional non-dominated records, k-d trees and bounding boxes, and will apply them later for finding maxima. In practice, each of these techniques can be incorporated equally well into other techniques for finding maxima.

3.1.1. Multidimensional non-dominated records

Except for simple data structures such as list, the deletion performed in algorithm Sequential is often the most complicated step as it requires a structure re-organization after the removal of the dominated elements. It is then natural to see if there are algorithms avoiding or reducing deletions.

Note that in the special case when $d = 1$, the two steps "deletion" and "insertion" in algorithm Sequential actually reduce to one, and the inserted elements are nothing but the records (or record-breaking elements, left-to-right maxima, outstanding elements, etc.). Recall that an element $\mathbf{p}_j$ in the sequence of reals $\{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ is called a record if $\mathbf{p}_j$ is not dominated by any element in $\{\mathbf{p}_1, \ldots, \mathbf{p}_{j-1}\}$.

The crucial observation is then based on extending the one-dimensional records to higher dimensions.

Definition (d-dimensional non-dominated records). A point $\mathbf{p}_j$ in the sequence of points in $\mathbb{R}^d$ $\{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ is said to be a *d-dimensional non-dominated record* of the sequence $\{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ if $\mathbf{p}_j$ is not dominated by $\mathbf{p}_i$ for all $1 \leq i < j$. We also define $\mathbf{p}_1$ to be a non-dominated record.

Such non-dominated records are called "weak records" in [31], but this term seems less informative; see also [24] for a different use of records. *For simplicity, we write, throughout this paper, records to mean non-dominated records when no ambiguity will arise.* Other notions of records can be found in [31, 38] and the references therein.

For convenience, denote by $\mathbf{Rec}(\mathbf{P})$ the set of records of $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$.

Lemma 1. *For any given set of points $\{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$,*

$$\mathbf{Max}(\{\mathbf{p}_1, \ldots, \mathbf{p}_n\}) = \mathbf{Rec}(\overline{\mathbf{Rec}(\{\mathbf{p}_1, \ldots, \mathbf{p}_n\})}),$$

where $\overline{\{\mathbf{q}_1, \ldots, \mathbf{q}_k\}} := \{\mathbf{q}_k, \ldots, \mathbf{q}_1\}$ denotes the reversed sequence.

In words, if $\{\mathbf{q}_1, \ldots, \mathbf{q}_k\}$ represents the records of the sequence $\{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$, then the maxima of $\{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ is equal to the records of the sequence $\{\mathbf{q}_k, \mathbf{q}_{k-1}, \ldots, \mathbf{q}_1\}$.

Proof. We prove by contradiction. Assume that there are two points $\mathbf{p}_i$ and $\mathbf{p}_j$ in the set

$$\mathbf{Rec}(\overline{\mathbf{Rec}(\{\mathbf{p}_1, \ldots, \mathbf{p}_n\})})$$

such that $\mathbf{p}_i \succ \mathbf{p}_j$. If $i < j$, then $\mathbf{p}_j$ cannot be a record and thus cannot be a member of the set $\mathbf{Rec}(\{\mathbf{p}_1, \ldots, \mathbf{p}_n\})$, a contradiction. On the other hand, if $i > j$, then $\mathbf{p}_j$ is a record and is included in the set $\mathbf{Rec}(\{\mathbf{p}_1, \ldots, \mathbf{p}_n\})$, but then after the order being reversed, it cannot be a record since it is dominated by $\mathbf{p}_i$, again a contradiction. □

Another interesting property regarding the connection between records and maxima is the following.

Corollary 1. *In algorithm Sequential for finding maxima, the points $\mathbf{p}_i$ to be inserted in the for-loop are necessarily the records, while those deleted are records but not maxima.*

3.1.2. A two-phase sequential algorithm

Lemma 1 provides naturally a two-phase, no-deletion algorithm for finding maxima: in the first phase, we identify the records, and in the second, we find the records of the reversed sequence of the output of the first phase (so as to remove the non-maximal records); an example of seven planar points is given in Figure 1. In other terms, we perform only the insertion in algorithm Sequential in the first phase, postponing the deletion to be carried out in the second.

The precise description of the algorithm is given as follows. Note that in the algorithm a list $\mathbf{R}$ is used to store the records and has to preserve their relative orders.

Algorithm Two-Phase
  //**Input**: A sequence of points $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$
  //**Output**: $\mathbf{Max}(\mathbf{P})$
  begin
   // Phase 1
    $\mathbf{R} := \{\mathbf{p}_1\}$ // $\mathbf{R}$ stores the non-dominated records
    $k := 1$ // k counts the number of records
    for $i := 2$ **to** n **do**
      if $\mathbf{p}_i$ is not dominated by any point in $\mathbf{R}$ **then**
        $k := k + 1$
        insert $\mathbf{p}_i$ at the end of $\mathbf{R}$ // so as to retain the input order

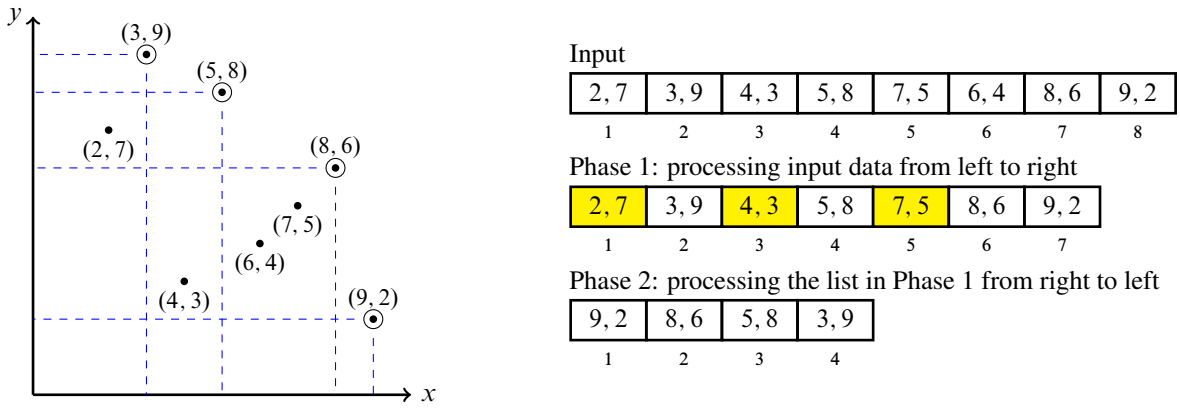

Figure 1: The maxima of the point sample $\{(2,7),(3,9),(4,3),(5,8),(7,5),(6,4),(8,6),(9,2)\}$ are marked by circles. After Phase 1, $(2,7)$, $(4,3)$ and $(7,5)$ are still left in the list though they are not maximal points. But after Phase 2, the resulting list contains all maximal points.

// After the for-loop, $\mathbf{R} = \{\mathbf{p}_{j_1}, \ldots, \mathbf{p}_{j_k}\}$, where $j_1 < j_2 < \cdots < j_k$.
// Phase 2
$\mathbf{M} := \{\mathbf{p}_{j_k}\}$ // $\mathbf{M}$ stores the maxima
for $i := k-1$ **downto** 1 **do**
if $\mathbf{p}_{j_i}$ is not dominated by any point in $\mathbf{M}$ **then** insert $\mathbf{p}_{j_i}$ in $\mathbf{M}$
end

The correctness of Algorithm Two-Phase is guaranteed by Lemma 1.

While the two-phase procedure may increase the total number of comparisons made, the real scalar comparisons made can actually be simplified since we need only to detect if the incoming element is dominated by some element in the list $\mathbf{R}$, and there is no need to check the reverse direction that the incoming element dominates some element in $\mathbf{R}$. Thus the code for the detection of dominance or non-dominance is simpler than that of algorithms performing deletions. Furthermore, for each vector comparison, it is not necessary to check all coordinates unless one element is dominated by the other. Briefly, the two-phase algorithm splits the comparisons made for checking dominance between elements in two directions.

3.1.3. The k-d trees

The data structure k-d tree (or multidimensional binary search tree) is a natural extension of binary search tree for multidimensional data, where k denotes the dimensionality. For more notational convenience and consistency, we also write, throughout this paper, d as the dimensionality (but still use k-d tree instead of d-d tree). It was first invented by Bentley [7]. The idea is to use each of the d coordinates cyclically at successive levels of the binary tree as the *discriminator* and direct points falling in the subtrees. If a node holding the point $\mathbf{r} = (r_1, \ldots, r_d)$ in a k-d tree has the ℓ-th coordinate as the discriminator, then, for any node holding the point $\mathbf{w} = (w_1, \ldots, w_d)$ in the subtrees of $\mathbf{r}$, we have the relation $w_\ell < r_\ell$ if $\mathbf{w}$ lies in the left-subtree of $\mathbf{r}$, $w_\ell \geq r_\ell$ if $\mathbf{w}$ lies in the right-subtree of $\mathbf{r}$. The children of $\mathbf{r}$ then move on to the $(\ell \bmod d) + 1$-st coordinate as the discriminator. A two-dimensional example is given in Figure 2.

3.1.4. Bounding-boxes

Bounding boxes are simple techniques for improving the performance of many algorithms, especially those dealing with intersecting geometric objects, and have been widely used in many theoretical and practical situations.

The application of bounding boxes is straightforward. Let $\mathbf{u_r} = (u_1, \ldots, u_d)$, where u_i is the maximum among all the i-th coordinates of points in the subtree rooted at $\mathbf{r}$. Then $\mathbf{u_r}$ is defined to be the *upper bound* of the subtree rooted at $\mathbf{r}$ or simply the upper bound of the node $\mathbf{r}$. Similarly, define $\mathbf{v_r} = (v_1, \ldots, v_d)$ to be the *lower bound* of the subtree rooted at $\mathbf{r}$, where v_i is the minimum among all the i-th coordinates of points in the subtree rooted at $\mathbf{r}$. A simple example of three-dimensional points is given in Figure 3. For simplicity, we also use the upper (or lower) bound of a node . The upper and lower bounds of a node constitute a bounding box for that subtree.

Now if a point $\mathbf{p}$ is not dominated by $\mathbf{u_r}$, then obviously $\mathbf{p}$ is not dominated by any point in the subtree rooted at $\mathbf{r}$. This means that all comparisons between $\mathbf{p}$ and all points in the subtree rooted at $\mathbf{r}$ can be avoided. Similarly, when searching for points in the subtree rooted at $\mathbf{r}$ that are dominated by $\mathbf{p}$, we can first compare it with $\mathbf{v_r}$, and all comparisons between $\mathbf{p}$ with each node of that subtree can be saved if $\mathbf{v_r}$ is not dominated by $\mathbf{p}$.

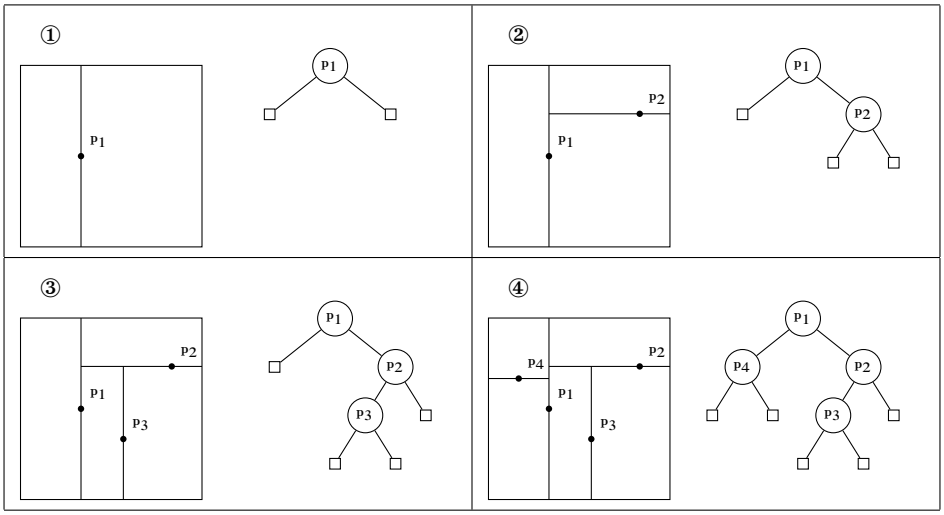

Figure 2: The stepwise construction of a 2-d tree of four points.

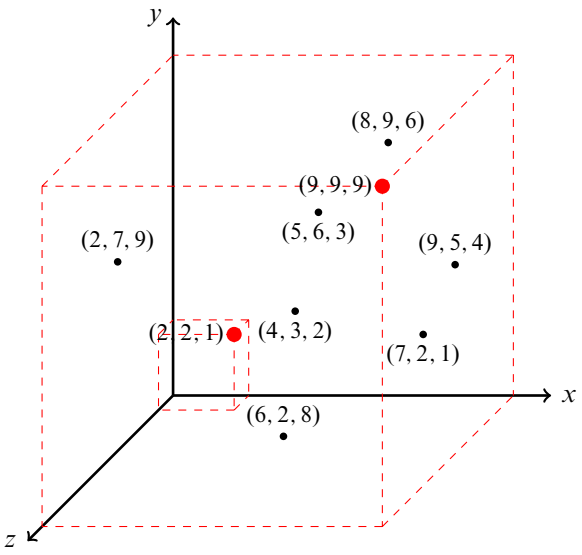

Figure 3: Consider the subtree containing the points $\{(4,3,2),(9,5,4),(7,2,1),(5,6,3),(8,9,6),(2,7,9),(6,2,8)\}$. Then $(9,9,9)$ and $(2,2,1)$ are the upper bound and the lower bound of the subtree, respectively.

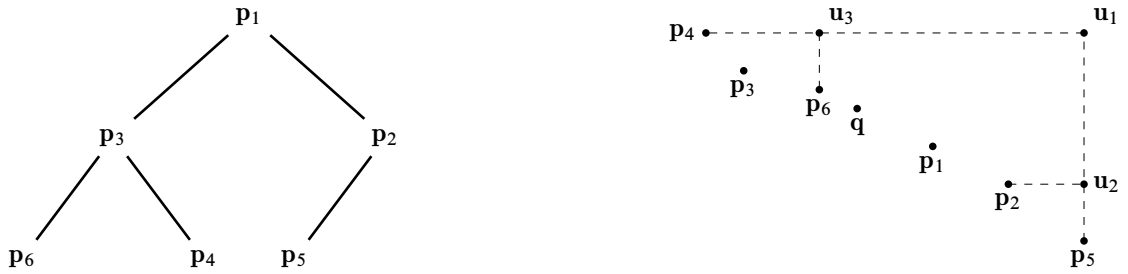

Figure 4: Consider the k-d tree with six points $\mathbf{p}_1$,$\mathbf{p}_2$,..., $\mathbf{p}_6$ and a new point $\mathbf{q}$. The upper bounds of the trees rooted at $\mathbf{p}_1$, $\mathbf{p}_2$ and $\mathbf{p}_3$ are $\mathbf{u}_1$, $\mathbf{u}_2$ and $\mathbf{u}_3$, respectively. To check if $\mathbf{q}$ is dominated by some point in the tree, the comparisons between $\mathbf{q}$ and subtrees rooted at $\mathbf{p}_2$ and $\mathbf{p}_3$ can all be skipped since $\mathbf{q}$ is not dominated by $\mathbf{u}_2$ and $\mathbf{u}_3$.

Note that although additional comparisons and spaces are needed for implementing the bounding boxes in maxima-finding algorithms, the overall performance is generally improved, especially, when dealing with samples with a large number of maxima.

3.2. The proposed algorithm

We give in this subsection our two-phase maxima-finding algorithm using k-d trees and bounding boxes. In this algorithm, we need only the upper bounds of the bounding boxes since in each phase we only detect if the new-coming element is dominated by existing records. An illustrative example is given in Figure 4.

For implementation details, the records are stored, during the first phase, not only in a k-d tree but also in a list to preserve the order of the records.

Algorithm Maxima
//Input: A sequence of points $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$
//Output: a k-d tree rooted at $\mathbf{r}$ consisting of $\mathbf{Max(P)}$
begin
    $\mathbf{r} := \mathbf{p}_1$; $\mathbf{u}_\mathbf{r} := \mathbf{p}_1$
    $\mathbf{q}_1 := \mathbf{p}_1$ // $\mathbf{R} := \{\mathbf{q}_1\}$, the sequence of the records.
    $k := 1$ // k counts the number of records
    for $i := 2$ **to** n **do**
        if (Dominated($\mathbf{r}, \mathbf{p}_i$) $= 0$) **then**
            Insert($\mathbf{r}, 1, \mathbf{p}_i$);
            $k := k + 1$; $\mathbf{q}_k := \mathbf{p}_i$
    // $\mathbf{R} = \{\mathbf{q}_1, \ldots, \mathbf{q}_k\}$ when $i = n$
    release the tree rooted at $\mathbf{r}$
    $\mathbf{r} := \mathbf{q}_k$; $\mathbf{u}_\mathbf{r} := \mathbf{q}_k$;
    for $i := k - 1$ **downto** 1 **do**
        if (Dominated($\mathbf{r}, \mathbf{q}_i$) $= 0$) **then** Insert($\mathbf{r}, 1, \mathbf{q}_i$)
end

Dominated($\mathbf{r}, \mathbf{p}$)
//Input: A node $\mathbf{r}$ in a k-d tree and a point $\mathbf{p}$
//Output: $\begin{cases} 0, & \text{if } \mathbf{p} \text{ is not dominated by any point in the subtree rooted at } \mathbf{r} \\ 1, & \text{otherwise} \end{cases}$
begin
    if ($\mathbf{p} \prec \mathbf{r}$) **then return** 1
    if ($\mathbf{r}$.left $\neq \emptyset$ and $\mathbf{p} \prec \mathbf{u}_{\mathbf{r}.\text{left}}$) **then**
        if (Dominated($\mathbf{r}$.left, $\mathbf{p}$) $= 1$) **then return** 1
    if ($\mathbf{r}$.right $\neq \emptyset$ and $\mathbf{p} \prec \mathbf{u}_{\mathbf{r}.\text{right}}$) **then**
        if (Dominated($\mathbf{r}$.right, $\mathbf{p}$) $= 1$) **then return** 1
    return 0
end

```
Insert(r, ℓ, p)
begin
        u_r := max{u_r, p}     // update the upper bound
        compare the ℓ-th component of p and that of r
        Case 1: p_ℓ ≥ r_ℓ and r.right ≠ ∅
                Insert(r.right, 1 + ℓ mod d, p)
        Case 2: p_ℓ ≥ r_ℓ and r.right = ∅
                r.right := p; u_r.right := p
        Case 3: p_ℓ < r_ℓ and r. left ≠ ∅
                Insert(r.left, 1 + ℓ mod d, p)
        Case 4: p_ℓ < r_ℓ and r. left = ∅
                r.left := p; u_r.left := p
end
```

Note that the upper bound of a subtree is updated after a new point is inserted. In the procedure Dominated, the "filtering role" played by the upper bounds can save many comparisons. In practice, if a point $\mathbf{p}$ is not dominated by $\mathbf{u}_{\mathbf{r}.\text{left}}$ (or $\mathbf{u}_{\mathbf{r}.\text{right}}$), then $\mathbf{p}$ is not dominated by any point in the subtree and the comparisons between $\mathbf{p}$ and the points of the subtree are all skipped.

3.3. Further improvements: sieving and pruning

The algorithm Maxima is not on-line in nature since it requires two passes through the input. In this section, we discuss sieving and periodic pruning techniques, and present an on-line algorithm.

Sieving. The idea is to select an element (or several elements) as a good sieve (or "keeper"), so as to dominate as many as possible in-coming points, thus reducing the total number of comparisons made. This idea was first introduced in [9].

For our algorithm Maxima, many of the points inserted into the k-d tree may have limited power of dominating in-coming points. We can improve further Algorithm Maxima by choosing the input point with the largest L^1-norm (which is the sum of the absolute values of all coordinates) to be the sieve and by incorporating such a procedure as part of algorithm Maxima. The resulting implementation is very efficient, notably for samples with only a small number of maxima.

A simple way to incorporate the maximum L^1-norm point is to replace the line

for $i := 2$ **to** n **do**

in algorithm Maxima by the following

$$\begin{array}{l}
\mathbf{s} := \mathbf{p}_1 \qquad // \ \mathbf{s} = \text{sieve} \\
\textbf{for } i := 2 \textbf{ to } n \textbf{ do} \\
\quad \textbf{if } (\mathbf{p}_i \not\prec \mathbf{s}) \textbf{ then} \\
\qquad \mathbf{s} := \begin{cases} \mathbf{s}, & \textbf{if } \|\mathbf{s}\|_1 \geq \|\mathbf{p}_i\|_1 ; \\ \mathbf{p}_i, & \textbf{if } \|\mathbf{s}\|_1 < \|\mathbf{p}_i\|_1 , \end{cases}
\end{array}$$

where $\|\cdot\|_1$ denotes the L^1-norm. Thus the sieving process is carried out only during the first phase. Other sieves can be considered similarly.

Pruning. In the first phase of Algorithm Maxima, the k-d tree may contain some nodes that are dominated by other nodes in the tree, and will only be removed in the second phase of the algorithm. In particular, if the dominated nodes are close to the root, then more comparisons may be made. It is thus more efficient to carry out an initial pruning of the k-d tree by removing dominated points in the tree after a sufficiently large number of records have been inserted (and still small compared with the total sample size). Such an early pruning idea can be implemented by running the following procedure.

```
Algorithm Prune
// only called once in the first for-loop of Algorithm Maxima
// Assume R = {q_1, ..., q_K}
begin
        release the k-d tree
        r := q_K; u_r := q_K
        for j := K − 1 downto 1
```

if (Dominated($\mathbf{r}, \mathbf{q}_j$) $= 0$) **then** Insert($\mathbf{r}, 1, \mathbf{q}_j$)
end

We can call Prune when, say $i = \lfloor n/\lambda \rfloor$ or $i = \lfloor n^{\delta} \rfloor$, where i is the index in the first **for**-loop of algorithm Maxima. For example, we can take $\lambda = 10$ and $\delta = 2/3$. Which choice is optimal is an interesting issue but depends on the practical implementations. Also one may consider the use of periodic pruning, but since pruning is a costly operation, we chose to apply it only once in our simulations.

An on-line algorithm. On-line maxima-finding algorithms always retain the maxima of the all input points read so far and are often needed in many practical situations. A simple means to convert our algorithm Maxima into an on-line one is to add a procedure to delete the dominated elements in the k-d tree. The deletions can be made immediately after comparison with each in-coming element, which results in restructuring the whole k-d tree and may be very costly if the elements deleted are not near the bottom of a large tree. A simple way to perform the deletion of a node is to re-insert all its descendant nodes one by one, in the order inherited from the original input sequence. However, the procedure can be time-consuming and the resulting tree may be quite imbalanced.

We introduce an on-line implementation by storing the current maxima in an extra list. In each iteration, we look for all points in the k-d tree that are dominated by the in-coming point $\mathbf{p}$, mark them, and delete the corresponding elements from the extra list. The lower bounds of the bounding boxes are useful here. Recall $\mathbf{v_r} = (v_1, \ldots, v_d)$, where v_i is the minimum among all the i-th coordinates of points in the subtree rooted at $\mathbf{r}$. When searching for those points in $\mathbf{M}$ that are dominated by $\mathbf{p}$, we can skip checking the subtree of $\mathbf{r}$ if $\mathbf{v_r}$ is not dominated by $\mathbf{p}$.

The on-line algorithm is given as follows.

Algorithm On-Line-Maxima
//Input: A sequence of points $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$
//Output: $\mathbf{M} :=$ the list containing $\mathbf{Max(P)}$
begin
  $\mathbf{r} := \mathbf{p}_1; \mathbf{u_r} := \mathbf{p}_1; \mathbf{v_r} := \mathbf{p}_1$
  $\mathbf{M} := \{\mathbf{p}_1\}$
  for $i := 2$ **to** n **do**
    if (Dominated($\mathbf{r}, \mathbf{p}_i$) $= 0$) **then**
      Delete($\mathbf{r}, \mathbf{p}_i$)
      Insert($\mathbf{r}, 1, \mathbf{p}_i$)
      $\mathbf{M} := \mathbf{M} \cup \{\mathbf{p}_i\}$
end

Delete($\mathbf{r}, \mathbf{p}$)
//Input: A node $\mathbf{r}$ of a k-d tree and a point $\mathbf{p}$
//Output: a more compact $\mathbf{M}$ (all dominated points are removed)
begin
  if ($\mathbf{r} \prec \mathbf{p}$) **then**
    if ($\mathbf{r}$ is unmarked) **then** // The set of unmarked nodes is exactly $\mathbf{M}$
      delete $\mathbf{r}$ from $\mathbf{M}$
      mark $\mathbf{r}$
  if ($\mathbf{r}$.left $\neq \emptyset$ and $\mathbf{v}_{\mathbf{r}.\text{left}} \prec \mathbf{p}$) **then** Delete($\mathbf{r}$.left, $\mathbf{p}$)
  if ($\mathbf{r}$.right $\neq \emptyset$ and $\mathbf{v}_{\mathbf{r}.\text{right}} \prec \mathbf{p}$) **then** Delete($\mathbf{r}$.right, $\mathbf{p}$)
end

Note that the only difference between the procedure Insert of algorithm On-Line-Maxima and that of algorithm Maxima is that we need to update both the upper bounds and the lower bounds in the procedure Insert($\mathbf{r}, j, \mathbf{p}$) of algorithm On-Line-Maxima.

3.4. Comparative discussions

We ran a few sequential algorithms and tested their performance under several types of random data, each with 1000 iterations; the average values of the results are given in Tables 1 and 2. The points are generated uniformly and independently at random from a given region D, which is either a hypercube or a simplex, the former roughly simulating samples with independent coordinates while the latter those with negatively correlated coordinates.

- list: a sequential algorithm using a linked list (see [9]);
- d-tree: a sequential algorithm using the d-ary tree proposed in [51];
- quadtree: a sequential algorithm using quadtree (see [36, 53, 48]);
- 2-phase: algorithm Maxima;
- +prune: algorithm Maxima with an early pruning for $i = n/10$;
- +sieve: algorithm Maxima with the max-L^1-norm sieve;
- +prune&sieve: algorithm Maxima with pruning for $i = n/10$ and the max-L^1-norm sieve.

Table 1: The average numbers of scalar comparisons per input point when $\boldsymbol{D} = [0,1]^d$, where $d \in \{3,4,6,10\}$.

$d = 3$

n	list	d-tree	quadtree	2-phase	+prune	+sieve	+prune&sieve
10^2	11.40	19.38	13.58	24.72	23.23	19.10	18.82
10^3	11.01	15.01	11.38	24.29	20.81	13.23	12.43
10^4	8.28	12.02	9.41	23.30	17.70	8.44	7.69
10^5	6.36	11.21	8.50	23.31	15.70	5.78	5.30
10^6	5.01	11.40	8.07	23.05	13.75	4.40	4.09
10^7	4.24	11.51	7.91	23.76	12.50	3.73	3.54
10^8	3.88	12.02	7.67	24.11	11.39	3.36	3.25

$d = 4$

n	list	d-tree	quadtree	2-phase	+prune	+sieve	+prune&sieve
10^2	26.96	47.28	30.29	50.22	50.05	44.28	44.78
10^3	37.41	49.48	31.53	53.28	51.07	38.43	37.76
10^4	32.48	40.62	26.80	48.34	43.94	25.73	24.79
10^5	22.36	34.32	22.60	44.30	37.75	16.65	15.78
10^6	14.69	32.36	20.66	42.69	33.00	11.32	10.61
10^7	10.08	32.46	19.47	42.74	29.87	8.40	7.80
10^8	8.40	33.04	19.05	52.22	28.88	6.83	6.08

$d = 6$

n	list	d-tree	quadtree	2-phase	+prune	+sieve	+prune&sieve
10^2	75.44	139.19	74.32	129.85	131.41	126.53	128.20
10^3	228.69	284.69	130.37	193.84	193.27	177.23	177.44
10^4	384.86	343.69	149.75	194.56	194.05	163.10	163.17
10^5	404.74	298.21	131.41	162.01	161.27	116.86	117.40
10^6	310.75	222.30	104.53	133.34	131.66	77.55	78.68
10^7	190.08	166.02	86.63	118.09	112.34	52.13	52.65
10^8	100.77	136.69	74.97	109.50	98.93	36.46	36.36

$d = 10$

n	list	d-tree	quadtree	2-phase	+prune	+sieve	+prune&sieve
10^2	137.56	296.70	132.72	267.90	270.67	269.49	272.22
10^3	1048.73	1496.07	458.30	774.85	777.16	769.01	771.42
10^4	5392.57	4916.40	1190.22	1526.83	1528.66	1498.47	1499.93
10^5	17779.34	11463.01	2201.99	2126.49	2132.18	2062.42	2067.98
10^6	38552.96	18775.90	—	2221.26	2234.51	2121.11	2132.94
10^7	59207.23	20769.36	—	2023.64	1844.68	1931.37	1750.01
10^8	—	19226.26	—	1544.68	1387.00	1429.45	1261.90

Table 1 shows evidently that our two-phase maxima-finding algorithms, whether coupling with sieving and pruning techniques or not, perform very well under random inputs from the d-dimensional hypercubes. They are efficient and

Table 2: The average numbers of scalar comparisons per input point when $\boldsymbol{D}$ is the d-dimensional simplex, where $d = 3, 4$ and 6.

$d = 3$

n	list	d-tree	quadtree	2-phase	+prune	+sieve	+prune&sieve
10^2	40.96	62.81	30.50	57.68	58.00	57.87	58.26
10^3	134.05	112.71	43.98	82.03	80.78	81.34	80.24
10^4	357.25	203.97	55.91	95.20	92.37	93.78	91.23
10^5	858.65	402.18	76.19	105.64	100.79	104.10	99.59
10^6	1957.22	835.16	126.45	117.42	107.53	117.11	107.60
10^7	4334.09	1678.73	161.25	129.18	106.81	130.72	108.50
10^8	9417.80	3543.73	331.25	142.22	116.74	142.73	116.98

$d = 4$

n	list	d-tree	quadtree	2-phase	+prune	+sieve	+prune&sieve
10^2	81.74	123.95	57.61	107.18	108.36	108.37	109.55
10^3	441.09	368.00	117.09	199.20	199.35	199.70	199.87
10^4	1917.26	910.44	208.67	287.21	286.60	287.09	286.49
10^5	7316.79	2230.39	356.48	373.86	371.80	373.60	371.60
10^6	25786.00	5948.65	614.88	474.28	460.27	474.84	461.06
10^7	86609.63	17071.62	1302.10	532.85	487.16	534.66	489.15
10^8	—	53140.49	4696.73	651.13	698.55	646.59	693.70

$d = 6$

n	list	d-tree	quadtree	2-phase	+prune	+sieve	+prune&sieve
10^2	126.37	221.77	91.42	175.93	177.77	177.79	179.63
10^3	1096.21	1175.40	268.67	467.27	468.87	468.96	470.56
10^4	8284.26	5660.90	758.05	993.25	995.64	994.77	997.16
10^5	55200.49	24332.05	2178.38	1849.37	1856.49	1850.86	1858.01
10^6	331776.01	93275.52	6825.69	3153.92	3125.31	3155.81	3127.10
10^7	—	368306.29	8418.26	5090.63	5029.78	5092.54	5031.71
10^8	—	—	—	7996.92	7403.24	7998.93	7405.39

uniformly scalable since the average number of scalar comparisons each point involved is gradually rising, in contrast to the faster increase of other algorithms compared. Note that, according to a result by Devroye [26], we expect that the average number of scalar comparisons each point involves tends eventually to d in each case. This is visible for $d = 3$ but less clear for higher values of d, as the convergence rate is very slow. Also the numbers in each column first increases as n increases and then decreases.

On the other hand, although the asymptotic growth rate of the expected numbers of maxima $\mu_{n,d}$ in such cases are approximately $(\log n)^{d-1}/(d-1)!$ for large n and fixed d, the real values of $\mu_{n,d}$ for moderate d soon become large; for example, when $d = 10$

$$\{\mu_{10^i,10}\}_{i=2,\ldots,8} \approx \{94, 765, 4947, 25113, 103300, 357604, 1076503\}.$$

These values were easily computed by the recurrence (see [1])

$$\mu_{n,d} = \frac{1}{d-1} \sum_{1 \le j < d} H_n^{(d-j)} \mu_{n,j} \qquad (d \ge 2),$$

with $\mu_{n,1} = 1$ for $n \ge 1$, where the $H_n^{(j)} := \sum_{1 \le i \le n} 1/i^j$ are Harmonic numbers. They can also be estimated by the asymptotic approximations given in [1].

The situation is very similar (see Table 2) when the random samples are generated from the d-dimensional simplex, $D = \{\mathbf{x} : x_i \ge 0, \sum_{1 \le i \le d} x_i \le 1\}$ for which the expected numbers of maxima $\nu_{n,d}$ are of order $n^{1-1/d}$ instead of $(\log n)^{d-1}$; see [1]. In such cases, $\nu_{n,d}$ grows even faster than $\mu_{n,d}$. For example, when $d = 6$,

$$\{\nu_{10^i,6}\}_{i=2,\ldots,8} \approx \{95, 863, 7281, 57858, 439110, 3223774, 23121832\}.$$

These values were computed by the exact formula

$$\nu_{n,d} = n \sum_{0 \le j < d} \binom{d-1}{j} (-1)^j \frac{\Gamma(n)\Gamma((j+1)/d)}{\Gamma(n+(j+1)/d)} \qquad (d \ge 2),$$

which follows from

$$\begin{aligned} \nu_{n,d} &= n\mathbb{P}(\mathbf{x}_1 \text{ is a maxima}) \\ &= d!n \int_D \left(1 - \left(1 - \textstyle\sum_{1 \le i \le d} x_i\right)^d\right)^{n-1} \mathrm{d}\mathbf{x} \\ &= dn \int_0^1 \left(1 - (1-y)^d\right)^{n-1} y^{d-1} \mathrm{d}y, \end{aligned}$$

by straightforward calculations, where Γ denotes the Gamma function. For similar details, see [1].

Unlike hypercubes where sieving is seen to be very helpful, the gain of using sieving for random samples whose coordinates are roughly negatively correlated is marginal since there is no "omnipotently powerful" point; see [6, 34].

A feature of the quadtree algorithm is that by its large amount of branching factors ($2^d - 2$), the position of a point in the tree is quickly identified, often after a few comparisons, and the bounding boxes are thus not helpful here. We also implemented our 2-phase algorithm on quadtrees and d-trees, the improvement over the original algorithms is much more significant in d-trees than in quadtrees. In contrast, since k-d trees are binary, the use of the bounding boxes plays a crucial role in accelerating the performance of the algorithm.

While the data collected in these two tables do not reflect exactly the running time of each program, our algorithms also perform much better than the others that are tested and compared.

Simulations also suggested that our on-line algorithm is also reasonably efficient when compared with other algorithms.

4. Average-case analysis of algorithm Maxima

We derive in this section a few analytic results in connection with the performance of the algorithms we proposed in this paper. In general, probabilistic analysis of sequential algorithms for finding the maxima of random samples is very difficult due to the dynamic nature of the algorithms; see [26, 18, 35] and the references therein.

4.1. How many non-dominated records are there?

The performance of Maxima depends heavily on the number of records, which in turn is closely related to the number of maxima.

Theorem 1. *Let R_n denote the number of non-dominated records in a sequence $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ of independent and uniformly distributed points from some region D in $\mathbb{R}^d$. Let M_n denote the number of the maxima of $\mathbf{P}$. Then*

$$\mathbb{E}(R_n) = \sum_{i=1}^{n} \frac{\mathbb{E}(M_i)}{i}, \tag{1}$$

where $\mathbb{E}(\cdot)$ denotes the expectation.

Proof. By assumption,

$$\mathbb{P}\left(\mathbf{p}_1 \in \mathbf{Max}(\mathbf{P})\right) = \cdots = \mathbb{P}\left(\mathbf{p}_n \in \mathbf{Max}(\mathbf{P})\right).$$

Thus

$$\mathbb{E}(M_n) = \sum_{i=1}^{n} \mathbb{P}\left(\mathbf{p}_i \in \mathbf{Max}(\mathbf{P})\right) = n\mathbb{P}\left(\mathbf{p}_n \in \mathbf{Max}(\mathbf{P})\right).$$

Then we have

$$\begin{aligned}\mathbb{E}(R_n) &= \sum_{i=1}^{n} \mathbb{P}\left(\mathbf{p}_i \in \mathbf{Max}(\{\mathbf{p}_1, \ldots, \mathbf{p}_i\})\right) \\ &= \sum_{i=1}^{n} \frac{\mathbb{E}(M_i)}{i}.\end{aligned}$$

□

Since $\mathbb{E}(M_n)$ is usually of order n^α or $(\log n)^\beta$ for some $\alpha, \beta \geq 0$ (see [1, 2, 24]), if we assume that $\mathbb{E}(M_n) \sim cn^\alpha(\log n)^\beta$, where $c, \beta > 0$ and $\alpha \in [0, 1]$, then, by (1),

$$\mathbb{E}(R_n) \sim \begin{cases} \dfrac{c}{\alpha} n^\alpha (\log n)^\beta \sim \dfrac{\mathbb{E}(M_n)}{\alpha}, & \text{if } 0 < \alpha \leq 1; \\ \dfrac{c}{\beta+1}(\log n)^{\beta+1} \sim \dfrac{\mathbb{E}(M_n)}{\beta+1}\log n, & \text{if } \alpha = 0, \end{cases}$$

where $a_n \sim b_n$ means that $a_n/b_n \to 1$ as $n \to \infty$. We see that if $\alpha > 0$ then the expected number of records is of the same order as the number of maxima. An algorithmic consequence is that the non-maximal records to be deleted in the second phase is on average of the same order as the expected number of maxima.

In the special case when the region D is the d-dimensional hypercube $[0, 1]^d$, then it is also easily seen that the number of non-dominated records in random samples from $[0, 1]^d$ is identically distributed as the number of maxima in random samples from $[0, 1]^{d+1}$; see [31].

Whichever the case, we always have the bounds

$$\mathbb{E}(R_n) \leq \mathbb{E}(M_n) \sum_{i=1}^{n} \frac{1}{i} = O(\mathbb{E}(M_n)\log n).$$

This partly explains why our two-phase algorithm runs reasonably efficient. Also we see that the expected additional memory used for the k-d tree (and possibly the array) is at most a $\log n$ factor more than the expected number of maxima.

4.2. Expected cost of the sieve algorithm

Assume that $\mathbf{p}_1, \ldots, \mathbf{p}_n$ are sampled independently and uniformly at random from $[0, 1]^d$. Let $\mathbf{s}_n$ be the point with the maximum L^1-norm. Let $\mathbf{1} = (\underbrace{1, \ldots, 1}_{d})$.

Lemma 2. *For any $c > 0$,*

$$\mathbb{P}\left(\|\mathbf{s}_n - \mathbf{1}\|_1 < (cd!)^{1/d} n^{-1/d} (\log n)^{1/d}\right) \geq 1 - n^{-c},$$

for sufficiently large n.

Proof. For $0 < \varepsilon < 1$

$$\begin{aligned}\mathbb{P}\left(\|\mathbf{s}_n - \mathbf{1}\|_1 < \varepsilon\right) &= 1 - \mathbb{P}\left(\|\mathbf{p}_i\|_1 \le d - \varepsilon, 1 \le i \le n\right) \\ &= 1 - \left(1 - \frac{\varepsilon^d}{d!}\right)^n \\ &\ge 1 - e^{-\varepsilon^d n/d!}.\end{aligned}$$

Taking $\varepsilon = (cd!)^{1/d} n^{-1/d} (\log n)^{1/d}$, we see that the last expression is equal to $1 - n^{-c}$. Note that $\varepsilon < 1$ if n is large enough. Indeed, $n/\log n > cd!$ suffices. □

Theorem 2. *Assume the n points $\{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ are sampled independently and uniformly at random from $[0,1]^d$. Then the expected number of scalar comparisons used by our sieve algorithm satisfies $dn + O(n^{1-1/d}(\log n)^{d+1/d})$.*

Proof. The number of scalar comparisons used for the sieve is at most dn. We claim that the expected number of the extra comparisons is only $O(n^{1-1/d}(\log n)^{d+1/d})$. Let $a_i = (2d!)^{1/d} i^{-1/d} (\log i)^{1/d}$. For i large enough

$$\mathbb{P}\left(\|\mathbf{s}_i - \mathbf{1}\|_1 < a_i\right) \ge 1 - i^{-2},$$

by Lemma 2. If $\mathbf{p}_{i+1} \in [0, 1 - a_i]^d$ and $\|\mathbf{s}_i - \mathbf{1}\|_1 < a_i$ both hold, then $\mathbf{p}_{i+1} \prec \mathbf{s}_i$, that is, $\mathbf{p}_{i+1}$ is filtered out. Thus, additional comparisons are required only when either $\mathbf{p}_{i+1} \notin [0, 1 - a_i]^d$ or $\|\mathbf{s}_i - \mathbf{1}\|_1 \ge a_i$. If $\mathbf{p}_{i+1} \notin [0, 1 - a_i]^d$, then the additional comparisons used is bounded above by $O(R_i)$; if $\|\mathbf{s}_i - \mathbf{1}\|_1 \ge a_i$, then the extra comparisons are at most $O(i)$. Note that $\mathbf{p}_{i+1}$ and R_i are independent. Thus, the expected number of the extra comparisons required in the for-loop of $\mathbf{p}_{i+1}$ is less than

$$\begin{aligned}&\mathbb{P}\left(\mathbf{p}_{i+1} \notin [0, 1 - a_i]^d\right) O(\mathbb{E}(R_i)) + \mathbb{P}\left(\|\mathbf{s}_i - \mathbf{1}\|_1 \ge a_i\right) O(i) \\ &= O(i^{-1/d}(\log i)^{d+1/d}) + O(i^{-1})\end{aligned}$$

since $\mathbb{E}(R_i) = O\left((\log i)^d\right)$. Summing over all $i = 2, \ldots, n$, we obtain the required bound. □

4.3. Expected performance of Maxima *when all points are maxima*

To further clarify the "scalability" of Maxima, we consider in this subsection the expected cost used by Maxima under the extreme situation when the d-dimensional input points are sampled independently and uniformly from the $(d-1)$-dimensional simplex $D = \{\mathbf{x} : x_i \ge 0, \sum_{1 \le i < d} x_i = 1\}$. Note that in the skyline context, an anti-correlated sample is often discussed, which is the $(d-1)$-dimensional simplex with a specified error range. In that case, most but not necessarily all points are maxima. Since no deletion is involved in our algorithm Maxima, the difference between random samples from the $(d-1)$-dimensional simplex and the anti-correlated sample is minor.

When D is the $(d-1)$-dimensional simplex, all points are maxima, and the time complexity of most algorithms such as the list algorithm (see [9]) is of order $O(M_n^2) = O(n^2)$. We show that the expected time complexity of Maxima is $O(n \log n)$ when $d = 2$.

Theorem 3. *Assume that the d-dimensional points $\{\mathbf{p}_1, \cdots, \mathbf{p}_n\}$ are independently and uniformly distributed in the $(d-1)$-dimensional simplex. The expected number of comparisons needed by algorithm* Maxima *for random samples is bounded above by $O(n \log n)$ when $d = 2$.*

We leave open the probabilistic analysis for the case when $d \ge 3$.

Proof. Since all points in the sample are maxima, the expected number of comparisons used in the first phase and that in the second phase are the same. Thus, we focus on the first phase.

Assume that $\{\mathbf{p}_1, \ldots, \mathbf{p}_m\}$ have been stored in a k-d tree. We consider the number of comparisons that $\mathbf{p}_{m+1}$ may involve inside the two procedures of the for-loop: Insert and Dominated. The expected number of comparisons used in Insert is of order

$$O(\text{the expected depth of the } k\text{-d tree}) = O(\log m),$$

since the k-d tree is essentially a binary search tree (see [7]).

We now estimate the expected number of comparisons used in Dominated. Since at most three vector comparisons are involved in the procedure Dominated, we analyze the number of times T_m the procedure Dominated is called. To complete the proof, we show that $\mathbb{E}(T_m) = O(\log m)$.

Obviously, Dominated$(\mathbf{r}, \mathbf{p}_{m+1})$ is called when $\mathbf{p}_{m+1} \prec \mathbf{u}_\mathbf{r}$. Thus, the number of times Dominated is called is equal to the number of nodes $\mathbf{r}$ such that $\mathbf{p}_{m+1} \prec \mathbf{u}_\mathbf{r}$. Let $D_\mathbf{r} \subset D$ be the region that $\mathbf{u}_\mathbf{r}$ covers. Then the probability of the event $\mathbf{p}_{m+1} \prec \mathbf{u}_\mathbf{r}$ conditioning on the k-d tree built from $\{\mathbf{p}_1, \ldots, \mathbf{p}_m\}$ equals $|D_\mathbf{r}| / |D|$. Thus

$$\mathbb{E}(T_m) = \frac{1}{|D|}\mathbb{E}\left(\sum_\mathbf{r} |D_\mathbf{r}|\right),$$

where the summation runs over all nodes and the expectation is taken with respect to the k-d tree for $\{\mathbf{p}_1, \ldots, \mathbf{p}_m\}$. To estimate $\sum_\mathbf{r} |D_\mathbf{r}|$, we consider $A_\mathbf{r} \subset D$, the possible ranges induced by the nodes of the subtrees rooted at $\mathbf{r}$. The precise definition is as follows. Define $A_\mathbf{r} := D$ when $\mathbf{r}$ is the root. If $\mathbf{r}$.**left** ($\mathbf{r}$.**right**) represents the point at the root node of the left (right) subtree of $\mathbf{r}$, respectively, then

$$\begin{cases} A_{\mathbf{r}.\mathbf{left}} := A_\mathbf{r} \cap [0,1]^{j-1} \times [0, x_j) \times [0,1]^{d-j}, \\ A_{\mathbf{r}.\mathbf{right}} := A_\mathbf{r} \cap [0,1]^{j-1} \times [x_j, 1] \times [0,1]^{d-j}, \end{cases} \qquad (j = 1, \ldots, d),$$

where $d = 2$, the j-th coordinate is the discriminator of node $\mathbf{r}$ and $\mathbf{r} = (x_1, x_2, \ldots, x_d)$.

Since the union of $A_\mathbf{r}$ in the same level of the k-d tree is at most D and $D_\mathbf{r} \subset A_\mathbf{r}$ (see Figure 5), we have

$$\mathbb{E}(T_m) \le \frac{1}{|D|}\mathbb{E}\left(\sum_\mathbf{r} |A_\mathbf{r}|\right) \le \text{ the expected depth of the } k\text{-}d \text{ tree} = O(\log m).$$

□

Note that $A_\mathbf{r}$ is determined by $\mathbf{r}$ and its ancestors; in contrast, $D_\mathbf{r}$ is determined by $\mathbf{r}$ and its offsprings.

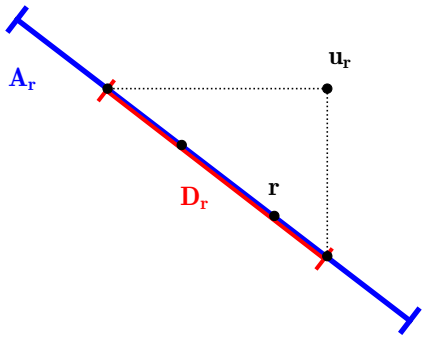

Figure 5: A possible configuration of $A_\mathbf{r}$ and $D_\mathbf{r}$ for $d = 2$.

For $d \ge 3$, the expected time-complexity remains open. However, simulations suggest that for fixed d the expected time be of order $O(n(\log n)^c)$ for some $c > 0$; see Figure 6. On the other hand, for fixed n and increasing d, the expected number of comparisons appears to be of order $O(dn \log n)$.

One way of seeing why our algorithm suffers less from the so-called "curse of dimensionality" than other algorithms in such extreme cases is as follows. As is obvious from the proof of Theorem 3, the time complexity is proportional to the order of $|D_\mathbf{r}|/|A_\mathbf{r}|$. The more slender $A_\mathbf{r}$ is, the larger $|D_\mathbf{r}|/|A_\mathbf{r}|$ becomes. All four possible patterns of $A_\mathbf{r}$ for $d = 3$ are shown in Figure 7. The slenderness does not seem to worsen rapidly as there is some sort of counter-balancing process at play; see Figure 7.

5. Applications

In this section, we apply algorithm Maxima to find successively the maximal layers and to search for the longest common subsequence of multiple sequences, respectively. In both cases, our algorithms generally achieve better performance.

5.1. Maximal layers

The problem is to split the input set of points $\mathbf{P}$ into layers according to maxima. Let $\mathbf{L}_k$ denote the k-th maximal layer of $\mathbf{P}$. Then $\mathbf{L}_1 = \mathbf{Max}(\mathbf{P})$ and

$$\mathbf{L}_k := \mathbf{Max}\left(\mathbf{P} \setminus \bigcup_{1 \le i < k} \mathbf{L}_i\right), \quad \text{for } k \ge 2.$$

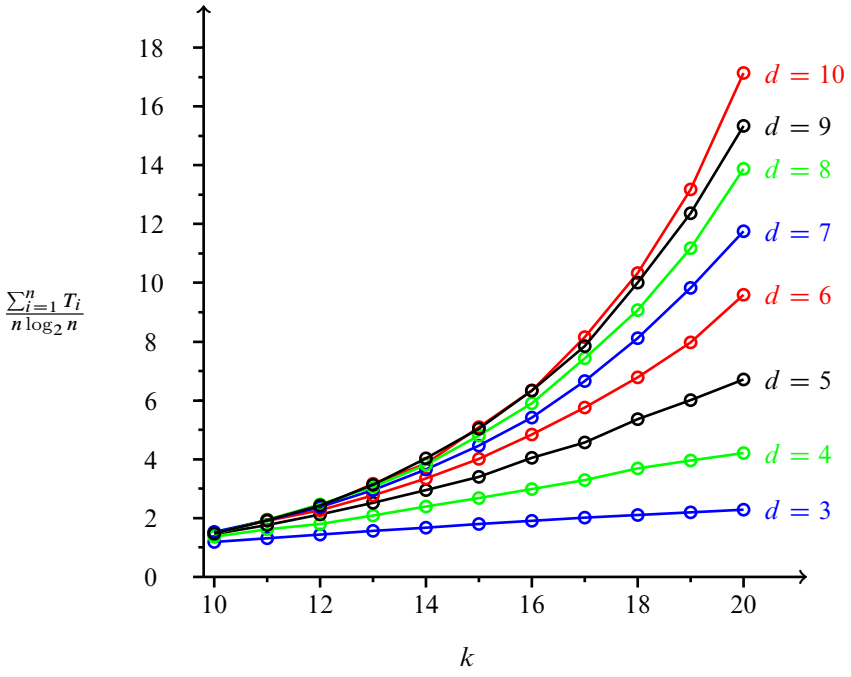

Figure 6: Simulation results of the total number of times the procedure Dominated is called for in the first phase for $d = 3, 4, 5, \ldots, 10$ and $n = 2^k$ for k from 10 to 20. Here we plot $\frac{\sum_{i=1}^{n} T_i}{n \log_2 n}$ against $k = \log_2 n$.

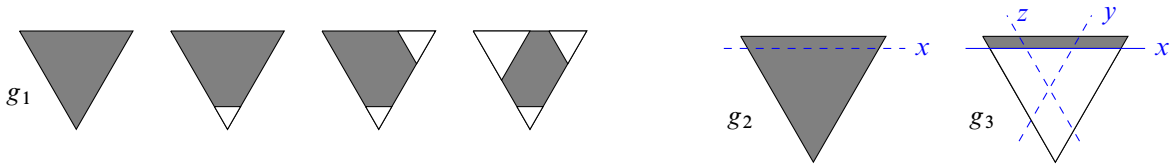

Figure 7: Here $d = 3$. All four possible configurations of A_r are shown on the left (the four smaller triangles). We can see how A_r tends to keep from getting too slender by the splittings resulted from x-, y- and z-coordinates, respectively, (as discriminators in the corresponding k-d tree). Take the leftmost region (graph g_1) for instance. If A_r is split less evenly by x-axis (graph g_2), then later splittings along y-axis or along z-axis tend to counterbalance the effect caused by x-axis (graph g_3).

Maximal layers have been widely applied in multi-objective optimization problems, and algorithms with $O(n \log n)$-time complexity were known for finding the two- and three-dimensional maximal layers; see [12, 15].

By identifying the first few layers of maxima to preserve the so-called elitism, Srinivas and Deb [52] proposed a multi-objective evolutionary algorithm, called non-dominated sorting genetic algorithm (NSGA). This algorithm was later improved and called NSGA-II [22], which reduces the worst-case time complexity from $O(dn^3)$ to $O(dn^2)$ and soon became extremely popular. Omitting the details of the corresponding genetic algorithms, the NSGA-II algorithm [22] for finding the maximal layers can be extracted and summarized in the following two steps.

Step 1: For each point $\mathbf{p}_i$, compute the rank n_i and the set S_i where $n_i := \left|\{\mathbf{p}_j : \mathbf{p}_i \prec \mathbf{p}_j\}\right|$ and $\mathbf{S}_i := \{\mathbf{p}_j : \mathbf{p}_j \prec \mathbf{p}_i\}$, by comparing all pair of points.

Step 2: Then the maximal layers can be determined by n_i and $\mathbf{S}_i$ as follows. The first layer $\mathbf{L}_1$ contains the points with zero rank. For $k \geq 2$, remove $\mathbf{L}_{k-1}$ and update the rank n_i by using $\mathbf{S}_i$. Then, $\mathbf{L}_k$ is the set of the points with zero rank among all points that remain.

The running time is obviously $O(dn^2)$ since all pairs of points are compared.

A straightforward way to compute the maximal layers is to find successively the maxima after the removal of each layer.

Algorithm Peeling

//**Input**: A sequence of points $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$

//Output: Maximal layers $\mathbf{L}_1, \mathbf{L}_2, \ldots$
begin
    $k := 0$
    while $(|\mathbf{P}| > 0)$
        $k := k + 1$
        $\mathbf{L}_k :=$ Find-Maxima $(\mathbf{P})$
        $\mathbf{P} := \mathbf{P} - \mathbf{L}_k$
end

Algorithm Peeling is simple and efficient in average situations, even though the worst-case complexity is $O(n^3)$. Any maxima-finding algorithm can be used for the procedure Find-Maxima($\mathbf{P}$). To study the average behavior of algorithm Peeling, we compare two procedures for Find-Maxima: algorithm Maxima and algorithm Naive. Algorithm Naive finds maxima using pairwise comparisons.

Algorithm Naive
//Input: A set of points $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$
//Output: $\mathbf{M} = \mathbf{Max}(\mathbf{P})$
begin
    $\mathbf{M} := \{\}$
    for $i := 1$ **to** n **do**
        for $j := 1$ **to** n **do**
            if $(i \neq j$ and $\mathbf{p}_i \prec \mathbf{p}_j)$ **then** break
            if $(j = n)$ **then** insert $\mathbf{p}_i$ into $\mathbf{M}$
end

Theorem 4. *If the n points $\{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ are independently and uniformly sampled from any given region in $\mathbb{R}^d$, then the expected running time of algorithm Peeling using algorithm Naive is $O\left(n^2 \log(K+1)\right)$, conditioned on the number of maximal layers K.*

Proof. Consider the event that the total number of layers is K and the number of points in the i-th layer $\mathbf{L}_i$ is ℓ_i for $1 \leq i \leq K$.

We now fix k. At the moment of computing $\mathbf{L}_k$, the total number of remaining points is equal to $N_k := \sum_{i=k}^{K} \ell_i$. If a point $\mathbf{p}$ is in the i-th layer for $i \geq k$, then the number of points that dominate $\mathbf{p}$ is at least $i - k$. Thus, the expected number of comparisons that $\mathbf{p}$ involves in the loop for computing the k-th layer maxima is upper bounded by

$$\leq \begin{cases} N_k, & \text{if } i = k, \\ N_k/(i-k), & \text{if } i > k, \end{cases}$$

since the remaining points preserve the randomness. Summing over all $\mathbf{p}$ and k, we obtain the upper bound for the expected number of comparisons used

$$\begin{aligned} \sum_{k=1}^{K} \ell_k N_k + \sum_{k=1}^{K} \sum_{i=k+1}^{K} \frac{\ell_i N_k}{i-k} &\leq n^2 + n \sum_{i=2}^{K} \sum_{k=1}^{i-1} \frac{\ell_i}{i-k} \\ &\leq n^2 + n^2 \left(1 + \log K\right). \end{aligned}$$

This completes the proof. □

Note that the proof also extends to more general non-uniform distributions.

We compare the numbers of scalar comparisons used by the following three algorithms for finding the maximal layers: Deb et al.'s algorithm [22], algorithm Peeling using Maxima, and algorithm Peeling using Naive. The simulation results are shown in Figure 8. Note that we reverse the order of the remainder after a layer is found to make the algorithm more efficient. It is clear that algorithm Peeling using Maxima outperforms generally the other two, especially for higher dimensional samples in large data sets.

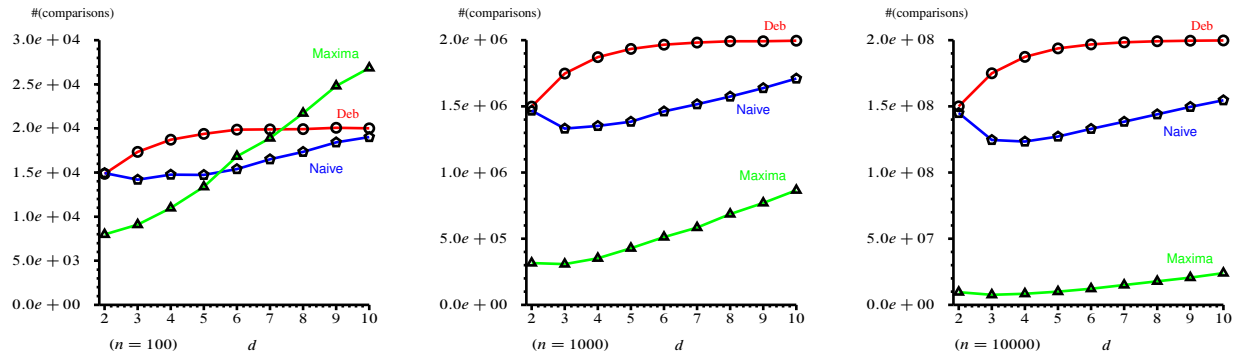

Figure 8: Simulation of Deb's algorithm, and the peeling method with algorithm Naive and algorithm Maxima, respectively. We compare the number of scalar comparisons used in the algorithms. Here the sample size $n = 10^2, 10^3, 10^4$ and the points are generated uniformly from $[0,1]^d$ for $d = 2, 3, \ldots, 10$.

5.2. The multiple longest common subsequence problem

Given two or more strings (or sequences), the longest common subsequence (LCS for short) problem is to determine the longest common subsequence obtained by removing zero or more symbols from each string. For example, if two strings $\mathrm{s}_1 = aabbc$ and $\mathrm{s}_2 = abac$, then the LCS of s_1 and s_2, denoted by $\mathrm{LCS}(\mathrm{s}_1, \mathrm{s}_2)$, is abc. The LCS of sequences is widely used in computational biology, notably in DNA and protein sequence analysis.

Various algorithms for computing an LCS between two strings were derived in the literature, but much fewer algorithms are devoted to the LCS of more than two strings. Hakata and Imai [37] proposed a method for solving efficiently the multiple LCS problem. The method is essentially based on minima-finding.

Let $\mathrm{s}_1 = a_1 a_2 \cdots a_n$ and $\mathrm{s}_2 = b_1 b_2 \cdots b_m$ be two strings. We say that (i, j) is a *match* if $a_i = b_j$. Consider two matches (i_1, j_1) and (i_2, j_2). If $i_1 < i_2$ and $j_1 < j_2$, then the length of $\mathrm{LCS}(a_1 \cdots a_{i_1}, b_1 \cdots b_{j_1})$ is less than $\mathrm{LCS}(a_1 \cdots a_{i_2}, b_1 \cdots b_{j_2})$; that is,

$$|\mathrm{LCS}(a_1 \cdots a_{i_1}, b_1 \cdots b_{j_1})| < |\mathrm{LCS}(a_1 \cdots a_{i_2}, b_1 \cdots b_{j_2})|.$$

Thus, finding the LCS can be roughly regarded as finding the maximal layers of all possible matches. However, the number of matches is usually too large. The approach proposed in [37] is to find the layers one after another as follows. Assume we have found the k-th layer, C_k, then the $(k+1)$-st layer is the minima of all successors of C_k, where a match (i_2, j_2) is called a *successor* of another match (i_1, j_1) if $i_1 < i_2$ and $j_1 < j_2$ and there is no match between them. The minima-finding algorithm proposed in [37] is an improvement over algorithm Naive. The algorithm runs as follows.

Algorithm Hakata-Imai
  //Input: A set of points $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$
  //Output: $\mathbf{M}$ contains minima of $\mathbf{P}$
  begin
    $\mathbf{M} := \{\}$
    for $i := 1$ **to** n **do**
      if $\mathbf{p}_i$ is unmarked **then**
        for $j := 1$ **to** n **do**
          if $\mathbf{p}_j$ is unmarked **then**
            if $(\mathbf{p}_i \prec \mathbf{p}_j)$ **then** mark $\mathbf{p}_j$
            if $(\mathbf{p}_j \prec \mathbf{p}_i)$ **then** mark $\mathbf{p}_i$
        if $\mathbf{p}_i$ is unmarked **then** insert $\mathbf{p}_i$ into $\mathbf{M}$
  end

This algorithm is similar to the list algorithm if we consider node-marking as a substitute of node-deletion.

We compare the performance of Hakata-Imai and Maxima for the number of strings $3, 5, 7$ and alphabet sizes $4, 20$. See the experimental results in Figure 9 where the improvement achieved by our algorithm is visible.

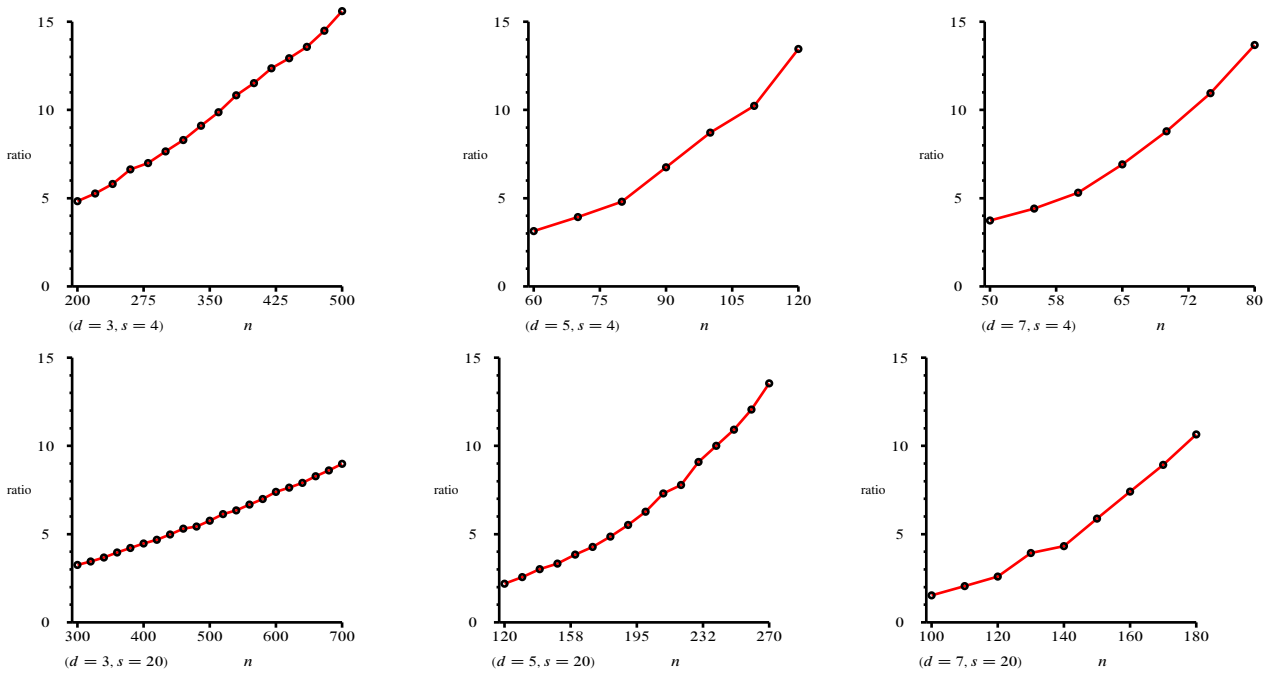

Figure 9: A plot of the ratio between the running time of Hakata-Imai [37] and that of Maxima when the numbers of strings $d = 3, 5, 7$, the alphabet size $s = 4, 20$, and n is the length of the strings. All strings are uniformly generated at random.

References

[1] Z.-D. Bai, L. Devroye, H.-K. Hwang and T.-H. Tsai, Maxima in hypercubes, *Random Structures and Algorithms*, **27** (2005), 290–309.

[2] Z.-D. Bai, H.-K. Hwang, W.-Q. Liang and T.-H. Tsai, Limit theorems for the number of maxima in random samples from planar regions, *Electronic Journal of Probability*, **6** (2001), paper no. 3, 41 pages.

[3] Z.-D. Bai, S. Lee and M. D. Penrose, Rooted edges of a minimal directed spanning tree on random points. *Advances in Applied Probability*, **38** (2006), 1–30.

[4] 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*, **12** (1999), 1119–1178.

[5] I. Bartolini, P. Ciaccia and M. Patella, Efficient sort-based skyline evaluation, *ACM Transactions on Database Systems*, **33** (2008), Article 31, 49 pages.

[6] Yu. Baryshnikov, On expected number of maximal points in polytopes, 2007 Conference on Analysis of Algorithms, *DMTCS Proc.* AH, 2007, 227–236.

[7] J. L. Bentley, Multidimensional binary search trees used for associative searching, *Communications of the ACM*, **18** (1975), 509–517.

[8] J. L. Bentley, Multidimensional divide-and-conquer, *Communications of the ACM*, **23** (1980), 214–229.

[9] J. L. Bentley, K. L. Clarkson and D. B. Levine, Fast linear expected-time algorithms for computing maxima and convex hulls, *Algorithmica*, **9** (1993), 168–183.

[10] A. G. Bhatt and R. Roy, On a random directed spanning tree. *Advances in Applied Probability*, **36** (2004), 19–42.

[11] G. Biau and L. Devroye, On the layered nearest neighbour estimate, the bagged nearest neighbour estimate and the random forest method in regression and classification, preprint, (2008).

[12] H. Blunck and J. Vahrenhold, In-place algorithms for computing (layers of) maxima, *Algorithmica*, to appear.

[13] B. Bollobás and P. Winkler, The longest chain among random points in Euclidean space, *Proceedings of the American Mathematical Society*, **103** (1988), 347–353.

[14] S. Börzsönyi, D. Kossmann and K. Stocker. The skyline operator, *Proceedings 17th International Conference on Data Engineering*, pp. 421–430, 2001.

[15] A. L. Buchsbaum and M. T. Goodrich, Three-dimensional layers of maxima, *Algorithmica*, **39** (2004), 275–286.

[16] W.-M. Chen, H.-K. Hwang and T.-H. Tsai, Efficient maxima-finding algorithms for random planar samples, *Discrete Mathematics and Theoretical Computer Science*, **6** (2003), 107–122.

[17] W.-M. Chen and W.-T. Lee, An efficient evolutionary algorithm for multiobjective optimization problems, in *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, 2007, pp. 30–33.

[18] K. L. Clarkson, More output-sensitive geometric algorithms (extended abstract), in *IEEE 35th Annual Symposium on Foundations of Computer Science*, pp. 695–702, Santa Fe, New Mexico, 1994.

[19] C. A. Coello Coello, Evolutionary multi-objective optimalization: a historical view the field, *IEEE Computational Intelligence Magazine*, February 2006, pp. 28–36.

[20] C. A. Coello Coello, D. A. Van Veldhuizen and G. B. Lamont, *Evolutionary Algorithms for Solving Multi-objective Problems*, 2nd Ed., Springer, New York, 2007.

[21] K. Deb, *Multi-Objective Optimization using Evolutionary Algorithms*, John Wiley & Sons, 2001.

[22] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, *IEEE Transactions on Evolutionary Computation*, **6** (2002), 182–197.

[23] L. Devroye, Moment inequalities for random variables in computational geometry, *Computing*, **30** (1983), 111–119.

[24] L. Devroye, Records, the maximal layer, and uniform distributions in monotone sets, *Computers and Mathematics with Applications*, **25** (1993), 19–31.

[25] L. Devroye, On random Cartesian trees, *Random Structures and Algorithms*, **5** (1994), 305–327.

[26] L. Devroye, A note on the expected time for finding maxima by list algorithms, *Algorithmica*, **23** (1999), 97–108.

[27] M. Ehrgott, *Multicriteria Optimization*, Berlin, Springer, 2000.

[28] J. Fieldsend, R.M. Everson and S. Singh, Using unconstrained elite archives for multi-objective optimisation, *IEEE Transactions on Evolutionary Computation*, **7** (2003), 305–323.

[29] P. Flajolet and M. Golin, Exact asymptotics of divide-and-conquer recurrences, *Lecture Notes in Computer Science*, **700**, pp. 137–149, Springer, Berlin, 1993.

[30] H. N. Gabow, J. L. Bentley and R. E. Tarjan, Scaling and related techniques for geometry problems, *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, pp. 135–143, 1984.

[31] A. V. Gnedin, The chain records, *Electronic Journal of Probability*, **12** (2007), 767–786 (electronic).

[32] P. Godfrey, R. Shipley and J. Gryz, Algorithms and analysis for maximal vector computation, *The VLDB Journal*, **16** (2007), 5–28.

[33] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[34] M. J. Golin, Maxima in convex regions, in *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms* (Austin, TX, 1993), 352–360, ACM, New York, 1993.

[35] M. J. Golin, A provably fast linear-expected-time maxima-finding algorithm, *Algorithmica*, **11** (1994), 501–524.

[36] W. Habenicht, Quad trees, a datastructure for discrete vector optimization problems, in *Essays and Surveys on Multiple Criteria Decision Making: Proceedings on the Fifth International Conference on Multiple Criteria Decision Making, 1982*, pp. 136–145, Springer (1983).

[37] K. Hakata and H. Imai, Algorithms for the longest common subsequence problem for multiple strings based on geometric maxima, *Optimization Methods and Software*, **10** (1998), 233–260.

[38] H.-K. Hwang and T.-H. Tsai, Multivariate records based on dominance, manuscript submitted for publication (2010).

[39] M. Jensen, Reducing the run-time complexity of multiobjective EAs: The NSGA-II and other algorithms, *IEEE Transactions on Evolutionary Computation*, **7** (2003), 503–515.

[40] A. Kaldewaij, Some algorithms based on the dual of Dilworth's theorem, *Science of Computer Programming*, **9** (1987), 85–89.

[41] D. G. Kirkpatrick abd R. Seidel, Output-size sensitive algorithms for finding maximal vectors, in *Proceedings of the first Annual Symposium on Computational Geometry*, 1985, 89–96.

[42] J. D. Knowles and D.W. Corne, Approximating the nondominated front using the Pareto archived evolution strategy, *Evolutionary Computation*, **8** (2000), 149–172.

[43] D. E. Knuth, *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, Third Edition, Addison-Wesley, Reading, Massachusetts, 1997.

[44] D. Kossmann, F. Ramsak and S. Rost, Shooting stars in the sky: An online algorithm for skyline queries, *Proceedings of the 28th International Conference on Very Large Data Bases*, pp. 275–286, 2002.

[45] H. T. Kung, F. Luccio and F. P. Preparata, On finding the maxima of a set of vectors, *Journal of the ACM*, **22** (1975), 469–476.

[46] H. Li and Q. Zhang, Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II, *IEEE Transactions on Evolutionary Computation*, **13** (2009), 284–302.

[47] X. Lian and L. Chen, Reverse skyline search in uncertain databases, *ACM Transactions on Database Systems* **35** (2010), 1–49.

[48] S. Mostaghim, J. Teich and A. Tyagi, Comparison of data structures for storing Pareto sets in MOEAs, *Proceedings World Congress on Computational Intelligence*, IEEE Press, pp. 843–849, 2002.

[49] D. Papadias, Y. Tao, G. Fu and B. Seeger, Progressive skyline computation in database systems, *ACM Transactions on Database Systems*, **30** (2005), 41–82.

[50] F. P. Preparata and M. I. Shamos, *Computational Geometry. An Introduction.* Springer-Verlag, New York, 1985.

[51] O. Schütze, A new data structure for the nondominance problem in multiobjective optimization, in *Evolutionary Multicriterion Optimization*, Edited by C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, Lecture Notes in Computer Science, Vol. 2632, Springer, Berlin, Germany, pp. 509–518, 2003.

[52] N. Srinivas and K. Deb, Multiobjective function optimization using nondominated sorting genetic algorithms, *Evolutionary Computation*, **2** (1995), 221–248.

[53] M. Sun and R. E. Steuer, Quad-trees and linear lists for identifying nondominated criterion vectors, *INFORMS Journal on Computing*, **8** (1996), 367–375.

[54] K. Tan, P. Eng and B. Ooi, Efficient progressive skyline computation, *Proceedings of the 27th International Conference on Very Large Data Bases*, pp. 301–310, 2001.

[55] E. Zitzler, K. Deb, and L. Thiele, Comparison of multiobjective evolutionary algorithms: Empirical results, *Evolutionary Computation*, **8** (2000), 173 – 195.

[56] E. Zitzler and L. Thiele, Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach, *IEEE Transactions on Evolutionary Computation*, **3** (1999), 257– 271.