Georgia Tech Fall 2024	CS 6515: Intro to Grad Algorithms

Lecture : Simplex Method

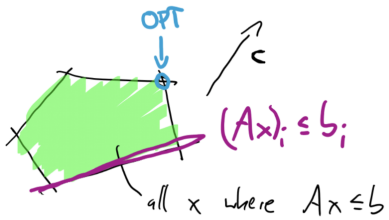
Scribe: *Prisha Sheth; Siddharth M. Sundaram* Last updated: March 18, 2025

1 Simplex Algorithm

1.1 LP Recap

We have the following standard form for LPs:

$$\max c^T x$$
$$s.t.\ Ax \leq b$$



Goal: We want an algorithm that can solve these kinds of problems to find the optimal value of x that satisfies the constraints.

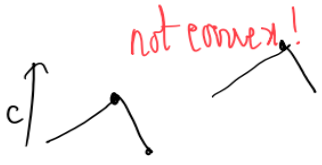
1.2 High-Level Algorithm Idea

- Assume we have some vertex $x \in \mathbb{R}^d$
- Repeat:
  - Check all edges incident to the vertex
  - If one of them is improving $c^T x$, then move along that edge to the next vertex
  - Else, return current vertex x

Our goal in this lecture is to formalize this intuitive algorithm.

Question 1. *Why can't there be a local optimum?*

For a formal proof we need some more definitions, so it's deferred to Lemma 1, near the end of these notes.
For now I'll only give an intuitive explanation/picture. Consider the picture on the right. Assume we are in a local optimum (left vertex) but there is another vertex with better $c^\top x$. Then at least from the picture that would seem to violate convexity.



Question 2. *How does this not boil down to a graph problem? We are looking for a path, so can't we run BFS/DFS?*

Answer: We could, but we do not know the graph. Computing the graph can take exponential time! Formally, the graph is defined as follows:

Definition 1. *Given a polytope $P = \{x \mid Ax \leq b\}$, the "polytope graph" $G = (V, E)$ is the undirected graph where V=vertices of the polytope, E=edges of the polytope.*

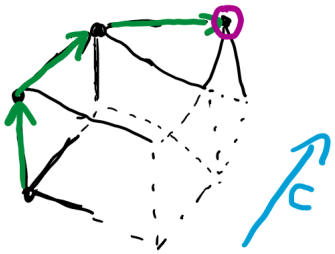
Keep in mind that the input when solving a linear program is matrix A, and vector b. We do not know yet what the vertices are. Computing all the vertices is super slow because there can be exponentially many vertices.

Withy only $2n$ constraints, we can already have 2^n vertices.

Example: $x \in \mathbb{R}^n$, $0 \leq x_i \leq 1$ forms a hypercube. Here each $x \in \{0, 1\}^n$ is a vertex, so there are 2^n many.

To prevent computation of the entire polytope graph, the simplex algorithm will only compute the edges incident to the currently visited vertex. Then go along one of the edges to the next vertex, and repeat. This way, we have a chance to only explore a small part of the entire polytope graph.

Consider the example on the right. Whenever we visit a vertex, we compute the incident edges (black lines), pick one of them (green line) and move along that edge to the next vertex. Then compute the incident edges for that next vertex and repeat. Importantly, the dotted edges/vertices are never visited by the algorithm so we never compute them.



Question 3. *In moving from one vertex to another, how should we choose among multiple edges that increase the objective value?*

Answer: There actually is not a clear answer to this question; it is still an open question as to what the best strategy would be. A popular choice is to greedily pick the direction that increases $c^\top x$ the most, but there are counter examples where this can lead you to a very long path.

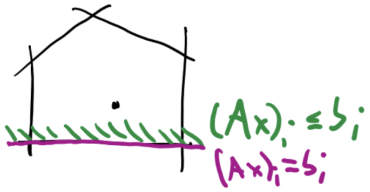
Open Problem: Does there exist a path of polynomial length from **any** starting point to the optimal point?

1.3 What is a vertex?

While we all have an intuitive understanding of what is a corner/vertex, let us formally define the notion of a vertex.

Definition 2. *For polytope $P = \{x \in \mathbb{R}^d \mid Ax \leq b\}$, vector $x \in \mathbb{R}^d$ is a vertex if:*

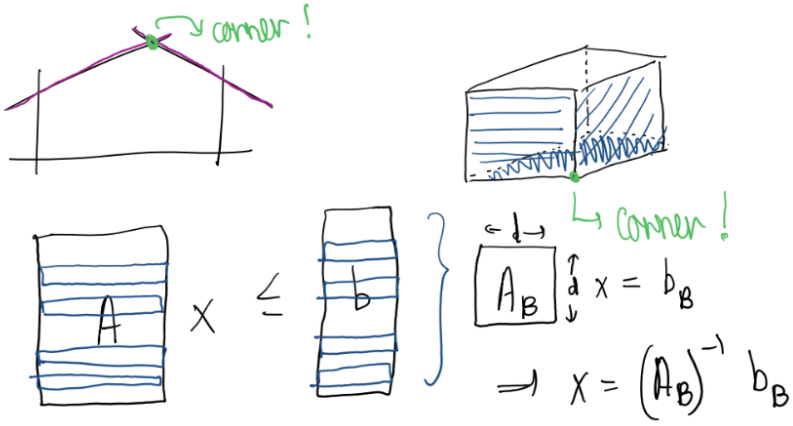
- *$Ax \leq b$, i.e., x is feasible*
- *For d constraints that are linearly independent, we have that $(Ax)_i = b_i$*



A vertex can be thought of as an intersection of d linearly independent constraints. Note that we can completely characterize any vertex using a subset $B \subseteq \{1, ..., n\}$ with $|B| = d$; denote by A_B and b_B are the d rows of A and b respectively with index in B, then we have $A_B x = b_B \Rightarrow x = (A_B)^{-1} b_B$. For this to work, the linear independence is important. Otherwise $A_B x = b_B$ does not have a unique solution.

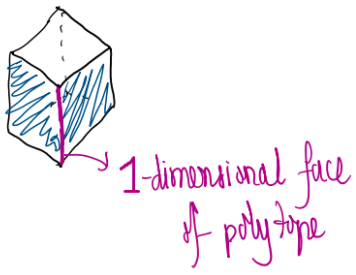
Definition 3. *We also call the set of d linear independent constraints that define a vertex a "feasible basis".*

This is because A_B is a basis for $\mathbb{R}^d$ and because the solution $A_B x = b_B$ is a feasible point of the linear program $(Ax \leq b)$.



Question 4. *What happens when we specify fewer than d constraints (i.e. $d - k$)?*

Answer: Then we don't have a vertex and have a k-dimensional face of a polytope, where the dimension k is equal to the number of missing tight constraints.
To see why, observe that the linear system $A_B x = b_B$ has d variables but only $d - k$ linear independent constraints. So the solution space is k-dimensional. E.g., for $k = 1$ (when we have only $d - 1$ tight constraints) we are specifying an edge, since an edge/line is 1-dimensional.



Question 5. *What happens if we may consider some linearly* dependent *constraints in the set of d constraints?*

Answer: Again, we will not specify a unique point with $A_B = b_B$ in that case, so we do not get a vertex.

Question 6. *To run simplex, we move from vertex to vertex; how do we get the first vertex?*

Answer: Say our linear program is $Ax \leq b$ subject to $x \geq 0$; we can assume this is the case without loss of generality. This can be done via transformation where we split x_i to x_i^+, x_i^- similar to lecture 15. So any linear program can be converted to a linear program of this form. Now, two cases arise:

1. If $b \geq 0$, then we can pick $x = 0$ as the initial vertex. The constraints $x_i \geq 0$ are the tight d linearly independent constraints.
2. If b has negative entries:
   - Add a new variable z and solve the linear program:

$$\begin{aligned} \min \ & -z \\ s.t. \ & Ax - z \leq b \\ & z \geq 0 \end{aligned}$$

     Here, we use the initial vertex $x = 0$ and $z = -\min_i b_i$. This is a vertex because we have $d + 1$ linearly independent tight constraints $x \geq 0$ and $(Ax - z)_i = b_i$ for the i with smallest b_i.
   - Observe that the optimal solution has $z = 0$ if and only if there exists an x with $Ax \leq b$.
   - After solving the new linear program, we can use the computed x as the initial vertex for the original linear program.

Question 7. *To specify a vertex, we must specify d constraints of the linear program to be tight; what if the linear program itself has less than d constraints?*

Answer: Generally, these LPs would have unbounded feasible regions. In linear algebra terms, any such subset of constraints when specified to be tight (equality) would be **underdetermined**.

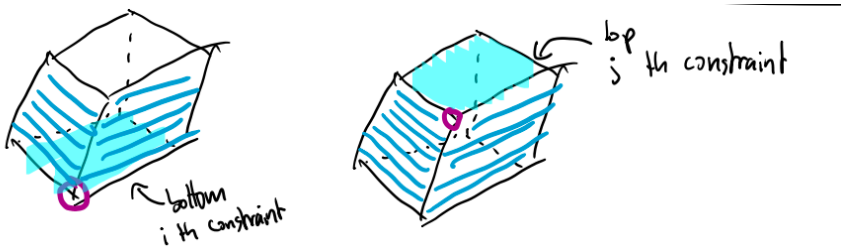
To summarize:

- We can identify a vertex by the set $B \subseteq \{1, ..., N\}$ that are the indices of the tight constraints (ones that have equality instead of inequality)
- Given B, the vertex is $x = (A_B)^{-1} b_B$

If you pick an arbitrary value of B (number of constraints), then we can end up with an infeasible solution.

1.4 What is an edge?

We said the simplex algorithm goes from vertex to vertex by moving along edges. So we need characterization for the edges, and what it means for two vertices to be connected by an edge.



Two vertices x and x' are connected by an edge if there are feasible bases B and B' (with $A_B x = b_B$ and $A_{B'} x' = b_{B'}$) that differ by only one element (i.e. $|B \triangle B'| = 2$, where $\triangle$ denotes symmetric difference). Thus it will satisfy:

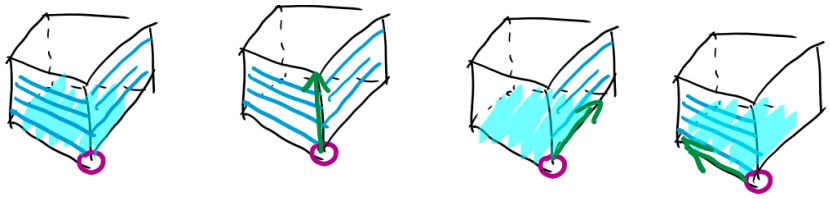
$$B' = B \backslash \{i\} \cup \{j\}$$

for some indices $i, j \in \{1, .., n\}$.

Observe that all points p on the edge connecting x and x' will satisfy $A_{B\backslash\{i\}} p = b_{B\backslash\{i\}}$ (this is a linear system with $d-1$ equalities, so the solution space is 1-dimensional).

1.5 How do we get all edges incident to a vertex?

When the simplex algorithm is at some vertex x, it must check which edges could improve $c^\top x$. For this we need to compute the set of edges.



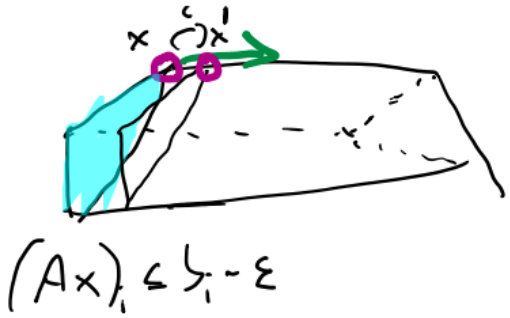
For feasible basis B (ie, vertex x), each index $i \in B$ (the constraint that we could remove) represents an edge.

(Small remark: it could technically happen that the edge has length 0, more on this later in Section 2.)

1.6 How do we get the direction of an edge?

We previously said that an edge can be represented by the index $i \in B$ that will be removed. But how do we compute the direction $\in \mathbb{R}^d$ in which that edge is going? We need that direction to figure out if $c^\top x$ will be improved.

Intuition: If we perturb a face (constraint) of the feasible region by a slight amount to restrict further the feasible region, the direction the face would be pushed is the same of that of the incident edge. Pictorially:



In the above, we consider x, and its corresponding specifying subset of constraints B. We consider some $i \in B$, say constraint $(Ax)_i \leq b_i$, and for small $\epsilon > 0$, we consider the perturbed constraint $(Ax)_i \leq b_i - \epsilon$. Clearly, this new constraint is more restrictive, and so the feasible region should decrease in size as shown above. Accordingly, this would give the direction of an incident edge. Say the vertex x get shifted to x' under the new constraint.

Now, the direction of the edge can be computed as the vector from x to x' as follows:

$$\begin{aligned} x' - x &= (A_B)^{-1}(b_B - \epsilon \cdot e_i) - (A_B)^{-1}b_B \\ &= -(A_B)^{-1}e_i \cdot \epsilon \end{aligned}$$

In the above, $(A_B)^{-1}e_i$ gives the direction of the edge (since ϵ is a scalar).

Now, which edge are we interested in? We are only interested in traversing an edge which improves our objective value. So, we desire:

$$c^T x' > c^T x \Leftrightarrow c^T \cdot (x' - x) > 0 \Leftrightarrow c^T (A_B)^{-1} e_i < 0$$

As a result, picking the edge to improve the solution precisely means to pick $i \in B$ where $c^T(A_B)^{-1}e_i < 0 \Leftrightarrow (c^T(A_B)^{-1})_i < 0$. Note that $c^T(A_B)^{-1}e_i = (c^T(A_B)^{-1})_i$ since e_i has 1 only at entry i and 0 everywhere else.

Picking the correct edge to improve the solution means to pick $i \in B$ where $(c^T(A_B)^{-1})_i < 0$.

1.7 How far can we move?

From the above, we saw that for a step size ϵ the new vertex point x' becomes $x' = x - (A_B)^{-1}e_i \cdot \epsilon$. For convenience, we now switch notation and use λ instead of ϵ. We now desire to find the largest value of λ so that $x' = x - (A_B)^{-1}e_i \cdot \lambda$ still remains inside the feasible region.

So, we have to maximize λ such that:

$$\begin{aligned} &(A(x-(A_B)^{-1}e_i\cdot\lambda))_j \leq b_j, \forall j \\ \Leftrightarrow &-(A(A_B)^{-1}e_i\cdot\lambda)_j \leq b_j-(Ax)_j \\ \Leftrightarrow &\lambda = \min_{j\notin B}\frac{b_j-(Ax)_j}{-(A(A_B)^{-1}e_i)_j} \end{aligned}$$

The above expression, while looking convoluted has the following intuitive meaning:

- λ represents a notion of "how long" (time) that a vertex can be moved while remaining in the feasible region, i.e., before we hit the first different constraint (outside B)
- j is the first constraint we hit
- $-(A(A_B)^{-1}e_i)_j$ represents a notion of velocity
- $b_j-(Ax)_j$ represents a notion a distance of "how far" the constraint j is from being tight (achieving equality)
- Then, the above formulae simply boil down to the relation: distance = time * velocity

1.8 Algorithm

Algorithm 1 Simplex Algorithm for initial vertex x (specified by $B \subseteq \{1,...,n\}$)

repeat
  Find $i \in B$ where $(c^T(A_B)^{-1})_i < 0$ ▷ Check edges. Does one improve the solution?
  if there exists such i then: ▷ There is an edge that improves the solution
    $\lambda = \min_{j\notin B,(A(A_B)^{-1}e_i)_j<0}\frac{(b-Ax)_j}{-(A(A_B)^{-1}e_i)_j}$ ▷ Compute how far we can move along that edge
    $B \leftarrow (B\backslash\{i\})\cup\{j\}$
    $x \leftarrow x-\lambda\cdot(A_B)^{-1}e_i$ ▷ Move our vertex
  else
    return x

2 Optimality & Time Complexity

At the start of these notes, there was Question 1, asking why we do not get stuck in a local optimum. The following lemma proves this cannot happen.

Lemma 1. *If $(c^\top(A_B)^{-1})_i \geq 0$ for all i, then the vertex x is optimal.*

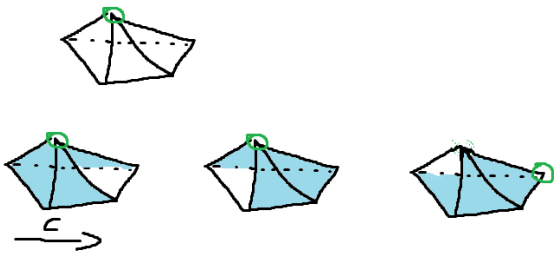
Observe that the simplex algorithm only terminates when $(c^\top(A_B)^{-1})_i \geq 0$. So the Simplex algorithm only terminates when it found an optimal solution.

Proof. Our algorithm is running on a linear program of form $\max c^\top x$, $Ax \leq b$. The dual is $\min b^\top y$, $A^\top y = c$, $y \geq 0$.

Assume for simplicity that $B = \{1....d\}$ (i.e., by reordering the rows of A). Let $y_i := (c^\top (A_B)^{-1})_i^\top = ((A_B^\top)^{-1}c)_i$ for $i \in B$, and $y_i = 0$ for $i \notin B$. Then $y \geq 0$ and $A^\top y = c$. So this is a feasible solution for the dual linear program. We also know that $b^\top y = \sum_{i \in B} b_i y_i = b_B^\top (A_B^\top)^{-1} c = x^\top c$, where x is the vertex corresponding to B. Thus by strong duality that x is an optimal solution. □

Stalling It could be that $\lambda = 0$ in Algorithm 1. This can happen when a vertex is touching more than d facets, i.e., if there are multiple different choices for the set B where $A_B x = b_B$.

Such a vertex is also called "degenerate." Consider for example the picture below. The vertex at the top has 4 incident facets: left, right, front, and back.



Now consider what the Simplex algorithm does when the direction c is to the right and initially our B are only the back, left and front constraint (highlighted in blue). When running simplex, then we get $\lambda = 0$, i.e., we are in a sense moving along an edge of length 0 and immediately hit the constraint on the right. Now our B is the front, back, and right constraint. While the set B changed, the vertex (highlighted in green) did not change. Only in the next iteration of the simplex algorithm can finally move to another vertex further on the right.

When getting stuck at a vertex for several iterations, such as the picture above, we refer to this as *stalling*. This happens often in practice.

Runtime The simplex algorithm as presented in Algorithm 1 could actually run in an infinite loop. That is a more extreme case of the stalling mentioned above. It could for example happen that when $\lambda = 0$ for several iterations, we end up with the same B that we had a few iterations before. This is referred to as *cycling*. There are various ways to prevent this, e.g., if we add tiny random noise to vector b then no vertex is degenerate and we always have $\lambda > 0$. There are also rules for tie-breaking the choice of i and j (e.g., Bland's rule) which prevent cycling. For the purpose of this course, we assume cycling does not happen since we can just apply Bland's rule.

If there is no cycling, then we can have at most $\binom{n}{d}$ iterations, because there are at most that many different sets $B \subset \{1....n\}$.

Lemma 2. *If the polytope has at most T feasible bases, then the simplex algorithm terminates after T iterations. In general, $T \leq \binom{n}{d} < n^d$.*

While n^d is exponential, in practice the simplex algorithm is observed to typically need only $O(n)$ iterations. One can also prove that adding small random noise to b results in a polynomial number of iterations in worst-case (see Smoothed Analysis).

Each iteration costs $O(d^2 + \text{nnz}(A)) < O(n^3)$ time, where $\text{nnz}(A)$ is the number of non-zero entries in A. The $O(d^2)$ is the cost for computing $(A_B)^{-1}$ (see problem set 8). The $O(\text{nnz}(A))$ is the cost for computing $\dfrac{(b - Ax)_j}{-(A(A_B)^{-1}e_i)_j}$ for all $j \notin B$.

From a theory perspective, $O(n^{1.529})$ time per iteration is also possible[1], but it has very large constant hidden in $O(\cdot)$ notation and you wouldn't want to use that in practice.

[1] https://arxiv.org/pdf/2010.13888