# Chapter 3

# Weighted sampling

## 3.1 Linear Programming

In the last lecture we used theory of linear programming as a device for understanding randomized and distributional complexity. Of course, linear programming is also an algorithmic primitive. Here is a brief recap. The basic problem is to

$$\begin{array}{ll} \text{minimize } c^* x & \text{here } c, x \in \mathbb{R}^d \\ \text{subject to } Ax \geq b & \text{here } b \in \mathbb{R}^n, A \in \mathbb{R}^{n \times d} \end{array}$$

Each row $A_{i*}x \geq b_i$ is a "constraint" $H_i$, and we'll call the hyperplane corresponding to it $h_i$.

The most popular algorithm for this problem is Simplex (Dantzig [13, 15]), although its worst-case runtime $T_{\text{Simplex},n,d}$ is quite bad. Still, for present purposes, it won't matter much whether we use Simplex or an asymptotically better algorithm, because we'll use it only for a small number of constraints. (E.g., there is an upper bound $T_{\text{Simplex},9d^2,d} \in (c_5 d)^{d/2}$ for some constant $c_5$.) Famously Khachiyan [34, 35, 20] showed the ellipsoid algorithm solves LP in polynomial time. Subsequent interior-point methods [32, 48, 58] brought the runtime down to $O(n^{1.5}dL)$ arithmetic operations for inputs with $L$-bit numbers, and there have even been some slight improvements since then [38].

Two comments on these runtimes:

(1) We'll assume all arithmetic operations are unit time (so we don't need to keep track of the parameter $L$); since the dependence on $L$ is common to all algorithms, this won't affect our qualitative conclusions.

(2) The bounds can be asymmetric in the dimension and the number of constraints because we do not assume the variables in the LP are nonnegative; thus, the dual LP may have more than $d$, likely as many as $n + d$, constraints. So you can't get around the fact that the above runtimes are superlinear in the number of constraints $n$ by going to the dual LP.

One situation in which the dependence on $n$ can be made much better is when $d$ is small compared with $n$. For this purpose, we will use the above-mentioned algorithms only as a base-case. We write $t_d$ for the time for solving an LP with $9d^2$ constraints in $d$ dimensions, using our favorite algorithm "B". (So, we could use B = ellipsoid or interior-point methods to get a polynomial bound in $d$, or B = Simplex to get an exponential bound in $d$.)

Megiddo [40] showed it was possible to achieve time linear in $n$ in fixed dimension. This low-dimensional case is what we study now. It is useful before beginning to make the following simplifying assumptions.

(a) The LP is feasible and has bounded feasible region. (And let $\mathcal{B}$ be a bounding box large enough to be guaranteed to contain the feasible region.)

(b) The LP is in general position in the following senses:

(b1) The objective function $c$ is not equal to any of the vectors $A_i$.

(b2) Consider any $d$ vectors chosen from $\{c, A_1, \dots, A_n\}$. Unless there are any duplicates in the list, they span $\mathbb{R}^d$.

(b3) If the hyperplanes corresponding to two different sets of constraints, $R_1$ and $R_2$, each intersect in a point, and those two points are different, then they also differ in the value of the objective function.

(These assumptions do not affect the difficulty of the problem. For (a), one can put in an extra slack variable on all constraints; there are also known, polynomial bounds on the necessary size for the bounding box. For (b), one can infinitesimally perturb the LP.)

Given a subset $S \subseteq [n]$, let $\text{Val}(S)$ be the least value of the objective function consistent with these constraints. By the assumptions (b), this value is achieved at a unique point, which is a vertex of the feasible polytope, call it $X(S)$.

Since we allow duplicates of a constraint $H_i$, and we denote the set of these duplicates by $\tilde{H}_i$, the "type" of constraint $H_i$. So we may for $i \neq j$ have $\tilde{H}_i = \tilde{H}_j$, if $H_j \in \tilde{H}_i$. $\tilde{h}_i$ is the common bounding hyperplane.

Let $T(S)$ be the tight types of constraints at $X(S)$, that is, those $\tilde{H}_i$ for which $X(S) \subseteq \tilde{h}_i$; observe that $|T(S)| = d$ and $X(S) = \cap_{i \in T(S)} \tilde{h}_i$.

For $\tilde{H}_i \in T(S)$ let $v_i$ be the normal to $\tilde{h}_i$. Here is a simple and very useful observation.

**Lemma 19.** *Select a constraint $H_i$ uniformly at random.* $\Pr(\text{Val}([n] - \{i\}) < \text{Val}([n])) \leq \frac{d}{n}$.

*Proof.* If removing $H_i$ from the list of constraints changes the value of the objective function, then $H_i$ is in $T([n])$ and is the sole element in $\tilde{H}_i$. There can be at most $d$ such. □

We'll now discuss some algorithms that take advantage of Lemma 19 to get around the naïve idea of "examining all $\binom{n}{d}$ sets of constraints." These algorithms achieve linear dependence on $n$ in the runtime. The first is Seidel's algorithm [53]: If $n = d$ this is just linear algebra which we can solve in time $d^a$ for some $1 \leq a < \infty$. If $d = 1$ simply compute min. Otherwise, pick a constraint $H_i$ uniformly at random, and solve recursively for $X([n] - \{i\})$. Check if this point violates $H_i$ (which takes time $d$); if it does, solve the LP restricted to the $(d-1)$-dimensional subspace $h_i$. Let $T_{n,d}$ be the expected runtime.

*Claim:* $T_{n,d} \in O(d^a d! n)$. *Proof* by induction on $n$ and $d$.

Basecases: $d = 1$ or $n = d$. Otherwise $n > d > 1$, induct on $n$: by the lemma,

$$\begin{aligned}
T_{n,d} &\leq d^a + T_{n-1,d} + \frac{d}{n} T_{n-1,d-1} \\
&\leq d^a(1 + d!(n-1) + \frac{d}{n}(d-1)^a(d-1)!(n-1)) \quad \text{by induction} \\
&\leq d^a(d!n + ((1-1/d)^a + \frac{1}{d!} - 1)d!) \\
&\leq d^a d! n \quad \text{for all } d \geq 1
\end{aligned}$$

□

Better dependence on $d$ was achieved by Clarkson [12]. He actually gave two slightly different methods, one recursive and one iterative; and the best runtime is achieved by combining them in a "mixed" algorithm. We'll start with the recursive method, but before that, a simple:

**Corollary 20.** *Pick a set $R$ of $r$ constraints uar without repetition. The expected number of remaining constraints which are violated by $X(R)$ is $\leq \frac{d(n-r)}{r+1}$.*

(Lemma 19 is the special case $r = n - 1$.)

*Proof.* An equivalent way of asking this question is, if I pick $R$ uar without repetition, and then I pick another constraint $H_i$ outside $R$, what is the probability $p$ that $H_i$ is violated by $X(R)$? Then (by linearity of expectation) the expected number of violations is $(n-r)p$.

We can equivalently perform this experiment by selecting the set $R' = R \cup \{H_i\}$ uar without repetition, then uniformly selecting $H_i$ from within $R'$. But, once we've picked $R'$, we can apply Lemma 19, and conclude that $p \leq \frac{d}{r+1}$ regardless of what $R'$ is. □

**Algorithm 1:** $\text{Clarkson}_{\text{Rec}}(S)$

```
Input: Set of linear constraints $S = \{H_i\}$
Output: Optimum point $X(S)$
$V^* := \emptyset$
$n := S$
$r := d\sqrt{n}$
If $|S| \leq 9d^2$ return $X(S) := B(S)$; Otherwise
repeat
    Choose $R \subseteq S - V^*$ of size $r$, uar
    $x := \text{Clarkson}_{\text{Rec}}(R \cup V^*)$
        Comments: (1) If $R \cup V^*$ has unbounded minimum, set $x$ to be a minimizing point in $\mathcal{B}$.
                  (2) We will show that the size of $R \cup V^*$ will always be $\leq 2d\sqrt{n}$.
    $V := \{H \in S : x \text{ violates } H\}$
    If $|V| \leq 2\sqrt{n}$ then $V^* := V^* \cup V$
until $V = \emptyset$
Return $x$
```

Toward analyzing the first algorithm $\text{Clarkson}_{\text{Rec}}$, let's call an execution of the loop "Augmenting" if $0 < |V| \leq 2\sqrt{n}$. From what we proved earlier, $E(|V|) = \frac{d(n-|V^*|-r)}{r+1} \leq \frac{dn}{r} = \sqrt{n}$ and therefore by the Markov inequality, $\Pr(\text{Augmenting}) \geq 1/2$.

**Lemma 21.** *If $V \neq \emptyset$ then $V$ contains at least one constraint from $T(S)$.*

*Proof.* If $V$ contains no such, then $X(R \cup V^*) = X(R \cup V^* \cup T(S)) = X(S)$ so $V = \emptyset$. □

Thus there can be at most $d$ augmenting rounds.

Thus we have several conclusions: (a) $V^*$ can grow to only $2d\sqrt{n}$. (b) Therefore, in every round of the execution, $|R \cup V^*| \leq 3d\sqrt{n}$. (c) The expected number of iterations of the loop is $\leq 2d$; more exactly, the number of iterations is distributed as the time to toss $d$ "Heads" using coins whose probability of coming up Heads is at least $1/2$. So we have the following recurrence for the expected runtime:

$$T_{\text{Rec},n,d} \leq \begin{cases} t_d & \text{if } m \leq 9d^2 \\ 2d(nd + T_{\text{Rec},3d\sqrt{n},d}) & \text{o'wise (the } nd \text{ is because checking each } H_i \text{ takes time } d) \end{cases}$$

Notice that the number of constraints goes down dramatically in the recursion; $\log_d(d\sqrt{n}) = 1 + \frac{1}{2}\log_d n$, so only $\lg(\log_d n) + O(1)$ levels are required until $n \leq 9d^2$. (The $O(1)$ is because the halving slows down a bit at the end.) This suggests the guess

**Theorem 22.** $T_{Rec,n,d} \leq c_1 nd^2 + d^{c_3 \lg \log_d n} t_d = c_1 nd^2 + (\log_d n)^{c_3 (\lg e) \log d} t_d$.

*Proof.* Verify the recurrence, for $n > 9d^2$:

$$\begin{aligned}
\text{RHS of recurrence} &= 2d(nd + 3c_1 d^3 \sqrt{n} + d^{c_3 \lg \log_d (3d\sqrt{n})} t_d) \\
&\leq 2nd^2(1 + o(1)) + d^{\log_d 2 + 1 + c_3 \lg \log_d (3d\sqrt{n})} t_d \\
\text{for } c_1, c_3 \text{ sufficiently large:} &\leq c_1 nd^2 + d^{c_3 \lg \frac{3}{2} \log_d (3d\sqrt{n})} t_d \\
&\leq c_1 nd^2 + d^{c_3 \lg \log_d (3d^{3/2} n^{3/4})} t_d \\
&\leq c_1 nd^2 + d^{c_3 \lg \log_d n} t_d = \text{LHS.}
\end{aligned}$$

□

In the same paper Clarkson gave a related "iterative reweighting" algorithm $\text{Clarkson}_{\text{Iter}}$. This can be viewed an instance of importance sampling, a widely applicable method. We'll see it shortly in MWU, later in # DNF and network reliability; it shows up in clustering; and it continues to play a role in current research (e.g., in "twice Ramanujan sparsifiers"). $\text{Clarkson}_{\text{Iter}}$ uses sampling weights $w(H)$ on the constraints. For a set $R \subseteq S$ of constraints, set $w(R) = \sum_{H \in R} w(H)$.

**Algorithm 2:** $\text{Clarkson}_{\text{Iter}}(S)$

```
Input: Set of linear constraints $S$
Output: Optimum point $X(S)$
$n := S$
$r := 9d^2 - 1$
For all $H \in S$ set $w(H) := 1$
If $|S| \leq r$ return $X(S) := B(S)$; Otherwise
repeat
    Sample according to weights $w$, with repetition, a multiset $R \subseteq S$ of size $r$
    $x := B(R)$
    $V := \{H \in S : x \text{ violates } H\}$
    If $w(V) \leq \frac{2w(S)}{9d}$ then double $w(H)$ for all $H \in V$
until $V = \emptyset$
Return $x$
```

Let us clarify the sampling method: $R$ will be formed as a list $H_{a_1}, \ldots, H_{a_r}$. Each index $a_j$ is chosen independently by $\Pr(a_j = i) = \frac{w(H_i)}{w(S)}$. Obviously constraints can repeat in this list, even if $S$ has no repeated constraints. Analogously to before, we have a notion of an augmenting round: a round is augmenting if we increase weights in it.

**Lemma 23.** *The expected weight of the violating set $V$ in the loop is at most $\frac{w(S)d}{r+1} = \frac{w(S)}{9d}$.*

*Proof.* First note that *if* we had been sampling without weights and without repetition, this would have followed from Cor. 20, because that bound is $\frac{d(n-9d^2)}{9d^2} < \frac{n}{9d}$. The matter of weights does not matter however, since the weights (which in this algorithm always remain integer) are equivalent to duplicating constraints (which was allowed in Cor. 20). How do we deal with the "with repetition"? Well, given weights $w(H_i)$, consider any $N \in \mathbb{N}_+$, and duplicate each original constraint $H_i$, $Nw(H_i)$ times. This gives a new collection of $Nw(S)$ (highly redundant, but unweighted) constraints. Let $A_{N,\text{without}}$ be the rv which is $\frac{1}{N}\times$ the number of violated constraints when we sample

$R$ without repetition from this redundant collection. The analysis in the previous paragraph applies to $A_{N,\text{without}}$, telling us that $E(A_{N,\text{without}}) \leq \frac{w(S)}{9d}$.

Let $A_{\text{with}}$ be the rv which is the total weight of violated constraints when we sample $R$ as in the algorithm (i.e., with repetition and not having scaled the collection by $N$).

The with-repetition process of the algorithm is identical to the same with-repetition process run on the redundant (but unweighted) collection; in particular, if we define $A_{N,\text{with}}$ to be the rv which is $\frac{1}{N}\times$ the number of violated constraints when we sample $R$ with repetition from the redundant collection, then the distribution of $A_{N,\text{with}}$ is the same as that of $A_{\text{with}}$.

In the limit of large $N$, the probability in the with-repetition process of having any repetitions, tends to 0. Moreover: (a) The rv $A_{N,\text{without}}$ is bounded (by $w(S)$). (b) The rv $A_{\text{with}}$ is bounded (also by $w(S)$). Therefore,

$$\lim_{N\to\infty} E(A_{N,\text{without}}) = E(A_{\text{with}}).$$

We have already seen that for all $N$ the LHS is upper bounded by $\frac{w(S)}{9d}$, therefore the same upper bound applies to the RHS. □

By the Markov inequality we now have:

**Corollary 24.** *With probability at least* $1/2$ *a loop is either augmenting, or has* $V = \emptyset$.

Using Lemma 21, if a loop is augmenting then $V$ intersects $T(S)$.

After $m$ augmenting rounds, the weight of $T(S)$ must therefore have increased to at least $d2^{m/d}$. (It's easy to see that the least increase is achieved by hitting each of the constraints in $T(S)$ equally often.) On the other hand, $w(S)$ can have increased to only $n(1+\frac{2}{9d})^m$.

Therefore, after $m$ is big enough to imply $d2^{m/d} \geq n(1+\frac{2}{9d})^m$, in particular after $m = O(d\log n)$, there can be no more augmenting loops. What then? That doesn't mean there aren't more loops; we might keep picking sets $R$ and not augmenting because $V$ is too heavy. However, the corollary remains, so what this means is that from that point on, each loop is with probability $\geq 1/2$ the last. So the expected number of loops to termination is $O(d\log n)$. Each loop takes time $t_d + O(nd)$.

This gives a runtime with a term of order $d^2 n\log n$, specifically, a bound of

**Theorem 25.** $T_{Iter,n,d} \leq (t_d + O(nd))d\log n$.

This is better than the recursive algorithm in terms of $d$, but doesn't dominate the recursive algorithm because of the $\log n$. But the following slight variant does dominate both algorithms: run the recursive algorithm only as the top level procedure call, and use calls to the iterative algorithm within it. Here is the pseudocode; it is identical to Clarkson$_{\text{Rec}}$ except that the internal procedure call is to Iter rather than Rec.

As proven before for the recursive algorithm, the expected number of loops here is $O(d)$, so

$$\begin{aligned} T_{\text{Mixed},n,d} &\leq O(d)(nd + T_{\text{Iter},3d\sqrt{n},d}) \\ &\leq O(nd^2 + d(t_d + O(d^2\sqrt{n}))d\log(3d\sqrt{n})) \\ &= O(nd^2 + d^4\sqrt{n}\log(d\sqrt{n}) + t_d\log(d\sqrt{n})). \end{aligned}$$

The middle term is always dominated by one of the others, and the $\log d$'s can be neglected, so

**Theorem 26.** $T_{Mixed,n,d} \leq O(nd^2 + t_d\log n)$.

```
**Algorithm 3:** $\text{Clarkson}_{\text{Mixed}}(S)$
**Input:** Set of linear constraints $S$
**Output:** Optimum point $X(S)$
$V^* := \emptyset$
$n := S$
$r := d\sqrt{n}$
If $|S| \leq 9d^2$ return $X(S) := B(S)$; Otherwise
**repeat**
    Choose $R \subseteq S - V^*$ of size $r$, uar
    $x := \text{Clarkson}_{\text{Iter}}(R \cup V^*)$
    $V := \{H \in S : x \text{ violates } H\}$
    If $|V| \leq 2\sqrt{n}$ then $V^* := V^* \cup V$
**until** $V = \emptyset$
Return $x$
```

Incidentally, another way of proving this (which is what I did in class) is to prove Lemma 19/Cor. 20 only for constraints in general position and without duplications; then for the with-repetition sampling, argue in Lemma 23 that the weight of a violated set (in the set system with duplicates) is less than it would be after infinitesimally perturbing those duplicates so that they satisfy the requirements of Lemma 19/Cor. 20 .