Discrete Mathematics and Theoretical Computer Science **6**, 2003, 107–122

Efficient maxima-finding algorithms for random planar samples

Wei-Mei Chen[1] and Hsien-Kuei Hwang[2] and Tsung-Hsi Tsai[2]

[1] *Department of Applied Mathematics, Tatung University, Taipei 104, Taiwan*
[2] *Institute of Statistical Science, Academia Sinica, Taipei 115, Taiwan*

received 12 September, 2003, *revised 22 September, 2003*, *accepted 7 October, 2003*.

We collect major known algorithms in the literature for finding the maxima of multi-dimensional points and provide a simple classification. Several new algorithms are proposed. In particular, we give a new maxima-finding algorithm with expected complexity $n+O(\sqrt{n\log n})$ when the input is a sequence of points uniformly chosen at random from general planar regions. We also give a sequential algorithm, very efficient for practical purposes.

Keywords: Maximal points, computational geometry, probabilistic analysis of algorithms, Pareto optimality, sieve algorithms, dominance

1 Introduction

Finding the maxima of a given point set is a fundamental problem in computational geometry, which also arises in many applications such as Pareto-optimality in bargaining games or multicriteria optimization, analysis of linear programming, analysis of unbounded knapsack problem, statistical decision theory, etc.; see below for a brief list of instances. In the special case when $d=1$, it reduces to finding the maximum value of n numbers. We propose in this paper a simple classification of several known algorithms for finding the maxima, together with several new algorithms; among these are two efficient algorithms—one with expected complexity $n+O(\sqrt{n\log n})$ when the point samples are issued from some planar regions, and another more efficient than existing ones.

A point $p=(x_1,\dots,x_d)$ in $\mathbb{R}^d$ is said to *dominate* another point $q=(y_1,\dots,y_d)$ if $x_i>y_i$ for all $i=1,\dots,d$. For convenience, we write $p\succ q$. The nondominated points in a point set are called *maximal points* or *maxima*. Similar terms like Pareto optimality (or Pareto front, Pareto solutions, etc.), admissibility, efficiency, and noninferiority have been used in other fields. Dominance of multidimensional points is a natural and one of the most widely used partial orders in diverse fields such as

1. multicriteria decision analysis: see [35, 52];
2. Pareto optimality in multicriteria optimization: see [29, 51];
3. evolutionary computation: see [62, 66];
4. VLSI design: see [42, 47, 58];

5. networks: see [33, 59, 64, 65];
6. knapsack problems: see [2, 27, 39];
7. longest common subsequence problem: see [3, 37];
8. data mining: see [13, 31];
9. graph drawing: see [7, 40];
10. heuristic search in artificial intelligence: see [1, 22, 49, 57];
11. analysis of linear programming: see [15, 16];
12. transportation: see [18, 20, 21];
13. gene finding: see [32, 38];
14. statistical decision theory: see [55, 63];
15. ecological process models: see [54, 61].

Also many problems in computational geometry involve the maxima-finding as a component problem; see [14, 28, 36, 53]. For more information and references, see [4, 5, 6], [19, 66] and the webpages *List of References on Evolutionary Multiobjective Optimization* maintained by C. A. Coello Coello and *Multiple Criteria Decision Aid Bibliography* by V. Mousseau. In particular, maxima-finding for points lying in bounded regions can essentially be formulated as an area-minimization problem; see [6].

Several algorithms are known for finding the maxima of given points, and they can be simply classified by looking at the pattern of the corresponding version for $d = 1$. Actually, we start from looking at the design paradigm of algorithms for finding the maximum of n numbers ($d = 1$), and then identify one natural algorithm for finding the maxima in higher dimension. The resulting classification is summarized in Table 1; see next section for a detailed discussion. As we will see, this viewpoint is not only coherent but also inspiring (for devising more new algorithms in higher dimensions).

$d = 1$	$d > 1$
Sequential algorithm	Sequential algorithm [46] [10, Algo. M3]
Divide-and-conquer algorithm	Divide-and-conquer algorithm [12, 23, 46]
Bucket-select algorithm	Bucket algorithm [24, 25]
Quickselect algorithm	Selection algorithm [17]
Sieve algorithm	Sieve algorithm [10, Algo. M1, M2]

Tab. 1: *A classification of known maxima-finding algorithms according to the design pattern for* $d = 1$.

Apart from the preceding classification, we are mainly interested in this paper in algorithms of incremental type, which are in general simple and have good expected performance. We briefly describe in the next section known algorithms for $d = 1$, and then give a corresponding version for higher dimensions. Our new algorithms for general planar regions are given in Sections 3 and 4. A comparative discussion of the latter algorithm with the list algorithm of Bentley et al. [10] is also given. We then discuss various aspects of different design paradigms in the last section.

2 Maxima-finding algorithms: from $d = 1$ to $d > 1$

How to find the maximum of n numbers? We collect below several simple algorithms and state a natural extension to finding the maxima in higher dimensions.

Sequential algorithms. The most straightforward algorithm for finding the maximum value of n numbers has the following incremental, on-line, one-loop pattern:

Algorithm SEQUENTIAL
$//$**input** $= \{a_1, \dots, a_n\}$, $a_j \in \mathbb{R}, j = 1, \dots, n//$
$m := a_1$
for $i := 2$ **to** n **do**
  if $a_i > m$ **then** $m := a_i$
$//$**output** $m = \max\{a_1, \dots, a_n\}//$

Obviously, the number of comparisons used by this algorithm is $n - 1$.
Such a pattern has a natural extension to multidimensional inputs (see [10, 46]):

Algorithm SEQUENTIAL(d)
$//$**input** $= \{a_1, \dots, a_n\}$, $a_j \in \mathbb{R}^d, j = 1, \dots, n, d \geq 2//$
$M := \{a_1\}$
for $i := 2$ **to** n **do**
  if a_i dominates some points in M **then**
    delete these points; $M := M \cup \{a_i\}$
  elseif $\forall b \in M$, a_i is incomparable with b **then**
    $M := M \cup \{a_i\}$
$//$**output** $M =$maxima$\{a_1, \dots, a_n\}//$

Here we say that two points p and q are *incomparable* if neither p dominates q nor q dominates p. Kung et al. [46] used *balanced search trees* to maintain the maxima found so far, while Bentley et al. [10] used a *list* to implement the steps inside the for-loop, and apply the move-to-front heuristic to the dominating point in the list of maxima when new-inserting points are dominated. We will see later that an implementation using *random search trees* turns out to be very efficient in practice.

Divide-and-conquer algorithms. Another simple way to find the maximum of n numbers is to use the half-half divide-and-conquer paradigm:

Algorithm DIVIDE-AND-CONQUER
$//$**input** $\{a_1, \dots, a_n\}$, $a_j \in \mathbb{R}, j = 1, \dots, n//$
maximum$\{a_1, \dots, a_n\}$ $(n \geq 1)$
  if $n = 1$ then **return** a_n
  $m_1 :=$ **maximum**$\{a_1, \dots, a_{\lfloor n/2 \rfloor}\}$
  $m_2 :=$ **maximum**$\{a_{\lfloor n/2 \rfloor + 1}, \dots, a_n\}$
  if $m_1 \geq m_2$ **then return** m_1
  else return m_2
$//$**output** $\max\{a_1, \dots, a_n\}//$

It is easily seen that the number of comparisons used is also $n-1$. Indeed, the solution to the recurrence $f_n = f_{\lambda(n)} + f_{n-\lambda(n)} + 1$ with $f_1 = 0$ and $\lambda(n) \in \{1,\dots,n-1\}$ satisfies $f_n = n-1$. Thus the half-half dividing rule can be replaced by other dividing rules without changing the (optimum) worst-case complexity.

This design pattern, although less useful for $d = 1$, turned out to be very fruitful for higher dimensions; see [9, 11, 12, 23, 30, 46, 53].

Bucket algorithms. The top-down description of the divide-and-conquer algorithm suggests another algorithm, bucket-select, for finding selected order statistics in a given sequence; see Mahmoud et al. [48]. The generalization of this algorithm to finding the maxima in higher dimensions is also straightforward; see Devroye [24, 25]. No description of these bucket algorithms is given here since we are not discussing them in this paper. Note that unlike sequential and divide-and-conquer algorithms (deterministic in nature), the performance of bucket algorithms is more sensitive to the distribution of the input.

Selection algorithms. Yet another algorithm for finding the maximum of n numbers is quickselect when always selecting the largest order statistics:

Algorithm QUICKSELECT
//**input** $P := \{a_1,\dots,a_n\}$, $a_j \in \mathbb{R}$, $j = 1,\dots,n$//
quickselect(P) $(|P| \geq 1)$
  if $|P| = 1$ **then return** P
  $x :=$ a "pivot" element from P
  $Q := \{p \in P : p > x\}$
  if $|Q| = 0$ **then return** x
  else quickselect(Q)
//**output** $\max\{a_1,\dots,a_n\}$//

Although the worst-case of this algorithm is quadratic, its expected number of comparisons (assuming that the input is a sequence of independent and identically distributed random variables with a common continuous distribution) is equal to $2n - 2\sum_{1\leq j\leq n} 1/j$; see Knuth [44].

This simple pattern can be easily extended to finding the maxima in $\mathbb{R}^2$. One natural pattern is the following sieve algorithm.

Algorithm SELECTION
//**input** $= P := \{a_1,\dots,a_n\}$, $a_j \in \mathbb{R}^2$, $j = 1,\dots,n$//
selection(P)
  if $|P| = 1$ **then return** P
  choose a maximal point, say x in P as the sieve
  discard points dominated by x
  $M := \{x\}$
  for each nonempty quadrant Q of x **do**
    $M := M \cup$ **selection**(Q)
  return M
//**output** $M = \text{maxima}\{a_1,\dots,a_n\}$//

The expected complexity of this algorithm can be quickly seen from considering the special case when the points are drawn uniformly at random, one independently from the other, from the right triangle with corners $(0,0)$, $(0,1)$ and $(1,0)$. A good sieve, suggested by the probabilistic analysis in [5], is the point, say x, that maximizes the sum of both coordinates. The algorithm thus proceeds by first locating x, and then continues recursively for points lying in the second and fourth quadrants of x, respectively. The expected number of comparisons μ_n satisfies $\mu_0 := 0$ and the recurrence (see [5])

$$\mu_n = \beta n + c + \frac{(n-1)!}{\Gamma(n+1/2)} \sum_{2 \le j < n} \frac{\Gamma(j+1/2)}{j!} \mu_j \qquad (n \ge 1),$$

(assuming that $\beta n + c$ comparisons on average are used to split the problem into two smaller problems) where Γ denotes the Gamma function. The exact solution is given by (see [5])

$$\begin{aligned} \mu_n &= 3\beta n + 2\beta - c - (2\beta - c)\sqrt{\pi}\frac{n!}{\Gamma(n+1/2)} \qquad (n \ge 1) \\ &= 3\beta n + O(\sqrt{n}). \end{aligned}$$

Thus *the expected complexity of such a selection algorithm is linear*. The limit distribution of the cost can also be derived by the approach used in [5] if more information on the "splitting toll cost" is available.

Note that the probabilistic analysis is in general less transparent for random points from other regions. For multidimensional points, a similar algorithm can be devised but an additional merge as that used in divide-and-conquer algorithms is required.

Another sequential selection algorithm is discussed in Clarkson [17] in which maxima are found one after another by the following steps: compare each incomparable element p thus far to the set of maxima (initially empty); drop p if it is dominated; otherwise p remains incomparable and is then used to find a new maximal element by scanning through the remaining incomparable elements, updating p if p is dominated (again, dominated elements are dropped in the process). The procedure is reminiscent of the selection sort. Clarkson's algorithm works for all dimensions, but becomes less useful if maxima are abundant.

Sieve algorithms. Selection algorithms spend some effort in finding first a good sieve point; when more information on the input is known (in addition to the dimension, like the underlying distribution), one may quickly identify a sieve point with almost no additional cost, and with good sieving power. A naive pattern for $d = 1$ is as follows.

```
Algorithm SIEVE
//input P := {a_1, ..., a_n}, a_j ∈ ℝ, j = 1, ..., n//
M := ∅
choose a sieve point, say x        //x ∉ P//
for i := 1 to n do
    if a_i > x then M := M ∪ {a_i}
if |M| > 0 then
    find the maximum value in M
else
    use other maximum-finding algorithm
//output max{a_1, ..., a_n}//
```

If the underlying distribution of the input is Uniform$(0,1)$, then we can take $x = 1-(\log n)/n$, so that M is empty with probability proportional to n^{-1}. If that really happens, then we can use any algorithm like SEQUENTIAL to find the maximum from zero, and the expected cost so introduced is bounded. The algorithm has thus an overall expected cost satisfying $n+O(\log n)$ if SEQUENTIAL is used when $|M|>0$.

Such a sieve-based approach is, although not useful for $d=1$, easily rewritten for higher dimensional problem:

Algorithm SIEVE(d)
//**input** $P=\{a_1,\dots,a_n\}$, $a_j \in \mathbb{R}^d$, $j=1,\dots,n$, $d \geq 2$//
$M := \emptyset$
choose a sieve point, say x //$x \notin P$//
for $i := 1$ **to** n **do**
    if a_i is not dominated by x **then** $M := M \cup \{a_i\}$
if all maxima are included in M **then**
    find these maxima
else
    use other maxima-finding algorithm
//**output** maxima$\{a_1,\dots,a_n\}$//

The two other algorithms proposed in [10] share the same pattern as above, and as good sieves for random points issuing from $[0,1]^d$, they take

1. $x=(1-\delta_n,\dots,1-\delta_n) \in \mathbb{R}^d$, where $\delta_n = n^{-1/d}(\log n)^{1/d}$;
2. $x=(\xi_n^{[1]},\dots,\xi_n^{[d]}) \in \mathbb{R}^d$, where $\xi_n^{[i]}$ denotes the $\lfloor n\delta_n \rfloor$-th largest element in the i-th dimension.

Both choices of p guarantee that all but $O(n^{1-1/d}(\log n)^{1/d})$ points on average are sieved out by p when the points are independently and identically chosen from $[0,1]^d$. To check the condition (or certificate) "all maxima included in M", one can divide points in M into two sets when filtering the points: points dominating p and points incomparable with p. Then the condition "all maxima included in M" holds if there are points dominating p; otherwise, we can resort to any algorithm with even $O(n^2)$ worst-case time bound.

3 A new sieve algorithm

The efficiency of sieve algorithms relies heavily on the quality of the sieve, which in turn depends on the underlying distribution. By construction, both sieve points for SIEVE(d) given in [10] are based on the assumption that the underlying shape (for the input points) is a hypercube (or, more generally, having an upper-right corner†; see [5]). It is obvious that when the underlying planar shape does not have an upper-right corner such as a circle or a triangle of the form ◺, no point inside the region has the power of dominating all but $O(n^{1-\varepsilon})$ points.

We propose a new sieve algorithm based on the same design pattern as SIEVE(d) but using a curve as a sieve instead of a point, so that almost all points can be discarded. While the idea is straightforward, finding a good certificate for assuring that all maxima are in the sieved region is non-trivial.

† Roughly, an upper-right corner of a bounded region R is a point on the boundary of R dominating all points in and on R.

Let $X_1, X_2, ..., X_n$ be independent and identically distributed random vectors in $\mathbb{R}^2$. Assume that X_1 is uniformly distributed in a region

$$D := \{(x,y) : 0 \le x \le 1, 0 \le y \le f(x)\}, \tag{1}$$

where $f(x)$ is a nonincreasing function on $[0,1]$ with $f(0) > 0$.

The reason of considering the shape D is that for general planar regions most maxima lie in the upper-right part; see [5] for more details. Also the case when $f(x) = 1-x$ is a prototype for general regions; see [6]. Our result on D can be easily extended to general planar regions R provided that the boundary of the upper-right part of R has the form of D.

Without loss of generality, we may assume that $|D| = 1$ and that

$$f(x) \begin{cases} > 0, & \text{for } 0 \le x < 1; \\ = 0, & \text{for } x \ge 1. \end{cases}$$

Let

$$S := \{(x,y) \in D : y + \delta < f(x+\delta)\};$$

see Figure 3. If $\delta > 0$ is suitably chosen, then there is no maxima in S and the number of points in $D - S$ is $O(\sqrt{n \log n})$ with probability $1 - o(n^{-1.5})$. Then we apply the bucket algorithm (see [25]) for points in $D - S$ and read off the maximal points there in a suitable order; see below for more details.

To state the algorithm precisely, we need the following lemma.

Lemma 1 *Let*

$$Z := \left\{(x,y) \in D : y + \frac{\delta}{2} \ge f\left(x + \frac{\delta}{2}\right)\right\},$$

and

$$\alpha := \inf\left\{x : f\left(x + \frac{\delta}{2}\right) \le \frac{\delta}{2}\right\}.$$

If

$$(0,\alpha) \subset \bigcup_{(x_i,y_i)\in Z} \left(x_i - \frac{\delta}{4}, x_i + \frac{\delta}{4}\right), \tag{2}$$

where $X_i = (x_i, y_i)$, then there is no maxima in S.

Proof. Suppose that there is a maximum in S. Then there exists a point (x_0, y_0) such that $y_0 + \delta = f(x_0 + \delta)$ and there is no point in the first quadrant of (x_0, y_0). We now find a point $(x_0', y_0) \in \partial Z$, i.e. $y_0 + \frac{\delta}{2} = f(x_0' + \frac{\delta}{2})$. Since $f(x_0 + \delta) > f(x_0' + \frac{\delta}{2})$ and f is nonincreasing,

$$x_0' - x_0 > \frac{\delta}{2};$$

and none of the points $\{X_1, \ldots, X_n\}$ is in

$$Z \cap \{(x,y) : x_0 < x < x_0'\}.$$

These contradict (2). □

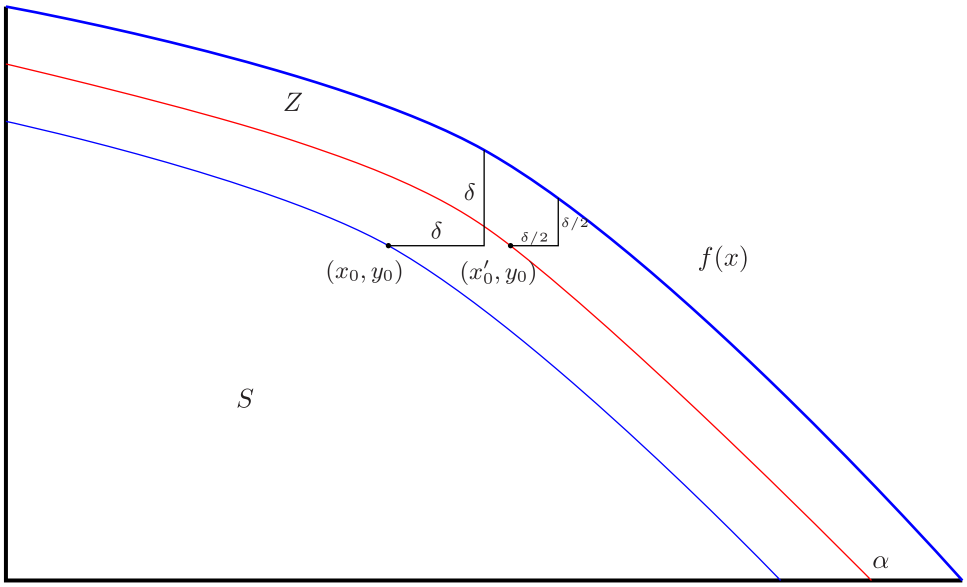

Fig. 1: *Dissection of the region D.*

In view of this lemma, the pattern of our algorithm is then as follows.

Algorithm CURVE-SIEVE
// **input** $= \{a_1, \ldots, a_n\}$, $a_j \in \mathbb{R}^2$, $j = 1, \ldots, n$//
$M := \emptyset$
for $i := 1$ **to** n **do**
  if $a_i.y + \delta > f(a_i.x + \delta)$ **then** $M := M \cup \{a_i\}$
if (2) holds **then** find these maxima
else use other maxima-finding algorithm
//**output** maxima$\{a_1, \ldots, a_n\}$//

Here $a_i.x$ and $a_i.y$ denotes respectively the x and y coordinates of a_i.

Assume that comparisons of the type $a_i.y + \delta > f(a_i.x + \delta)$ takes unit cost. Let C_n denote the number of comparisons used by the algorithm CURVE-SIEVE for finding the maxima of $\{X_1, \ldots, X_n\}$.

Theorem 1 *Take* $\delta = 4\sqrt{\frac{\log n}{n}}$. *Then the average number of comparisons used by the* CURVE-SIEVE *algorithm satisfies*

$$E(C_n) = n + O\left(\sqrt{n \log n}\right).$$

Proof. The number of comparisons used to test if a point lies in S or $D - S$ is n. Assume at the moment that the expected number of comparisons used in checking (2) is proportional to the number of points in

$D-S$. Then we have

$$C_n = \begin{cases} n+O(\text{the number of points in } D-S), & \text{if (2) holds;} \\ O\left(n^2\right), & \text{otherwise.} \end{cases}$$

Now we show that the probability of the event that (2) fails is less than $o(n^{-1.5})$. Let $m = \lceil 4\alpha/\delta \rceil + 1$ and define the strips

$$Z_i := Z \cap \{(x,y) : \delta(i-1)/4 < x < \delta i/4\},\ 1 \le i \le m.$$

If (2) fails, then there is at least a strip Z_j, $j \in \{0,1,\ldots,m\}$, in which there is no point. Observe that due to our choice of α, the height of Z_i is no less than $\delta/2$, and thus $|Z_i| \ge \delta^2/8$. It follows that the probability that (2) fails is bounded above by

$$\sum_{1\le i\le m} (1-|Z_i|)^n \le m\left(1-\delta^2/8\right)^n = o(n^{-1.5}).$$

Observe that

$$|D-S| = \int_0^1 (f(x) - \max\{0, f(x+\delta)-\delta\})\,\mathrm{d}x \le (1+f(0))\delta = O\left(\sqrt{\frac{\log n}{n}}\right), \tag{3}$$

where the O-term depends on $f(0)$, which is finite by our assumption.

Since the probability of the event that the number of points in $D-S$ is more than $O(\sqrt{n\log n})$ is less than $o(n^{-1})$, we have

$$\begin{aligned} E(C_n) &= \left(n+O\left(\sqrt{n\log n}\right)\right)\left(1-o(n^{-1})\right) + O(n)\,o(n^{-1}) + O\left(n^2\right)o(n^{-1.5}) \\ &= n+O\left(\sqrt{n\log n}\right). \end{aligned}$$

We still need to show that (i) the maxima in $D-S$ can be found in time $O(\sqrt{n\log n})$, and (ii) the condition (2) can be checked within the same time bound.

The problem (ii) is easy: we divide the unit interval into $\lceil 2/\delta \rceil$ buckets, find points in each interval with the maximum and minimum x-coordinates, respectively, and then the condition (2) can be easily checked. This is essentially the so-called maximal gap problem; see [25, p. 85].

For problem (i), we use the bucket argument used in Devroye [25] by dividing the region $D-S$ into small buckets each of which has at most $O(1)$ expected number of points; this is achievable by uniformity of the input points in D and by (3). We scan the strips from right to left and for the buckets in each strip we go from top to bottom, checking for the maximal points and each time keeping track of the topmost nonempty bucket in each strip (so that we need only to check buckets with higher values of y-coordinates for the strip left to the current one). □

Special cases of $f(x)$. (i) When $f(x)=1$ for $0\le x\le 1$, our sieve equation becomes

$$y \begin{cases} > 1-\delta, & \text{if } x < 1-\delta; \\ = 0, & \text{if } 1-\delta \le x \le 1. \end{cases}$$

(ii) if $f(x) = 2(1-x)$, then $\alpha = 1-3\delta/4$ and the sieve equation is $y < 2(1-x)-3\delta$ for $x \le \alpha$.

Extension to higher dimensions. The same sieving procedure can be applied to higher dimensions with suitable modifications, but the bucket argument (used to guarantee a linear expected time bound) does not extend easily. One may apply the divide-and-conquer algorithm (see [46, 53]) instead of bucket algorithm for points not sieved out, but the error term becomes slightly bigger $O(n^{1-1/d}(\log n)^c)$ for some constant $c > 1/d$ depending on d.

4 A sequential tree algorithm

Our description of SEQUENTIAL(d) is completely general in that we can implement the steps inside the for-loop by different types of data structures. The original implementation by Bentley et al. [10] used a list (together with the move-to-front heuristic), and it is proved by Devroye [26] that the expected number of vector comparisons used by the list algorithm (with or without the move-to-front rule) satisfies $n+o(n)$ if the input points are taken uniformly at random from $[0,1]^d$, one independently of the others; see also Clarkson [17], Golin [34] for $d = 2$.

When the underlying distribution changes from the hypercubes to other regions, simulations suggest that the list algorithm may become less efficient, although rigorous analysis is still lacking; see [10, 34]. We propose in this section an implementation using random search trees for $d = 2$ (instead of balanced search trees as in [46]). The tree is essentially a quadtree, but without the first and the third quadrants. The algorithm is very efficient in practice and robust to different distributions, but the extension to higher dimensions and the analysis of its expected cost are not clear and left open.

Algorithm FIND-MAX
//**input**: A list of points $\mathcal{P} := \{p_i = (x_i, y_i) : 1 \le i \le n\}$
    $T \leftarrow \varnothing$
    for $i := 1$ **to** n **do**
        update-maxima(p_i, T)
//**output**: $T :=$ the tree containing all maximal vectors of $\mathcal{P}$

//**input**: A point p_i and $T :=$ the tree containing all maximal vectors of $\{p_1, p_2, \dots, p_{i-1}\}$.
update-maxima(p_i, T)
    if $(T = \varnothing)$ **then**
        insert p_i into T; **return**
    compare p_i and $T.key$ //$T.key :=$ point at the root node //
    case 1: p_i lies in the first quadrant of $T.key$
        $T.key := p_i$ //p_i replaces $T.key$ //
        prune$(left, T)$ //prune the left subtree //
        prune$(right, T)$ //prune the right subtree //
    case 2: p_i lies in the second quadrant of $T.key$
        update-maxima$(p_i, T.left)$
    case 3: p_i lies in the third quadrant of $T.key$
        do nothing
    case 4: p_i lies in the fourth quadrant of $T.key$
        update-maxima$(p_i, T.right)$
//**output**: $T :=$ the tree containing all maximal vectors of $\{p_1, p_2, \dots, p_i\}$.

//**input**: $b \in \{left, right\}$ and $T :=$ a tree to be properly pruned.
prune(b, T)
  check $:= T$
  next $:=$ *check.b*
  while $(next \neq \varnothing)$ **do**
    if (*next.key* lies in the third quadrant of *check.key*) **then**
      next $:=$ *next.b*
      remove *check.b* and *check.b*'s $\overline{b}$ subtrees
        //$\overline{left} := right$ and $\overline{right} := left$//
    else
      $b := \overline{b}$
      next $:=$ *check.b*
//**output**: more compact T

List algorithm vs tree algorithm. While a general analysis of the expected number of comparisons used by our tree algorithm is not easy, there is a special case when the analysis becomes straightforward: it is the case when

$$C_n^{\text{tree}}(A) \le C_n^{\text{list}}(A),$$

where A is the set of input points, C_n^{tree} denotes the number of comparison used by our tree algorithm, and C_n^{list} that by the list algorithm of Bentley et al. when the "move-to-front" heuristic is not used.

When the input points are taken independently and identically at random from the unit square, Devroye [26] showed that

$$E(C_n^{\text{list}}) = n + o(n),$$

and the same time bound applies to $E(C_n^{\text{tree}})$.

n	list	list+MTF	tree
10	1.499	1.307	1.316
10^2	1.808	1.560	1.406
10^3	1.573	1.309	1.187
10^4	1.370	1.201	1.069
10^5	1.261	1.136	1.028
10^6	1.198	1.084	1.010
10^7	1.137	1.025	1.003
10^8	1.095	1.016	1.0002

n	list	list+MTF	tree
10	2.076	1.792	1.655
10^2	4.695	3.683	2.442
10^3	7.249	4.901	2.567
10^4	9.672	5.237	2.543
10^5	12.629	7.019	2.512
10^6	14.351	9.189	2.513
10^7	17.585	10.918	2.517
10^8	21.326	13.866	2.576

Tab. 2: *Average number of comparisons per element (by simulations) when the underlying region is the unit square (left) and the right triangle with corners* $(0,0)$, $(0,1)$ *and* $(1,0)$ *(right); MTF denotes "move-to-front."*

Simulations also suggested that the number of comparisons has large variance (of the order of mean square), so that larger scale simulations are needed in order to obtain more meaningful approximations.

5 Comparative discussions

We briefly discuss the preceding algorithms from several different angles.

Paradigms	Main operations
Sequential algorithms	Insertion
Selection algorithms	Selection
Divide-and-conquer algorithms	Merging
Bucket algorithms	Distribution
Sieve algorithms	Sieving

Tab. 3: *Main operations used by each design paradigm.*

Maxima-finding vs sorting. Knuth [45] classified major sorting algorithms by the main operations used: *insertion, exchanging, selection, merging, and distribution*. An alternative classification, similar to that of Knuth's, of maxima-finding algorithms is given in Table 3.

Sorting before divide-and-conquer vs sorting after divide-and-conquer. Sorting is usually used before divide-and-conquer to achieve better worst-case performance. The scenario changes when considering average-case complexity because in most cases the number of maxima is at most $O(n^{1-\varepsilon})$ on average, and thus sorting after divide-and-conquer is preferable, especially when the number of maxima is small like $\log n$ or $\sqrt{n}$.

Probabilistic vs deterministic. Once a good sieve for sieve algorithms is selected, we may completely neglect the case when the "certificate" fails, giving a probabilistic algorithm that finds all maxima with high probability. This adds to the flexibility of sieve algorithms. However, the certificate like (2) may in general not easy to be devised.

Known vs unknown underlying distribution. Different cost is needed when finding the maximum of n numbers with given information. Typical examples include:

1. Maximum-finding without any *a priori* information on the input requires $n-1$ comparisons;
2. Zero comparison is needed if one knows that the given input is a sorted sequence;
3. Maximum-finding takes $\Theta(\sqrt{n})$ expected time if the input is a random walk; see Odlyzko [50];
4. A probabilistic algorithm (with an error probability $O(n^{-c})$ of finding the maximum) with cost satisfying $O((\log n)^2)$ is given in Ting and Yao [60];
5. $O(\log n)$ expected time is needed for finding the maximum value in a binary search tree.

Our sieve algorithm CURVE-SIEVE assumes that the input points are randomly taken from some planar region with about $O(\sqrt{n})$ maxima on average; are there good sieve algorithms when this *a priori* information of the input points is not available?

Worst-case vs typical case. The two-dimensional maxima problem may be regarded as an easy problem as far as worst-case complexity is concerned because one can simply sort the points according to their x-coordinates and then read off their y-coordinates from right to left, looking for right-to-left maximal elements. This results in an $O(n \log n)$ algorithm. Our main interests here lie in maxima-finding algorithms with linear expected cost and of practical value.

There are data structures in the literature capable of maintaining dynamically (insertion and deletion) the current maxima in worst-case time bound $\log n$, but their practical usefulness is unclear; see [41] and the references therein. For other notions like output-sensitive and distribution-sensitive algorithms, see [43, 56]. Again the main focus of these algorithms is on worst-case instead of average-case as we emphasized in this paper.

Square vs other regions. Most probabilistic analysis of maxima-finding algorithms assume that the input is uniformly distributed in a unit square. The results so obtained only reflect the tip of an iceberg since the number of maxima is very different for other general planar regions. Also efficient randomized algorithms for square may become less efficient for other inputs. See Fukuda et al. [31] for a concrete situation where different distributions arise.

Acknowledgements

We thank Ralph Neininger and Michael Fuchs for useful suggestions.

References

[1] N. Alechina and B. Logan, State space search with prioritised soft constraints, *Applied Intelligence*, **14** (2001), 263–272.

[2] R. Andonov, V. Poirriez and S. Rajopadhye, Unbounded knapsack problem: Dynamic programming revisited, *European Journal of Operational Research*, **123** (2000), 394–407.

[3] A. Apostolico and C. Guerra, The longest common subsequence problem revisited, *Algorithmica*, **2** (1987), 315–336.

[4] Z.-D. Bai, C.-C. Chao, H.-K. Hwang and W.-Q. Liang, On the variance of the number of maxima in random vectors and its applications, *Annals of Applied Probability*, **8** (1998), 886–895.

[5] 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**, paper no. 3, 41 pages (2001).

[6] Z.-D. Bai, H.-K. Hwang and T.-H. Tsai, Berry-Esseen bounds for the number of maxima in planar regions, *Electronic Journal of Probability*, **8**, paper no. 9, 26 pages (2003).

[7] G. Di Battista, P. Eades, R. Tamassia and I. G. Tollis, Algorithms for drawing graphs: an annotated bibliography, *Computational Geometry: Theory and Applications*, **4** (1994), 235–282.

[8] R. A. Becker, L. Denby, R. Mcgill and A. R. Wilds, Analysis of data from Places Rated Almanac, *American Statisticians*, **41** (1987), 169–186.

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

[10] 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.

[11] J. L. Bentley, H. T. Kung, M. Schkolnick and C. D. Thompson, On the average number of maxima in a set of vectors and applications, *Journal of the ACM*, **25** (1978), 536–543.

[12] J. L. Bentley and M. I. Shamos, Divide and conquer for linear expected time, *Information Processing Letters*, **7** (1978), 87–91.

[13] P. Berman, Z. Zhang, Y. I. Wolf, E. V. Koonin and W. Miller, Winnowing sequences from a database search, *Journal of Computational Biology*, **7** (2000), 293–302.

[14] S. Bespamyatnikh and D. Kirpatrick, Rectilinear 2-center problems, in *Proceedings of the 11th Canadian Conference on Computational Geometry (CCCG'99)*, pp. 68–71, 1999.

[15] C. Blair, Random inequality constraint systems with few variables, *Mathematical Programming*, **35** (1986), 135–139.

[16] C. Blair, Random linear programs with many variables and few constraints, *Mathematical Programming*, **34** (1986), 62–71.

[17] 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.

[18] D. D. L. Cardillo and T. Fortuna, A DEA model for the efficiency evaluation of nondominated paths on a road network, *European Journal of Operational Research*, **121** (2000), 549–558.

[19] C. A. Coello Coello, D. A. Van Veldhuizen amd G. B. Lamont, *Evolutionary Algorithms for Solving Multi-objective Problems*, Kluwer Academic Publishers, New York, 2002.

[20] R.-S. Daruwala, *On Computing the Pareto Optimal Solution Set in a Large Scale Dynamic Network*, Ph. D. Thesis, New York University, 2002; available at www.cs.nyu.edu/csweb/Research/theses.html.

[21] S. K. Das, A. Goswami and S. S. Alam, Multiobjective transportation problem with interval cost, source and destination parameters, *European Journal of Operational Research*, 117 (1999), 100–112.

[22] P. Dasgupta, P. P. Chakrabarti and S. C. DeSarkar, *Multiobjective Heuristic Search: An Introduction to Intelligent Search Methods for Multicriteria Optimization*, Vieweg Verlag, 1999.

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

[24] L. Devroye, A note on the expected time required to construct the outer layer, *Information Processing Letters*, **20** (1985), 255–257.

[25] L. Devroye, *Lecture Notes on Bucket Algorithms*, Progress in Computer Science, 6, Birkhäuser, Boston, MA, 1986.

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

[27] M. E. Dyer and J. Walker, Dominance in multi-dimensional multiple-choice knapsack problems, *Asia-Pacific Journal of Operational Research*, **15** (1998), 159–168.

[28] H. Edelsbrunner and M. H. Overmars, On the equivalence of some rectangle problems, *Information Processing Letters*, **14** (1982), 124–127.

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

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

[31] T. Fukuda, Y. Morimoto, S. Morishita and T. Tokuyama, Interval finding and its application to data mining, *IEICE Transaction on Fundamentals of Electronics, Communications and Computer Sciences*, **E80-A** (1997), 620–626.

[32] M. S. Gelfand, L. I. Podolsky, T. V. Astakhova, M. A. Roytberg, Recognition of genes in human DNA sequences, *Journal of Computational Biology*, **3** (1996), 223–234.

[33] T. Getachew, M. Kostreva and L. Lancaster, A generalization of dynamic programming for Pareto optimization in dynamic networks, *RAIRO Operations Research*, **34** (2000), 27–47.

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

[35] S. Greco, B. Matarazzo and R. Slowinski, Rough approximation by dominance relations, *International Journal of Intelligent Systems*, **17** (2002), 153–171.

[36] R. H. Güting, O. Nurmi and T. Ottmann, Fast algorithms for direct enclosures and direct dominances, *Journal of Algorithms*, **10** (1989), 170–186.

[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] A. Hero and G. Fleury, Pareto-optimal methods for gene analysis, preprint (2002); available at www.eecs.umich.edu/~hero/bioinfo.html.

[39] R. E. Johnston and L. R. Khan, A note on dominance in unbounded knapsack problems, *Asia-Pacific Journal of Operational Research*, **12** (1995), 145–160.

[40] T. Kameda, On the vector representation of the reachability in planar directed graphs, *Information Processing Letters*, **3** (1975), 75–77.

[41] S. Kapoor, Dynamic maintenance of maxima of 2-d point sets, *SIAM Journal on Computing*, **29** (2000), 1858-1877.

[42] V. Kathail, S. Aditya, R. Schreiber, B. R. Rau, D. C. Cronquist and M. Sivaraman, PICO: automatically designing custom computers, *IEEE Computer*, **35**:9 (2002), 39–47.

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

[44] D. E. Knuth, Mathematical analysis of algorithms, in *IFIP Congress*, **1** (1971), 19–27.

[45] D. E. Knuth, *The Art of Computer Programming. Volume 3. Sorting and Searching*, Second Edition, Addison-Wesley, Reading, MA, 1998.

[46] 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.

[47] J. Lillis and C. K. Cheng, Timing optimization for multisource nets: characterization and optimal repeater insertion, *IEEE Transations on Computer Aided Design of Integrated Circuits and Systems*, **18** (1999), 322–331.

[48] H. M. Mahmoud, P. Flajolet, P. Jacquet and M. Régnier, Analytic variations on bucket selection and sorting, *Acta Informatica*, **36** (2000), 735–760.

[49] M. Müller, Partial order bounding: A new approach to evaluation in game tree search, *Artificial Intelligence*, **129** (2001), 279–311.

[50] A. M. Odlyzko, Search for the maximum of a random walk, *Random Structures and Algorithms*, **6** (1995), 275–295.

[51] B. O'Neill, The number of outcomes in the Pareto-optimal set of discrete bargaining games, *Mathematics of Operations Research*, **6** (1980), 571–578.

[52] J.-C. Pomerol and S. Barba-Romero, *Multicriterion Decision in Management*, Kluwer Academic Publishers, Boston, 2000.

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

[54] J. H. Reynolds and E. D. Ford, Multi-criteria assessment of ecological process models, *Ecology*, **80** (1999), 538–553.

[55] A. L. Rukhin, Admissibility: survey of a concept in progress, *International Statistical Review*, **63** (1995), 95-115.

[56] S. Sen and N. Gupta, Distribution-sensitive algorithms, *Nordic Journal of Computing*, **6** (1999), 194–211.

[57] B. S. Stewart and C. C. White, III, Multiobjective A^*, *Journal of the ACM*, **38** (1991), 775–814.

[58] Y. Sun and C. L. Liu, An area minimizer for floorplans with L-shaped regions, *Proceedings of the IEEE ICCD'92*, (1992) pp. 383–386.

[59] G. K. Tayi, D. J. Rosenkrantz and S. S. Ravi, Path problems in networks with vector-valued edge weights, *Networks*, **34** (1999), 19–35.

[60] H. F. Ting and A. C. Yao, A randomized algorithm for finding maximum with $O((\log n)^2)$ polynomial tests, *Information Processing Letters*, **49** (1994), 39–43.

[61] Y. Tyutyunov, I. Senina, C. Jost and R. Arditi, Risk assessment of the harvested pike-perch population of the Azov Sea, *Ecological Modelling*, **149** (2002), 297–311.

[62] D. A. van Veldhuizen and G. B. Lamont, Multiobjective evolutionary algorithms: analyzing the state-of-the-art, *Evolutionary Computation*, **8** (2000), 125–147.

[63] A. Wald, *Statistical Decision Functions*, second edition, Chelsea, New York, 1971.

[64] A. Warburton, Aproximation of Pareto optima in multiple-objective shortest-path problems, *Operations Research*, **35** (1987), 70–79.

[65] M. Zachariasen, Rectilinear full Steiner tree generation, *Networks*, **33** (1999), 125–143.

[66] E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello, and D. Corne (Editors), *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization* (EMO 2001), Lecture Notes in Computer Science, vol. 1993, Springer-Verlag, 2001.