Improved Deterministic Algorithms for Linear Programming in Low Dimensions[*]

Timothy M. Chan[†]

October 16, 2017

Abstract

Chazelle and Matoušek [*J. Algorithms*, 1996] presented a derandomization of Clarkson's sampling-based algorithm [*J. ACM*, 1995] for solving linear programs with n constraints and d variables in $d^{(7+o(1))d}n$ deterministic time. The time bound can be improved to $d^{(5+o(1))d}n$ with subsequent work by Brönnimann, Chazelle, and Matoušek [*SIAM J. Comput.*, 1999]. We first point out a much simpler derandomization of Clarkson's algorithm that avoids ε-approximations and runs in $d^{(3+o(1))d}n$ time. We then describe a few additional ideas that eventually improve the deterministic time bound to $d^{(1/2+o(1))d}n$.

1 Introduction

This paper studies the well-known *linear programming* problem and focuses on algorithms in the *real RAM* model, where running time (the number of arithmetic operations) is analyzed as a function of the number of constraints n and the number of variables d, but not the bit complexity of the input. The major open question is whether a polynomial algorithm on the real RAM, also known as a "strongly polynomial" algorithm, exists for linear programming. (In contrast, for the latest advance on polynomial-time linear programming algorithms that depend on the input bit complexity, see [20], for example.)

In the early 80s, Megiddo [24] discovered a linear programming algorithm that runs in $O(n)$ time for any constant dimension d; this result has become a cornerstone of computational geometry, with many applications in low-dimensional settings. Since Megiddo's work, many other linear-time algorithms have been proposed, which improve the (super)exponential dependency of the hidden factor on d; the story has been recounted in many books and surveys [3, 7, 13, 16, 21, 25]. Table 1 summarizes the previous results, and also states our new result.

As one can see, our improvement is a modest one and does not alter the form $d^{O(d)}n$ of the best known deterministic time bounds; also, our new algorithm does not beat existing randomized algorithms. However, we believe the result is still interesting for several reasons. First, linear programming is among the most fundamental algorithmic problems of all time, and no progress on deterministic real-RAM algorithms has been reported for years. Second, we obtain a dependency

[*]A preliminary version of this paper has appeared in *Proc. 27th Annual ACM–SIAM Symposium on Discrete Algorithms* (SODA), pages 1213–1219, 2016. This work was done while the author was at the Cheriton School of Computer Science, University of Waterloo, Canada.

[†]Department of Computer Science, University of Illinois at Urbana-Champaign, USA (tmc@illinois.edu)

simplex method	det.	$O(n/d)^{d/2+O(1)}$
Megiddo [24]	det.	$2^{O(2^d)}n$
Clarkson [9]/Dyer [14]	det.	$3^{d^2}n$
Dyer and Frieze [15]	rand.	$O(d)^{3d}(\log d)^d n$
Clarkson [10]	rand.	$d^2n + O(d)^{d/2+O(1)}\log n + d^4\sqrt{n}\log n$
Seidel [26]	rand.	$d!n$
Kalai [19]/Matoušek, Sharir, and Welzl [23]	rand.	$\min\{d^2 2^d n,\ e^{2\sqrt{d\ln(n/\sqrt{d})}+O(\sqrt{d}+\log n)}\}$
combination of [10] and [19, 23]	rand.	$d^2n + 2^{O(\sqrt{d\log d})}$
Hansen and Zwick [18]	rand.	$2^{O(\sqrt{d\log((n-d)/d)})}n$
Agarwal, Sharir, and Toledo [4]	det.	$O(d)^{10d}(\log d)^{2d}n$
Chazelle and Matoušek [8]	det.	$O(d)^{7d}(\log d)^d n$
Brönnimann, Chazelle, and Matoušek [5]	det.	$O(d)^{5d}(\log d)^d n$
this paper	det.	$O(d)^{d/2}(\log d)^{3d}n$

Table 1: Deterministic and randomized time bounds for linear programming on the real RAM.

of $d^{(1/2+o(1))d}$ on the dimension, reaching a natural limit for the kinds of approach considered here and in Chazelle and Matoušek's prior work [8]. These are based on derandomization of Clarkson's sampling algorithms [10], which reduce the problem to subproblems with no fewer than $n \approx d^2$ constraints, for which the currently best deterministic real-RAM upper bound is about $(n/d)^{d/2} = d^{d/2}$, as guaranteed by the simplex method. (The best randomized algorithms by Kalai [19] and Matoušek, Sharir, and Welzl [23] are faster, but no derandomization of results of that kind is known.) Note that our new $d^{(1/2+o(1))d}n$ deterministic result even beats the well-loved randomized $O(d!n)$-time algorithm by Seidel [26]; and it is superior to all existing deterministic bounds on the real RAM for $n \gg d^2$.

Our technical ideas are also noteworthy in several ways:

1. We observe that derandomizing Clarkson's recursive sampling algorithm [10] can lead to a *simple* deterministic linear-time algorithm for constant dimensions. Chazelle and Matoušek [8] previously derandomized Clarkson's algorithm, but needed a combination of several ideas (including the *method of conditional probabilities* and a nontrivial *merge-and-reduce* algorithm for computing *ε-approximations*). In contrast, our simplest derandomization can be described in under two pages, as we explain in Section 2, and is completely self-contained except for the use of the standard greedy algorithm for set cover or hitting set. The resulting algorithm is actually simpler than Megiddo's original deterministic algorithm [24] and all its subsequent refinements [14, 4]. This is perhaps the main takeaway of the paper. (Megiddo's algorithm grouped the input into pairs and invoked linear-time median finding repeatedly; our algorithm uses groups of a larger size but does not need median finding at all. Megiddo's algorithm also required more complicated primitive operations; our algorithm only requires standard orientation tests on $d+1$ input hyperplanes.) Incidentally, even without attempting to optimize the dependency on d, our algorithm in its simplest version has a running time at most $d^{(3+o(1))d}n$, already improving all prior published deterministic bounds.

2. Clarkson's paper [10] described a second sampling algorithm, based on *iterative reweighting* (or in more trendy terms, the *multiplicative weights update* method). This second algorithm has expected $n\log n$ running time in terms of n, but has better dependency on d. We point

out a simple variant of Clarkson's second algorithm that has running time linear in n; this variant appears new. If randomization is allowed, this isn't too interesting, since Clarkson [10] eventually combined his two algorithms to obtain his best results with running time linear in n. However, the variant makes a difference for deterministic algorithms, as we will see in Section 3.

3. The heart of the matter in obtaining the best derandomization of Clarkson's algorithms lies in the construction of *ε-nets*. This was the focus of the previous paper by Chazelle and Matoušek [8]. In Section 4, we present an improved ε-net algorithm (for halfspace ranges) which has running time about $O(1/\varepsilon)^{d/2}$ when n is polynomial in d, and which may be of independent interest. This is the most technical part of the paper (here we need to go back to previous derandomization concepts, namely, *(sensitive) ε-approximations*). Still, the new idea can be described compactly with an interesting recursion.

2 Clarkson's First Algorithm

In linear programming, we want to minimize a linear function over an intersection of n halfspaces. Without loss of generality, we assume that the halfspaces are in general position (by standard perturbation techniques) and all contain $(0, \ldots, 0, -\infty)$ (by adding an extra dimension if necessary). The problem can then be restated as follows:

> Given a set H of n hyperplanes in $\mathbb{R}^d$, find a point p that lies on or below[1] all hyperplanes in H, while minimizing a given linear objective function.

All our algorithms rely on the concept of ε-nets, which in this context can be defined as follows:

Definition 2.1. Given $p \in \mathbb{R}^d$, let $\text{Violate}_p(H) = \{h \in H : h \text{ is strictly below } p\}$. A subset $R \subseteq H$ is an *ε-net* of H if for every $p \in \mathbb{R}^d$,

$$\text{Violate}_p(R) = \emptyset \;\Rightarrow\; |\text{Violate}_p(H)| \leq \varepsilon |H|.$$

Fact 2.2.

(a) (Mergability) *If R_i is an ε-net of H_i for each i and the H_i's are disjoint, then $\bigcup_i R_i$ is an ε-net of $\bigcup_i H_i$.*

(b) *Given a set H of $n \geq d$ hyperplanes in $\mathbb{R}^d$, we can construct an ε-net of H of size $O(\frac{d}{\varepsilon} \log n)$ in $O(n/d)^{d+O(1)}$ deterministic time.*

Proof. (a) is clear from the definition. For (b), we want a subset $R \subseteq H$ that hits the set $\text{Violate}_p(H)$ for every $p \in \mathbb{R}^d$ of level $> \varepsilon n$, where the *level* of a point p is defined to be $|\text{Violate}_p(H)|$. It suffices to consider just the vertices of the arrangement of H (since for every $p \in \mathbb{R}^d$, there is an equivalent vertex having the same set $\text{Violate}_p(H)$). The number of vertices in the arrangement is $m = O(\binom{n}{d}) = O(n/d)^d$. We can enumerate them trivially in $d^{O(1)} m$ time. Afterwards, we can run the standard greedy algorithm [12] to compute a hitting set for m given sets in a universe of size n in $O(mn)$ time. In our case when all given sets have size $> \varepsilon n$, the greedy algorithm produces a hitting set of size $O(\frac{1}{\varepsilon} \log m) = O(\frac{d}{\varepsilon} \log n)$. □

[1]Throughout the paper, "below" and "above" refer to the d-th coordinate value.

Let $\delta > 0$ be a sufficiently small constant. We begin with a version of Clarkson's first algorithm [10] for linear programming, described in the pseudocode below. (The original version used a random sample R of size near $d\sqrt{n}$, which behaves like a $(1/\sqrt{n})$-net; our version instead uses a $\Theta(1/d^2)$-net R, following Chazelle and Matoušek [8].)

```
LP(H):
0.  if $|H| = O(d^3 \log^c d)$ then return answer directly
1.  compute a $(\delta/d^2)$-net $R$ of $H$
2.  for $j = 1, 2, \ldots$ {
3.      $p = \mathbf{LP}(R \cup X_1 \cup \cdots \cup X_{j-1})$
4.      $X_j = \mathrm{Violate}_p(H)$
5.      if $X_j = \emptyset$ then return $p$
    }
```

Let B^* denote the set of d hyperplanes defining the optimal solution. Observe that in each iteration of the for loop before the last, X_j must include a new member of B^*, because otherwise, p would not be violated by any member of B^* and would thus have a worse objective value than $\mathbf{LP}(B^*) = \mathbf{LP}(H)$, a contradiction. It follows that the loop has at most $d+1$ iterations.

Much of the effort in Chazelle and Matoušek's derandomization [8] is devoted to the construction of the net R in line 1. To this end, they generalized the problem to the construction of *ε-approximations* and described a clever linear-time algorithm [22, 8] that used repeated *merge* and *reduce* steps, with a base case that involved the *method of conditional probabilities* (another alternative is the *bounded independence* technique [17]). Our new idea for line 1 bypasses all this and is a lot more straightforward—we just apply a standard grouping trick, based on Fact 2.2(a), and construct a net for each group naively, as described in lines 1.1–1.3 below. The key observation is that for Clarkson's algorithm, we do not need a net of optimal size—slightly sublinear size is already good enough.

```
1.1.  arbitrarily divide $H$ into groups $H_1, \ldots, H_{\lceil |H|/b \rceil}$ of size at most $b$
1.2.  for $i = 1, \ldots, \lceil |H|/b \rceil$, compute a $(\delta/d^2)$-net $R_i$ of $H_i$
1.3.  $R = \bigcup_i R_i$
```

Assume that there is an algorithm to construct an ε-net for n hyperplanes of size $O(\frac{d}{\varepsilon}\log^c n)$ in $T_{\rm net}(n,\varepsilon)$ deterministic time for some constant c. Let $n = |H|$. Then line 1.2 takes $O(\lceil n/b \rceil \cdot T_{\rm net}(b, \delta/d^2))$ time. The set R in line 1.3 has size $O(\lceil n/b \rceil \cdot d^3 \log^c b)$, which can be made at most $n/(2d) + O(d^3 \log^c d)$ by choosing a group size $b = \Theta(d^4 \log^c d)$. Furthermore, each X_j has size at most $\delta n/d^2$, so $R \cup X_1 \cup \cdots \cup X_{j-1}$ has size at most $n/(2d) + O(d^3 \log^c d) + \delta n/d$. Line 0 takes $O(d^2 \log^c d)^{d/2}$ time by the simplex method. Line 4 takes $O(dn)$ time. Thus, the running time of $\mathbf{LP}(H)$ satisfies the recurrence

$$T_{\rm LP}(n) = \begin{cases} O(d^2 \log^c d)^{d/2} & \text{if } n = O(d^3 \log^c d) \\ (d+1)\,T_{\rm LP}\left((\frac{1}{2}+\delta)\frac{n}{d} + O(d^3 \log^c d)\right) + O(T_{\rm net}(O(d^4 \log^c d), \delta/d^2)\, n) & \\ \quad + O(d^2 n) & \text{else,} \end{cases}$$

implying

$$T_{\rm LP}(n) \;=\; O(T_{\rm net}(O(d^4 \log^c d), \delta/d^2)\, n) \;+\; O(d^2 \log^c d)^{d/2} n.$$

By Fact 2.2(b), $T_{\rm net}(O(d^4 \log^c d), \delta/d^2) = O(d^3 \log^c d)^d$ with $c = 1$. We have therefore obtained a simple deterministic algorithm for linear programming with running time $O(d^3 \log d)^d n$.

3 Clarkson's Second Algorithm

Clarkson [10] gave a second randomized algorithm for linear programming, based on iterative reweighting. Originally the algorithm required $O(d \log n)$ iterations and led to a running time that is slightly superlinear in n. We describe a new variant that requires $O(d \log d)$ iterations and maintains linear dependency on n. The main advantage of the second algorithm is that it works with $\Theta(1/d)$-nets instead of $\Theta(1/d^2)$-nets.

The algorithm is described in the pseudocode below, where initially all elements of H have multiplicity 1. (The main novelty is that when the multiplicity of each element reaches a certain limit, we move the element to a set X. Another change is that instead of a purely iterative algorithm, our variant will continue to use recursion.)

```
LP(H):
 0. if $|H| = O(d^2 \log^c d)$ then return answer directly
 1. repeat {
 2.     compute a $(\delta/d)$-net $R$ of $H$
 3.     $X = \{h \in H : h$ has multiplicity $\geq d^2$ in $H\}$ (with multiplicities reset to 1 in $X$)
 4.     $p = \mathbf{LP}(R \cup X)$
 5.     for each $h \in \text{Violate}_p(H)$ do
 6.         double $h$'s multiplicity in $H$
 7.     if $\text{Violate}_p(H) = \emptyset$ then return $p$
    }
```

Let B^* denote the set of d hyperplanes defining the optimal solution. Observe that in each iteration of the repeat loop before the last, at least one member of B^* has its multiplicity doubled, because otherwise, p would not be violated by any member of B^*, and would thus have a worse objective value than $\mathbf{LP}(B^*) = \mathbf{LP}(H)$. The multiplicity of an element stays unchanged in H when it reaches d^2 or above. Thus, the loop has at most $d \log_2(d^2) + 1 = O(d \log d)$ iterations.

Let $|H|$ denote the size of H, counting multiplicities. Let n denote the initial size of H. In each iteration, $|H|$ increases by at most $\delta|H|/d$. Thus, at the end of the loop, $|H| \leq (1+\delta/d)^{O(d \log d)} n \leq d^{O(\delta)} n$. On the other hand, $|X| \leq |H|/d^2 \leq n/d^{2-O(\delta)}$.

As before, we replace line 2 with:

```
 2.1. arbitrarily divide $H$ into groups $H_1, \ldots, H_{\lceil |H|/b \rceil}$ of size at most $b$
 2.2. for $i = 1, \ldots, \lceil |H|/b \rceil$, compute a $(\delta/d)$-net $R_i$ of $H_i$
 2.3. $R = \bigcup_i R_i$
```

Line 2.2 takes $O(\lceil d^{O(\delta)} n/b \rceil \cdot T_{\text{net}}(b, \delta/d))$ time. The set R in line 2.3 has size $O(\lceil d^{O(\delta)} n/b \rceil \cdot (1/\delta) d^2 \log^c b)$, which can be made at most $n/d^{1+\Omega(\delta)} + O(d^2 \log^c d)$ by choosing a group size $b = d^{3+\Theta(\delta)}$. Line 0 takes $O(d \log^c d)^{d/2}$ time by the simplex method. Lines 5–6 take $O(dn)$ time. Thus, the running time satisfies the recurrence

$$T_{\text{LP}}(n) = \begin{cases} O(d \log^c d)^{d/2} & \text{if } n = O(d^2 \log^c d) \\ O(d \log d)\, T_{\text{LP}}\left(\frac{n}{d^{1+\Omega(\delta)}} + O(d^2 \log^c d)\right) + O(T_{\text{net}}(d^{3+O(\delta)}, \delta/d)\, n) & \\ \quad + O((d^2 \log d)\, n) & \text{else,} \end{cases}$$

implying

$$T_{\mathrm{LP}}(n) \;=\; O(T_{\mathrm{net}}(d^{3+O(\delta)}, \delta/d)\, n) \;+\; O(d \log^c d)^{d/2} n. \tag{1}$$

By Fact 2.2(b), $T_{\mathrm{net}}(d^{3+O(\delta)}, \delta/d) \leq d^{(2+O(\delta))d}$ with $c = 1$. We have therefore obtained an improved simple deterministic algorithm for linear programming with running time $O(d)^{(2+O(\delta))d} n$ for any constant $\delta > 0$. (The small $O(\delta)d$ term in the exponent can probably be improved by using a weighted version of ε-nets, but this will not matter at the end.)

4 New ε-Net Construction

In this section, we provide better bounds for $T_{\mathrm{net}}(n, \varepsilon)$. Two ideas come to mind:

1. we can improve Fact 2.2(b) by enumerating only vertices with level capped at $\lfloor \varepsilon n \rfloor + 1$ rather than all vertices in the entire arrangement;

2. we can compute ε-nets faster by following Brönnimann, Chazelle, and Matoušek's approach of using *sensitive approximations* [5].

Either idea can lead to an improvement, but for our best improvement, we need a combination of both, which creates new technical challenges. Brönnimann, Chazelle, and Matoušek's algorithm for sensitive ε-approximations [5], based on earlier algorithms by Matoušek [22] and Chazelle and Matoušek [8] for ε-approximations, involves repeatedly dividing into groups, and merging and reducing approximations of groups; however, arbitrarily dividing into groups does not preserve caps on levels. We suggest a new solution that adds another recursive call for the division step. (The dependency on n is no longer linear, unlike the previous algorithms [22, 8, 5], but will not matter at the end.)

We need to review technical facts concerning the concept of sensitive ε-approximations; the construction in Fact 4.2(c) below requires the *method of conditional probabilities* [5].

Definition 4.1. Let $\rho_p(H) = |\mathrm{Violate}_p(H)|/|H|$. A subset $R \subseteq H$ is a *sensitive ε-approximation* of H w.r.t. P if for every $p \in P$,

$$|\rho_p(R) - \rho_p(H)| \leq (\varepsilon/2)\sqrt{\rho_p(H)} + \varepsilon^2/2.$$

Fact 4.2. (Brönnimann, Chazelle and Matoušek [5])

(a) (Mergability) *If R_i is a sensitive ε-approximation of H_i w.r.t. P and the H_i's are disjoint and have equal size, then $\bigcup_i R_i$ is a sensitive ε-approximation of $\bigcup_i H_i$ w.r.t. P.*

(b) (Reducibility) *If R is a sensitive ε-approximation of H w.r.t. P and R' is a sensitive ε'-approximation of R w.r.t. P, then R' is a sensitive $(\varepsilon + 2\varepsilon')$-approximation of H w.r.t. P.*

(c) *Given a set H of n hyperplanes in $\mathbb{R}^d$ with $n \geq r \geq \frac{c_0}{\varepsilon^2} \log |P|$ for some constant c_0, we can construct a sensitive ε-approximation of H w.r.t. P of size r in $O(|P|n)$ deterministic time.*

For our purposes of constructing ε-nets, it suffices to construct sensitive approximation w.r.t. points with level bounded by an appropriate cap.

Fact 4.3. *Define* $\mathrm{Cap}(H, \alpha) = \{p \in \mathbb{R}^d : \rho_p(H) \leq \alpha\}$.

(d) *If R is a sensitive $\sqrt{\varepsilon}$-approximation of H w.r.t.* $\mathrm{Cap}(H, \varepsilon + \frac{1}{|H|})$*, then R is an ε-net of H.*

(e) *If R is a sensitive $c\varepsilon$-approximation of H w.r.t.* $\mathrm{Cap}(H, (t\varepsilon)^2)$*, then* $\mathrm{Cap}(H, (t\varepsilon)^2) \subseteq \mathrm{Cap}(R, ((t+c)\varepsilon)^2)$*.*

(f) *Given a set H of n hyperplanes in $\mathbb{R}^d$ with $n \geq r \geq \frac{c_0 d}{\varepsilon^2}\log n$ for some constant c_0, and given a parameter $t \geq 1$, we can construct a sensitive ε-approximation of H w.r.t.* $\mathrm{Cap}(H, (t\varepsilon)^2)$ *of size r in $O(t\varepsilon n/d)^{d+O(1)}$ deterministic time.*

Proof. For (d), observe that for every p with level $\lfloor \varepsilon|H| \rfloor + 1$ (which is in $\mathrm{Cap}(H, \varepsilon + \frac{1}{|H|})$), we have $\rho_p(H) > \varepsilon$, implying that $\rho_p(R) \geq \rho_p(H) - (\sqrt{\varepsilon}/2)\sqrt{\rho_p(H)} - \varepsilon/2 > 0$. This suffices to confirm that R is an ε-net.

For (e), observe that if $\rho_p(H) \leq (t\varepsilon)^2$, then

$$\rho_p(R) \;\leq\; \rho_p(H) + (c\varepsilon/2)\sqrt{\rho_p(H)} + (c\varepsilon)^2/2 \;\leq\; (t^2 + (c/2)t + (c^2/2))\varepsilon^2 \;<\; ((t+c)\varepsilon)^2.$$

For (f), it suffices to run the algorithm in Fact 4.2(c) w.r.t. just the vertices of level at most k in the arrangement of H, with $k = \lceil (t\varepsilon)^2 n \rceil$. Clarkson and Shor [11] (and also Wagner [27]) proved that the number m of vertices of level at most k is $m \leq \binom{n}{\lfloor d/2 \rfloor} \cdot O(k/d+1)^{\lceil d/2 \rceil} = O(n/d)^{\lfloor d/2 \rfloor} \cdot O(t^2\varepsilon^2 n/d+1)^{\lceil d/2 \rceil} = O(t\varepsilon n/d)^{d+O(1)}$. We can enumerate all vertices of level at most k by an output-sensitive algorithm that uses repeated ray shooting [2, 1] in $O(d^{O(1)}mn)$ time (this can viewed as a generalization of the standard gift-wrapping algorithm for convex hulls [6] in dual space; the time bound gets better in small constant dimensions by using ray shooting data structures [2, 1], but this will not matter). Afterwards, we can run the algorithm in Fact 4.2(c) in $O(mn)$ time. ◻

Lemma 4.4. *Given a set H of n hyperplanes in $\mathbb{R}^d$ with n a power of 2 and $n \geq \frac{c_1 d}{\varepsilon^2}\log\frac{d}{\varepsilon}$ for some constant c_1, and given parameters $\varepsilon > 0$ and $t \geq 1$, we can compute a sensitive ε-approximation of size exactly $n/2$ w.r.t.* $\mathrm{Cap}(H, (t\varepsilon)^2)$ *in deterministic time*

$$T_{\mathrm{halver}}(n, \varepsilon, t) \;=\; O(\tfrac{t+\log n}{\varepsilon}\log\tfrac{d}{\varepsilon})^d n^{1.59}.$$

Proof. We describe our algorithm by the following deceptively concise pseudocode using recursion:

```
Halver($H, \varepsilon, t$):
  0.  if $|H| = O(\frac{d}{\varepsilon^2}\log\frac{d}{\varepsilon})$ or $\varepsilon \geq 2$ then return answer directly
  1.  $A$ = Halver($H, 2\varepsilon, \lceil t/2 \rceil$)
  2.  return Halver($A, \varepsilon, t+2$) $\cup$ Halver($H - A, \varepsilon, t+2$)
```

Since A is a sensitive 2ε-approximation of H w.r.t. $\mathrm{Cap}(H, (t\varepsilon)^2)$, from the definitions it follows that $H - A$ is also a sensitive 2ε-approximation (since $|H - A| = |A| = |H|/2$ implies that $\rho_p(H - A) - \rho_p(H) = \rho_p(H) - \rho_p(A)$). We thus have $\mathrm{Cap}(H, (t\varepsilon)^2) \subseteq \mathrm{Cap}(A, ((t+2)\varepsilon)^2), \mathrm{Cap}(H - A, ((t+2)\varepsilon)^2)$ by Fact 4.3(e). By induction hypothesis and Fact 4.2(a), the union in line 2 is a sensitive ε-approximation of H w.r.t. $\mathrm{Cap}(H, (t\varepsilon)^2)$. (To recap, the desired sensitive ε-approximation is computed by the recursive calls in line 2; the only purpose of the recursive call in line 1 is in providing a good cap for line 2.)

By Fact 4.3(f), line 0 takes $O(\frac{t}{\varepsilon}\log\frac{d}{\varepsilon})^d$ time. Assuming that **Halver** returns linked lists for both the sensitive approximation and its complement, we see that cost of the algorithm excluding the three recursive calls in lines 1–2 is $O(1)$ by merging linked lists. The running time of **Halver**(H, ε, t) satisfies the recurrence

$$T_{\text{halver}}(n, \varepsilon, t) \;=\; \begin{cases} O(\frac{t}{\varepsilon}\log\frac{d}{\varepsilon})^d & \text{if } n = O(\frac{d}{\varepsilon^2}\log\frac{d}{\varepsilon}) \\ T_{\text{halver}}(n, 2\varepsilon, \lceil t/2\rceil) + 2\,T_{\text{halver}}(n/2, \varepsilon, t+2) + O(1) & \text{else.} \end{cases}$$

Observe that the depth of the recursion is at most $\log_2 n + \log_2(1/\varepsilon)$, since n is either halved or ε is doubled in each recursive call. The number of nodes in the recursion tree is thus at most $3^{\log_2 n + \log_2(1/\varepsilon)} = O(n/\varepsilon)^{1.59}$. The parameter t increases by at most $O(\log n)$ and the parameter ε does not decrease. It follows that $T_{\text{halver}}(n, \varepsilon, t) \le O(\frac{t+\log n}{\varepsilon}\log\frac{d}{\varepsilon})^d n^{1.59}$. (The $n^{1.59}$ factor is improvable but will not matter at the end.) ☐

Theorem 4.5. *Given a set H of n hyperplanes in $\mathbb{R}^d$ with n a power of 2, and given parameters $\varepsilon > 0$ and $t \ge 1$, for some constant c_1, we can compute a sensitive $\varepsilon\log n$-approximation of size $\left\lceil \frac{c_1 d}{\varepsilon^2}\log\frac{d}{\varepsilon}\right\rceil$ w.r.t.* $\mathrm{Cap}(H, (t\varepsilon)^2)$ *in deterministic time*

$$\widetilde{T}_{\text{approx}}(n, \varepsilon, t) \;=\; O(\tfrac{t+\log n}{\varepsilon}\log\tfrac{d}{\varepsilon})^d n^{1.59}.$$

Proof. We describe another recursive algorithm:

```
**Approx**$(H, \varepsilon, t)$:
  0.  if $|H| = O(\frac{d}{\varepsilon^2}\log\frac{d}{\varepsilon})$ then return answer directly
  1.  $A = $ **Halver**$(H, \varepsilon, t)$
  2.  $R = $ **Approx**$(A, \varepsilon, t+1) \cup$ **Approx**$(H - A, \varepsilon, t+1)$
  3.  return **Halver**$(R, \varepsilon/2, t + \log n)$
```

Since both A and $H - A$ are sensitive ε-approximations of H w.r.t. $\mathrm{Cap}(H, (t\varepsilon)^2)$, we have $\mathrm{Cap}(H, (t\varepsilon)^2) \subseteq \mathrm{Cap}(A, ((t+1)\varepsilon)^2), \mathrm{Cap}(H - A, ((t+1)\varepsilon)^2)$ by Fact 4.3(e). By induction hypothesis and Fact 4.2(a), the union R in line 2 is a sensitive $\varepsilon\log(n/2)$-approximation of H w.r.t. $\mathrm{Cap}(H, (t\varepsilon)^2)$. We have $\mathrm{Cap}(H, (t\varepsilon)^2) \subseteq \mathrm{Cap}(R, ((t + \log n)\varepsilon)^2)$ by Fact 4.3(e). By Fact 4.2(b), the set in line 3 is a sensitive $(\varepsilon\log(n/2) + 2(\varepsilon/2) = \varepsilon\log n)$-approximation of H w.r.t. $\mathrm{Cap}(H, (t\varepsilon)^2)$.

By Fact 4.3(f), line 0 takes $O(\frac{t}{\varepsilon}\log\frac{d}{\varepsilon})^d$ time. By Lemma 4.4, lines 1 and 3 take $O(\frac{t+\log n}{\varepsilon}\log\frac{d}{\varepsilon})^d n^{1.59}$ time. The running time of **Approx**(H, ε, t) satisfies the recurrence

$$\widetilde{T}_{\text{approx}}(n, \varepsilon, t) \;=\; \begin{cases} O(\frac{t}{\varepsilon}\log\frac{d}{\varepsilon})^d & \text{if } n = O(\frac{d}{\varepsilon^2}\log\frac{d}{\varepsilon}) \\ 2\,\widetilde{T}_{\text{approx}}(n/2, \varepsilon, t+1) + O(\frac{t+\log n}{\varepsilon}\log\frac{d}{\varepsilon})^d n^{1.59} & \text{else.} \end{cases}$$

It follows that $\widetilde{T}_{\text{approx}}(n, \varepsilon, t) = O(\frac{t+\log n}{\varepsilon}\log\frac{d}{\varepsilon})^d n^{1.59}$. ☐

Corollary 4.6. *Given a set H of n hyperplanes in $\mathbb{R}^d$ with n a power of 2, and parameters $\varepsilon > 0$ and $t \ge 1$, we can compute a sensitive ε-approximation of size $O(\frac{d}{\varepsilon^2}\log^2 n\log\frac{d}{\varepsilon})$ w.r.t.* $\mathrm{Cap}(H, (t\varepsilon)^2)$ *in deterministic time*

$$T_{\text{approx}}(n, \varepsilon, t) \;\le\; \widetilde{T}_{\text{approx}}(n, \varepsilon/\log n, t\log n) \;=\; O(\tfrac{t}{\varepsilon}\log^2 n\log\tfrac{d}{\varepsilon})^d n^{1.59}.$$

By Fact 4.3(d), we conclude:

Corollary 4.7. *Given a set H of n hyperplanes in $\mathbb{R}^d$ with n a power of 2, and parameter $\varepsilon > 0$, we can compute an ε-net of size $O(\frac{d}{\varepsilon}\log^2 n \log\frac{d}{\varepsilon})$ in deterministic time*

$$T_{\rm net}(n,\varepsilon) \;\le\; T_{\rm approx}(n,\sqrt{\varepsilon},\sqrt{1+\tfrac{1}{\varepsilon n}}) \;=\; O(\tfrac{1}{\varepsilon})^{d/2}(\log^2 n\log\tfrac{d}{\varepsilon})^d n^{1.59}.$$

Note the extra logarithmic factors on the net size in the above corollaries (which are improvable if we increase the running time, but will not matter at the end).

Combining with the approach in Section 3 and applying Corollary 4.7 to (1) with $c = 3$, we have finally obtained a deterministic algorithm for linear programming with running time $O(d)^{d/2}(\log d)^{3d}n$.

5 Remarks

The same results hold for the problem of computing the smallest enclosing ball of n points in $\mathbb{R}^d$; this problem can be transformed to minimizing a convex function in an intersection of n halfspaces in one dimension higher.

Like Chazelle and Matoušek's derandomization [8] of Clarkson's algorithm, our algorithms in Sections 2 and 3 can more generally solve any *LP-type problem* [23] of combinatorial dimension d, assuming just the availability of a *subsystem oracle* (see [8] for the formal definitions). The improvement in Section 4 does not completely generalize, because the same combinatorial bound on "($\le k$)-levels" does not necessarily hold. However, sensitive approximations can still be used and we can still obtain a $d^{(1+o(1))d}n$ deterministic time bound as follows: First, in Fact 4.3(f), the time bound without caps is $O(n/d)^{d+O(1)}$. In Theorem 4.5, we can apply a more naive divide-and-conquer where in line 1 we simply choose an arbitrary subset A of size $|H|/2$. The recurrence becomes

$$\widetilde{T}_{\rm approx}(n,\varepsilon) \;=\; \begin{cases} O(\frac{1}{\varepsilon^2}\log\frac{d}{\varepsilon})^d & \text{if } n = O(\frac{d}{\varepsilon^2}\log\frac{d}{\varepsilon}) \\ 2\,\widetilde{T}_{\rm approx}(n/2,\varepsilon) + O(\frac{1}{\varepsilon^2}\log\frac{d}{\varepsilon})^d & \text{else,} \end{cases}$$

yielding $\widetilde{T}_{\rm approx}(n,\varepsilon) = O(\frac{1}{\varepsilon^2}\log\frac{d}{\varepsilon})^d n$. In Corollaries 4.6 and 4.7, we then obtain running times of $T_{\rm approx}(n,\varepsilon) = O(\frac{1}{\varepsilon^2}\log n\log\frac{d}{\varepsilon})^d n$ and $T_{\rm net}(n,\varepsilon) = O(\frac{1}{\varepsilon}\log n\log\frac{d}{\varepsilon})^d n$, implying a final time bound of $T_{\rm LP}(n) = O(d\log^2 d)^d n$ for LP-type problems.

Acknowledgement. I would like to dedicate this work to Jirí Matoušek in memoriam.

References

[1] P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. *SIAM Journal on Computing*, 22(4):794–806, 1993.

[2] P. K. Agarwal and J. Matoušek. Dynamic half-space range reporting and its applications. *Algorithmica*, 13(4):325–345, 1995.

[3] P. K. Agarwal and M. Sharir. Efficient algorithms for geometric optimization. *ACM Computing Surveys*, 30(4):412–458, 1998.

[4] P. K. Agarwal, M. Sharir, and S. Toledo. An efficient multi-dimensional searching technique and its applications. Technical Report CS-1993-20, Department of Computer Science, Duke University, 1993.

[5] H. Brönnimann, B. Chazelle, and J. Matoušek. Product range spaces, sensitive sampling, and derandomization. *SIAM Journal on Computing*, 28(5):1552–1575, 1999.

[6] D. R. Chand and S. S. Kapur. An algorithm for convex polytopes. *Journal of the ACM*, 17(1):78–86, 1970.

[7] B. Chazelle. *The Discrepancy Method: Randomness and Complexity.* Cambridge University Press, 2001.

[8] B. Chazelle and J. Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *Journal of Algorithms*, 21(3):579–597, 1996.

[9] K. L. Clarkson. Linear programming in $O(n3^{d^2})$ time. *Information Processing Letters*, 22(1):21–24, 1986.

[10] K. L. Clarkson. Las Vegas algorithms for linear and integer programming when the dimension is small. *Journal of the ACM*, 42(2):488–499, 1995.

[11] K. L. Clarkson and P. W. Shor. Application of random sampling in computational geometry, II. *Discrete and Computational Geometry*, 4:387–421, 1989.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, Cambridge, MA, third edition, 2001.

[13] M. Dyer, N. Megiddo, and E. Welzl. Linear programming in low dimensions. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 45. CRC Press LLC, Boca Raton, FL, second edition, 2004.

[14] M. E. Dyer. On a multidimensional search technique and its application to the Euclidean one-centre problem. *SIAM Journal on Computing*, 15(3):725–738, 1986.

[15] M. E. Dyer and A. M. Frieze. A randomized algorithm for fixed-dimensional linear programming. *Mathematical Programming*, 44(1-3):203–212, 1989.

[16] M. H. Goldwasser. A survey of linear programming in randomized subexponential time. *SIGACT News*, 26:96–104, 1995.

[17] M. T. Goodrich and E. A. Ramos. Bounded-independence derandomization of geometric partitioning with applications to parallel fixed-dimensional linear programming. *Discrete and Computational Geometry*, 18(4):397–420, 1997.

[18] T. D. Hansen and U. Zwick. An improved version of the random facet pivioting rule for the simplex algorithm. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing*, pages 209–218, 2015.

[19] G. Kalai. A subexponential randomized simplex algorithm. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 475–482, 1992.

[20] Y. T. Lee and A. Sidford. Efficient inverse maintenance and faster algorithms for linear programming. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science*, pages 230–249, 2015.

[21] J. Matoušek. Derandomization in computational geometry. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 559–595. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[22] J. Matoušek. Approximations and optimal geometric divide-and-conquer. *Journal of Computer and System Sciences*, 50(2):203–208, 1995.

[23] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4/5):498–516, 1996.

[24] N. Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM*, 31(1):114–127, 1984.

[25] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, New York, NY, 1995.

[26] R. Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete and Computational Geometry*, 6:423–434, 1991.

[27] U. Wagner. On a geometric generalization of the Upper Bound Theorem. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 635–645, 2006.