Chapter 8

Linear Programming in Low Dimensions

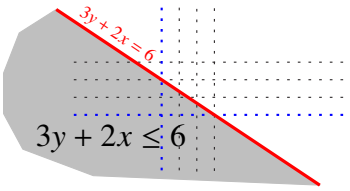
By Sariel Har-Peled, February 8, 2023[①]

> At the sight of the still intact city, he remembered his great international precursors and set the whole place on fire with his artillery in order that those who came after him might work off their excess energies in rebuilding.
>
> The tin drum, Gunter Grass

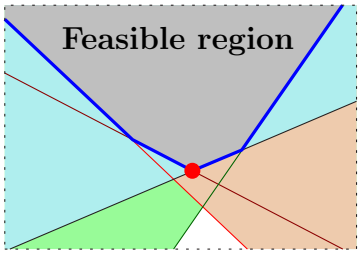
In this chapter, we shortly describe (and analyze) a simple randomized algorithm for linear programming in low dimensions. Next, we show how to extend this algorithm to solve linear programming with violations. Finally, we will show how one can efficiently approximate the number constraints that one needs to violate to make a linear program feasible. This serves as a fruitful ground to demonstrate some of the techniques we visited already. Our discussion is going to be somewhat intuitive – it can be made more formal with more work.

8.1. Linear programming

Assume we are given a set of n linear inequalities of the form $a_1x_1 + \cdots + a_dx_d \leq b$, where $a_1, \ldots, a_d, b$ are constants and $x_1, \ldots, x_d$ are the variables. In the ***linear programming*** (LP) problem, one has to find a ***feasible solution***, that is, a point $(x_1, \ldots, x_d)$ for which all the linear inequalities hold. In the following, we use the shorthand ***LPI*** to stand for ***linear programming instance***. Usually we would like to find a feasible point that maximizes a linear expression (referred to as the ***target function*** of the given LPI) of the form $c_1x_1 + \cdots + c_dx_d$, where $c_1, \ldots, c_d$ are prespecified constants.



The set of points complying with a linear inequality $a_1x_1+\cdots+a_dx_d \leq b$ is a halfspace of $\mathbb{R}^d$ having the hyperplane $a_1x_1 + \cdots + a_dx_d = b$ as a boundary; see the figure on the right. As such, the feasible region of a LPI is the intersection of n halfspaces; that is, it is a ***polyhedron***. If the polyhedron is bounded, then it is a ***polytope***. The linear target function is no more than specifying a direction, such that we need to find the point inside the polyhedron which is extreme in this direction. If the polyhedron is unbounded in this direction, the optimal solution is ***unbounded***.



For the sake of simplicity of exposition, it will be easier to think of the direction for which one has to optimize as the negative x_d-axis direction. This can be easily realized by rotating the space such that the required direction is pointing downward. Since the feasible region is the intersection of convex sets (i.e., halfspaces), it is convex. As such, one can imagine the boundary of the feasible region as a vessel (with a convex interior). Next, we release a ball at the top of the vessel, and the ball rolls

down (by "gravity" in the direction of the negative x_d-axis) till it reaches the lowest point in the vessel and gets "stuck". This point is the optimal solution to the LPI that we are interested in computing.

In the following, we will assume that the given LPI is in general position. Namely, if we intersect k hyperplanes, induced by k inequalities in the given LPI (the hyperplanes are the result of taking each of this inequalities as an equality), then their intersection is a $(d-k)$-dimensional affine subspace. In particular, the intersection of d of them is a point (referred to as a ***vertex***). Similarly, the intersection of any $d+1$ of them is empty.

A polyhedron defined by an LPI with n constraints might have $O\left(n^{\lfloor d/2 \rfloor}\right)$ vertices on its boundary (this is known as the upper-bound theorem [**g-cp-03**]). As we argue below, the optimal solution is a vertex. As such, a naive algorithm would enumerate all relevant vertices (this is a non-trivial undertaking) and return the best possible vertex. Surprisingly, in low dimension, one can do much better and get an algorithm with linear running time.

We are interested in the best vertex of the feasible region, while this polyhedron is defined implicitly as the intersection of halfspaces, and this hints to the quandary that we are in: We are looking for an optimal vertex in a large graph that is defined implicitly. Intuitively, this is why proving the correctness of the algorithms we present here is a non-trivial undertaking (as already mentioned, we will prove correctness in the next chapter).

8.1.1. A solution and how to verify it

Observe that an optimal solution of an LPI is either a vertex or unbounded. Indeed, if the optimal solution p lies in the middle of a segment s, such that s is feasible, then either one of its endpoints provides a better solution (i.e., one of them is lower in the x_d-direction than p) or both endpoints of s have the same target value. But then, we can move the solution to one of the endpoints of s. In particular, if the solution lies on a k-dimensional facet F of the boundary of the feasible polyhedron (i.e., formally F is a set with affine dimension k formed by the intersection of the boundary of the polyhedron with a hyperplane), we can move it so that it lies on a $(k-1)$-dimensional facet F' of the feasible polyhedron, using the proceeding argumentation. Using it repeatedly, one ends up in a vertex of the polyhedron or in an unbounded solution.

Thus, given an instance of LPI, the LP solver should output one of the following answers.

(A) **Finite.** The optimal solution is finite, and the solver would provide a vertex which realizes the optimal solution.

(B) **Unbounded.** The given LPI has an unbounded solution. In this case, the LP solver would output a ray ζ, such that the ζ lies inside the feasible region and it points down the negative x_d-axis direction.

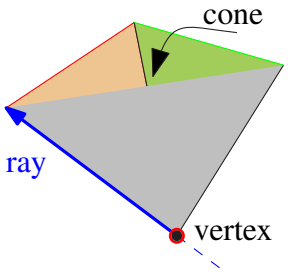
(C) **Infeasible.** The given LPI does not have any point which complies with all the given inequalities. In this case the solver would output $d+1$ constraints which are infeasible on their own.

Lemma 8.1.1. *Given a set of d linear inequalities in $\mathbb{R}^d$, one can compute the vertex induced by the intersection of their boundaries in $O(d^3)$ time.*

Proof: Write down the system of equalities that the vertex must fulfill. It is a system of d equalities in d variables and it can be solved in $O(d^3)$ time using Gaussian elimination. ∎

A ***cone*** is the intersection of d constraints, where its apex is the vertex associated with this set of constraints. A set of such d constraints is a ***basis***. An intersection of $d-1$ of the hyperplanes of a basis

forms a line and intersecting this line with the cone of the basis forms a ray. Clipping the same line to the feasible region would yield either a segment, referred to as an ***edge*** of the polyhedron, or a ray (if the feasible region is an unbounded polyhedron). An edge of the polyhedron connects two vertices of the polyhedron.



As such, one can think about the boundary of the feasible region as inducing a graph – its vertices and edges are the vertices and edges of the polyhedron, respectively. Since every vertex has d hyperplanes defining it (its basis) and an adjacent edge is defined by $d-1$ of these hyperplanes, it follows that each vertex has $\binom{d}{d-1} = d$ edges adjacent to it.

The following lemma tells us when we have an optimal vertex. While it is intuitively clear, its proof requires a systematic understanding of what the feasible region of a linear program looks like, and we omit it here – see bibliographical notes for details.

Lemma 8.1.2. *Let L be a given LPI, and let $\mathcal{P}$ denote its feasible region. Let v be a vertex of $\mathcal{P}$, such that all the d rays emanating from v are in the upward x_d-axis direction (i.e., the direction vectors of all these d rays have positive x_d-coordinate). Then v is the lowest (in the x_d-axis direction) point in $\mathcal{P}$ and it is thus the optimal solution to L.*

Interestingly, when we are at a vertex v of the feasible region, it is easy to find the adjacent vertices. Indeed, compute the d rays emanating from v. For such a ray, intersect it with all the constraints of the LPI. The closest intersection point along this ray is the vertex u of the feasible region adjacent to v. Doing this naively takes $O(dn + d^{O(1)})$ time.

Lemma **??** offers a simple algorithm for computing the optimal solution for an LPI. Start from a feasible vertex of the LPI. As long as this vertex has at least one ray that points downward, follow this ray to an adjacent vertex on the feasible polytope that is lower than the current vertex (i.e., compute the d rays emanating from the current vertex, and follow one of the rays that points downward, till you hit a new vertex). Repeat this till the current vertex has all rays pointing upward, by Lemma **??** this is the optimal solution. Up to tedious (and non-trivial) details this is the ***simplex*** algorithm.

Lemma 8.1.3. *If L is an LPI in d dimensions which is not feasible, then there exist $d+1$ inequalities in L which are infeasible on their own.*

Proof: Let H be the halfspaces defined by the constraints of L. Let $\binom{H}{d+1}$ be the set of all $d+1$ distinct tuples of halfspaces from H. If for all $S \in \binom{H}{d+1}$, we have that $\cap_{h \in S} h \neq \emptyset$, then by Helly's theorem, we have that $\cap_{h \in H} h \neq \emptyset$. But this would imply that L is feasible. It thus must be that that there is $S \in \binom{H}{d+1}$ such that the intersection of all its halfspaces is empty. ∎

Note that given a set of $d+1$ inequalities, it is easy to verify (in polynomial time in d) if they are feasible or not. Indeed, compute the $\binom{d+1}{d}$ vertices formed by this set of constraints, and check whether any of these vertices are feasible (for these $d+1$ constraints). If all of them are infeasible, then this set of constraints is infeasible.

8.2. Low-dimensional linear programming

8.2.1. An algorithm for a restricted case

There are a lot of tedious details that one has to take care of to make things work with linear programming. As such, we will first describe the algorithm for a special case and then provide the envelope required so that one can use it to solve the general case.

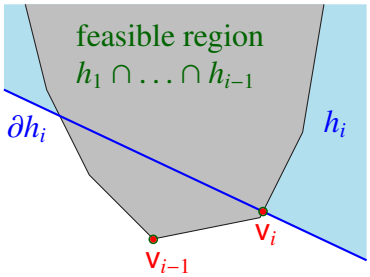
A vertex v is ***acceptable*** if all the d rays associated with it point upward (note that the vertex might not be feasible). The optimal solution (if it is finite) must be located at an acceptable vertex.

Input for the restricted case. The input for the restricted case is an LPI L, which is defined by a set of n linear inequalities in $\mathbb{R}^d$, and a basis $B = \{h_1, \ldots, h_d\}$ of an acceptable vertex.

Let $h_{d+1}, \ldots, h_n$ be a random permutation of the remaining constraints of the LPI L.

We are looking for the lowest point in $\mathbb{R}^d$ which is feasible for L. Our algorithm is randomized incremental. At the ith step, for $i > d$, it will maintain the optimal solution for the first i constraints. As such, in the ith step, the algorithm checks whether the optimal solution v_{i-1} of the previous iteration is still feasible with the new constraint h_i (namely, the algorithm checks if v_{i-1} is inside the halfspace defined by h_i). If v_{i-1} is still feasible, then it is still the optimal solution, and we set $\mathsf{v}_i \leftarrow \mathsf{v}_{i-1}$.

The more interesting case is when $\mathsf{v}_{i-1} \notin h_i$. First, we check if the basis of v_{i-1} together with h_i forms a set of constraints which is infeasible. If so, the given LPI is infeasible, and we output $B(\mathsf{v}_{i-1}) \cup \{h_i\}$ as the proof of infeasibility.



Otherwise, the new optimal solution must lie on the hyperplane associated with h_i. As such, we recursively compute the lowest vertex in the $(d-1)$-dimensional polyhedron $(\partial h_i) \cap \bigcap_{j=1}^{i-1} h_j$, where ∂h_i denotes the hyperplane which is the boundary of the halfspace h_i. This is a linear program involving $i-1$ constraints, and it involves $d-1$ variables since the LPI lies on the $(d-1)$-dimensional hyperplane ∂h_i. The solution found, v_i, is defined by a basis of $d-1$ constraints in the $(d-1)$-dimensional subspace ∂h_i, and adding h_i to it results in an acceptable vertex that is feasible in the original d-dimensional space. We continue to the next iteration.

Clearly, the vertex v_n is the required optimal solution.

8.2.1.1. Running time analysis

Every set of d constraints is feasible and computing the vertex formed by this constraint takes $O(d^3)$ time, by Lemma **??**.

Let X_i be an indicator variable that is 1 if and only if the vertex v_i is recomputed in the ith iteration (by performing a recursive call). This happens only if h_i is one of the d constraints in the basis of v_i. Since there are most d constraints that define the basis and there are at least $i-d$ constraints that are being randomly ordered (as the first d slots are fixed), we have that the probability that $\mathsf{v}_i \neq \mathsf{v}_{i-1}$ is

$$\alpha_i = \mathbb{P}[X_i = 1] \leq \min\left(\frac{d}{i-d}\ ,\ 1\right) \leq \frac{2d}{i},$$

for $i \geq d+1$, as can be easily verified.[2] So, let $T(m,d)$ be the expected time to solve an LPI with m constraints in d dimensions. We have that $T(d,d) = O(d^3)$ by the above. Now, in every iteration, we need to check if the current solution lies inside the new constraint, which takes $O(d)$ time per iteration and $O(dm)$ time overall.

Now, if $X_i = 1$, then we need to update each of the $i-1$ constraints to lie on the hyperplane h_i. The hyperplane h_i defines a linear equality, which we can use to eliminate one of the variables. This takes

[2] Indeed, $\frac{(d)+d}{(i-d)+d}$ lies between $\frac{d}{i-d}$ and $\frac{d}{d} = 1$.

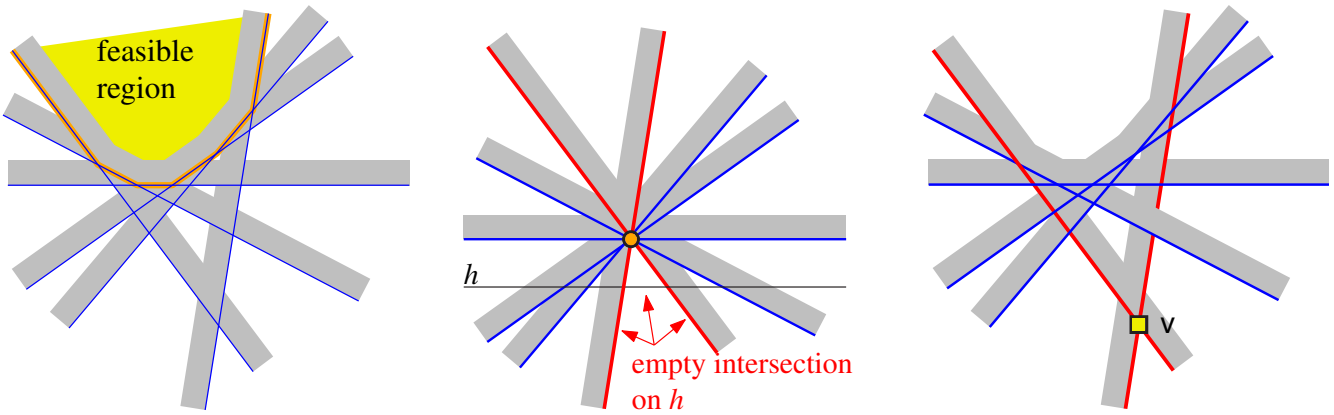


Figure 8.2.1: Demonstrating the algorithm for the general case: (a) given constraints and feasible region, (b) constraints moved to pass through the origin, and (c) the resulting acceptable vertex v.

$O(di)$ time, and we have to do the recursive call. The probability that this happens is α_i. As such, we have

$$\begin{aligned} T(m,d) &= \mathbb{E}\left[O(md) + \sum_{i=d+1}^{m} X_i(di + T(i-1,d-1))\right] \\ &\leq O(md) + \sum_{i=d+1}^{m} \alpha_i\Big(di + T(i-1,d-1)\Big) \\ &= O(md) + \sum_{i=d+1}^{m} \frac{2d}{i}\Big(di + T(i-1,d-1)\Big) \\ &= O\Big(md^2\Big) + \sum_{i=d+1}^{m} \frac{2d}{i} T(i-1,d-1). \end{aligned}$$

Guessing that $T(m,d) \leq \mathsf{c}_d m$, we have that

$$T(m,d) \leq \widehat{\mathsf{c}}_1 md^2 + \sum_{i=d+1}^{m} \frac{2d}{i}\mathsf{c}_{d-1}(i-1) \leq \widehat{\mathsf{c}}_1 md^2 + \sum_{i=d+1}^{m} 2d\mathsf{c}_{d-1} = (\widehat{\mathsf{c}}_1 d^2 + 2d\mathsf{c}_{d-1})m,$$

where $\widehat{\mathsf{c}}_1$ is some absolute constant. We need that $\widehat{\mathsf{c}}_1 d^2 + 2\mathsf{c}_{d-1}d \leq \mathsf{c}_d$, which holds for $\mathsf{c}_d = O\Big((3d)^d\Big)$ and $T(m,d) = O\Big((3d)^d m\Big)$.

Lemma 8.2.1. *Given an LPI with n constraints in d dimensions and an acceptable vertex for this LPI, then can compute the optimal solution in expected $O\Big((3d)^d n\Big)$ time.*

8.2.2. The algorithm for the general case

8.2.2.0.1 Algorithm for the general case.. Let L be the given LPI, and let L' be the instance formed by translating all the constraints so that they pass through the origin. Next, let h be the hyperplane $x_d = -1$. Consider a solution to the LP L' when restricted to h. This is a $(d-1)$-dimensional instance of linear programming, and it can be solved recursively.

If the recursive call on $L' \cap h$ returned no solution, then the d constraints that prove that the LP L' is infeasible on h corresponds to a basis in L of a vertex v which is acceptable in the original LPI. Indeed, as we move these d constraints to the origin, their intersection on h is empty (i.e., the cone that their intersection forms is unbounded only in the upward direction). As such, we can now apply the algorithm of Lemma **??** to solve the given LPI. See Figure **??**.

If there is a solution to $L' \cap h$, then it is a vertex v on h which is feasible. Thus, consider the original set of $d-1$ constraints in L that corresponds to the basis B of v. Let ℓ be the line formed by the intersection of the hyperplanes of B. It is now easy to verify that the intersection of the feasible region with this line is an unbounded ray, and the algorithm returns this unbounded (downward oriented) ray, as a proof that the LPI is unbounded.

Theorem 8.2.2. *Given an LP instance with n constraints defined over d variables, the algorithm described above solve it in expected $O\Big((3d)^d n\Big)$ time.*

Proof: The algorithm for the general case is describe above. The expected running time is

$$S(n,d) = O(nd) + S(n,d-1) + T(m,d),$$

where $T(m,d)$ is the time to solve an LP in the restricted case of Section **??**. Indeed, we first solve the problem on the $(d-1)$-dimensional subspace $h \equiv x_d = -1$. This takes $O(dn) + S(n,d-1)$ time (we need to rewrite the constraints for the lower-dimensional instance, and that takes $O(dn)$ time). If the solution on h is feasible, then the original LPI has an unbounded solution, and we return it. Otherwise, we obtained an acceptable vertex, and we can use the special case algorithm on the original LPI. Now, the solution to this recurrence is $O\Big((3d)^d n\Big)$; see Lemma **??**. ∎