> *Simplicibus itaque verbis gaudet Mathematica Veritas, cum etiam per se*
> *simplex sit Veritatis oratio. [And thus Mathematical Truth prefers simple*
> *words, because the language of Truth is itself simple.]*
>
> — Tycho Brahe (quoting Seneca (quoting Euripides))
> *Epistolarum astronomicarum liber primus* (1596)

> *When a jar is broken, the space that was inside*
> *Merges into the space outside.*
> *In the same way, my mind has merged in God;*
> *To me, there appears no duality.*
>
> — Sankara, *Viveka-Chudamani* (c. 700), translator unknown

*26 Linear Programming Algorithms

In this lecture, we'll see a few algorithms for actually solving linear programming problems. The most famous of these, the *simplex method*, was proposed by George Dantzig in 1947. Although most variants of the simplex algorithm performs well in practice, no deterministic simplex variant is known to run in sub-exponential time in the worst case.[1] However, if the dimension of the problem is considered a constant, there are several linear programming algorithms that run in *linear* time. I'll describe a particularly simple randomized algorithm due to Raimund Seidel.

My approach to describing these algorithms will rely much more heavily on geometric intuition than the usual linear-algebraic formalism. This works better for me, but your mileage may vary. For a more traditional description of the simplex algorithm, see Robert Vanderbei's excellent textbook *Linear Programming: Foundations and Extensions* [Springer, 2001], which can be freely downloaded (but not legally printed) from the author's website.

26.1 Bases, Feasibility, and Local Optimality

Consider the canonical linear program $\max\{c \cdot x \mid Ax \le b, x \ge 0\}$, where A is an $n \times d$ constraint matrix, b is an n-dimensional coefficient vector, and c is a d-dimensional objective vector. We will interpret this linear program geometrically as looking for the lowest point in a convex polyhedron in $\mathbb{R}^d$, described as the intersection of $n + d$ halfspaces. As in the last lecture, we will consider only *non-degenerate* linear programs: Every subset of d constraint hyperplanes intersects in a single point; at most d constraint hyperplanes pass through any point; and objective vector is linearly independent from any $d - 1$ constraint vectors.

A **basis** is a subset of d constraints, which by our non-degeneracy assumption must be linearly independent. The **location** of a basis is the unique point x that satisfies all d constraints with equality; geometrically, x is the unique intersection point of the d hyperplanes. The **value** of a basis is $c \cdot x$, where x is the location of the basis. There are precisely $\binom{n+d}{d}$ bases. Geometrically, the set of constraint hyperplanes defines a decomposition of $\mathbb{R}^d$ into convex polyhedra; this cell decomposition is called the **arrangement** of the hyperplanes. Every subset of d hyperplanes (that is, every basis) defines a *vertex* of this arrangement (the location of the basis). I will use the words 'vertex' and 'basis' interchangeably.

A basis is **feasible** if its location x satisfies all the linear constraints, or geometrically, if the point x is a vertex of the polyhedron. If there are no feasible bases, the linear program is **infeasible**.

---

[1]However, there are *randomized* variants of the simplex algorithm that run in subexponential *expected* time, most notably the RANDOMFACET algorithm analyzed by Gil Kalai in 1992, and independently by Jiří Matoušek, Micha Sharir, and Emo Welzl in 1996. No randomized variant is known to run in polynomial time. In particular, in 2010, Oliver Friedmann, Thomas Dueholm Hansen, and Uri Zwick proved that the worst-case expected running time of RANDOMFACET is superpolynomial.

A basis is ***locally optimal*** if its location x is the optimal solution to the linear program with the same objective function and *only* the constraints in the basis. Geometrically, a basis is locally optimal if its location x is the lowest point in the intersection of those d halfspaces. A careful reading of the proof of the Strong Duality Theorem reveals that local optimality is the dual equivalent of feasibility; a basis is locally feasible for a linear program Π if and only if the same basis is feasible for the dual linear program $\amalg$. For this reason, locally optimal bases are sometimes also called ***dual feasible***. If there are no locally optimal bases, the linear program is ***unbounded***.[2]

Two bases are ***neighbors*** if they have $d-1$ constraints in common. Equivalently, in geometric terms, two vertices are neighbors if they lie on a *line* determined by some $d-1$ constraint hyperplanes. Every basis is a neighbor of exactly dn other bases; to change a basis into one of its neighbors, there are d choices for which constraint to remove and n choices for which constraint to add. The graph of vertices and edges on the boundary of the feasible polyhedron is a subgraph of the basis graph.

The Weak Duality Theorem implies that the value of every feasible basis is less than or equal to the value of every locally optimal basis; equivalently, every feasible vertex is higher than every locally optimal vertex. The Strong Duality Theorem implies that (under our non-degeneracy assumption), if a linear program has an optimal solution, it is the *unique* vertex that is both feasible and locally optimal. Moreover, the optimal solution is both the lowest feasible vertex and the highest locally optimal vertex.

26.2   The Primal Simplex Algorithm: Falling Marbles

From a geometric standpoint, Dantzig's simplex algorithm is very simple. The input is a set H of halfspaces; we want the lowest vertex in the intersection of these halfspaces.

```
SIMPLEX1(H):
  if ∩H = ∅
       return INFEASIBLE
  x ← any feasible vertex
  while x is not locally optimal
       ⟨⟨pivot downward, maintaining feasibility⟩⟩
       if every feasible neighbor of x is higher than x
            return UNBOUNDED
       else
            x ← any feasible neighbor of x that is lower than x
  return x
```

Let's ignore the first three lines for the moment. The algorithm maintains a feasible vertex x. At each so-called ***pivot*** operation, the algorithm moves to a *lower* vertex, so the algorithm never visits the same vertex more than once. Thus, the algorithm must halt after at most $\binom{n+d}{d}$ pivots. When the algorithm halts, either the feasible vertex x is locally optimal, and therefore the optimum vertex, or the feasible vertex x is not locally optimal but has no lower feasible neighbor, in which case the feasible region must be unbounded.

Notice that we have not specified *which* neighbor to choose at each pivot. Many different pivoting rules have been proposed, but for almost every known pivot rule, there is an input polyhedron that requires an exponential number of pivots under that rule. *No* pivoting rule is known that guarantees a polynomial number of pivots in the worst case, or even in expectation.[3]

---

[2]For non-degenerate linear programs, the feasible region is unbounded in the objective direction if and only if no basis is locally optimal. However, there are degenerate linear programs with no locally optimal basis that are infeasible.

[3]In 1957, Hirsch conjectured that for *any* linear programming instance with d variables and $n+d$ constraints, starting at any feasible basis, there is a sequence of ***at most n*** pivots that leads to the optimal basis. This long-standing conjecture was finally disproved in 2010 by Fransisco Santos, who described an counterexample with 43 variables, 86 facets, and diameter 44.

26.3   The Dual Simplex Algorithm: Rising Bubbles

We can also geometrically interpret the execution of the simplex algorithm on the dual linear program П. Again, the input is a set H of halfspaces, and we want the lowest vertex in the intersection of these halfspaces. By the Strong Duality Theorem, this is the same as the *highest locally-optimal* vertex in the hyperplane arrangement.

SIMPLEX2(H):
    if there is no locally optimal vertex
        return UNBOUNDED
    $x \leftarrow$ any locally optimal vertex
    while x is not feasbile
        ⟨⟨*pivot upward, maintaining local optimality*⟩⟩
        if every locally optimal neighbor of x is lower than x
            return INFEASIBLE
        else
            $x \leftarrow$ any locally-optimal neighbor of x that is higher than x
    return x

Let's ignore the first three lines for the moment. The algorithm maintains a locally optimal vertex x. At each pivot operation, it moves to a *higher* vertex, so the algorithm never visits the same vertex more than once. Thus, the algorithm must halt after at most $\binom{n+d}{d}$ pivots. When the algorithm halts, either the locally optimal vertex x is feasible, and therefore the optimum vertex, or the locally optimal vertex x is not feasible but has no higher locally optimal neighbor, in which case the problem must be infeasible.
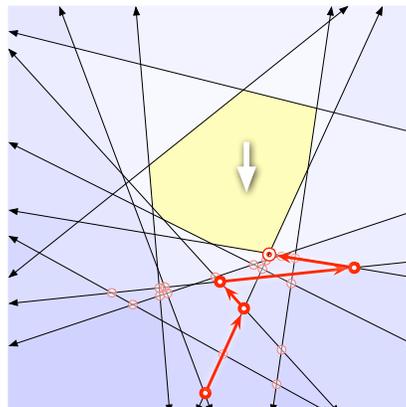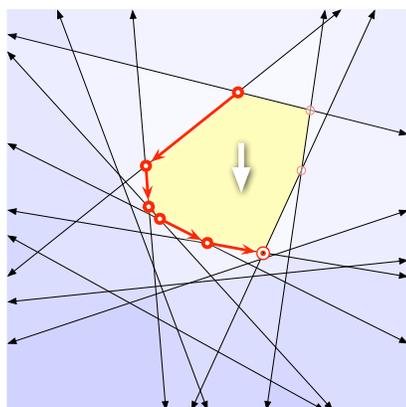
The primal simplex (falling marble) algorithm in action.   The dual simplex (rising bubble) algorithm in action.

From the standpoint of linear algebra, there is absolutely no difference between running SIMPLEX1 on any linear program П and running SIMPLEX2 on the dual linear program П. The actual *code* is identical. The only difference between the two algorithms is how we interpret the linear algebra geometrically.

26.4   Computing the Initial Basis

To complete our description of the simplex algorithm, we need to describe how to find the initial vertex x. Our algorithm relies on the following simple observations.

First, the feasibility of a vertex does not depend at all on the choice of objective vector; a vertex is either feasible for every objective function or for none. No matter how we rotate the polyhedron, every feasible vertex stays feasible. Conversely (or by duality, equivalently), the local optimality of a vertex

does not depend on the exact location of the d hyperplanes, but only on their normal directions and the objective function. No matter how we translate the hyperplanes, every locally optimal vertex stays locally optimal. In terms of the original matrix formulation, feasibility depends on A and b but not c, and local optimality depends on A and c but not b.

The second important observation is that *every* basis is locally optimal for *some* objective function. Specifically, it suffices to choose any vector that has a positive inner product with each of the normal vectors of the d chosen hyperplanes. Equivalently, we can make *any* basis feasible by translating the hyperplanes appropriately. Specifically, it suffices to translate the chosen d hyperplanes so that they pass through the origin, and then translate all the other halfspaces so that they strictly contain the origin.

Our strategy for finding our initial feasible vertex is to choose *any* vertex, choose a new objective function that makes that vertex locally optimal, and then find the optimal vertex for *that* objective function by running the (dual) simplex algorithm. This vertex must be feasible, even after we restore the original objective function!
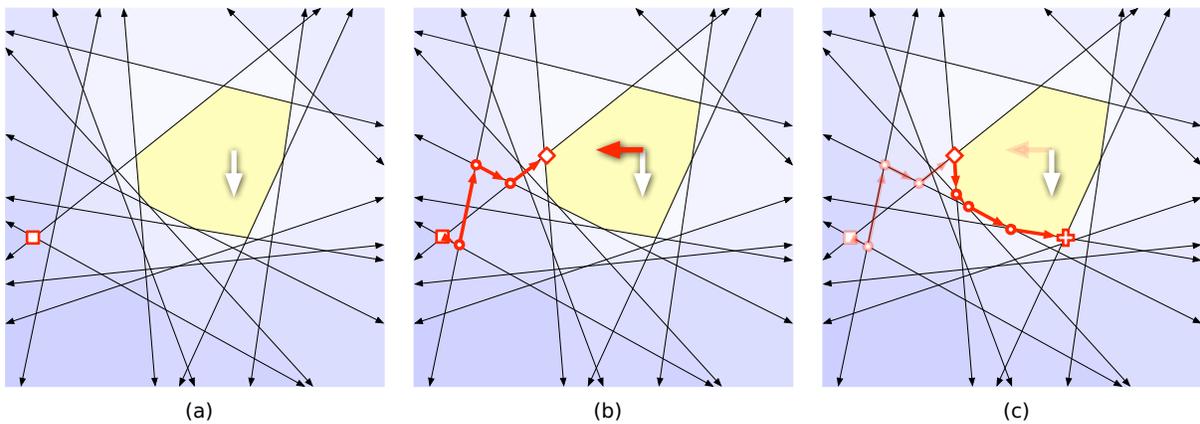
(a) Choose any basis. (b) Rotate objective to make it locally optimal, and pivot 'upward' to find a feasible basis.
(c) Pivot downward to the optimum basis for the original objective.

Equivalently, to find an initial locally optimal vertex, we choose *any* vertex, translate the hyperplanes so that that vertex becomes feasible, and then find the optimal vertex for those translated constraints using the (primal) simplex algorithm. This vertex must be locally optimal, even after we restore the hyperplanes to their original locations!
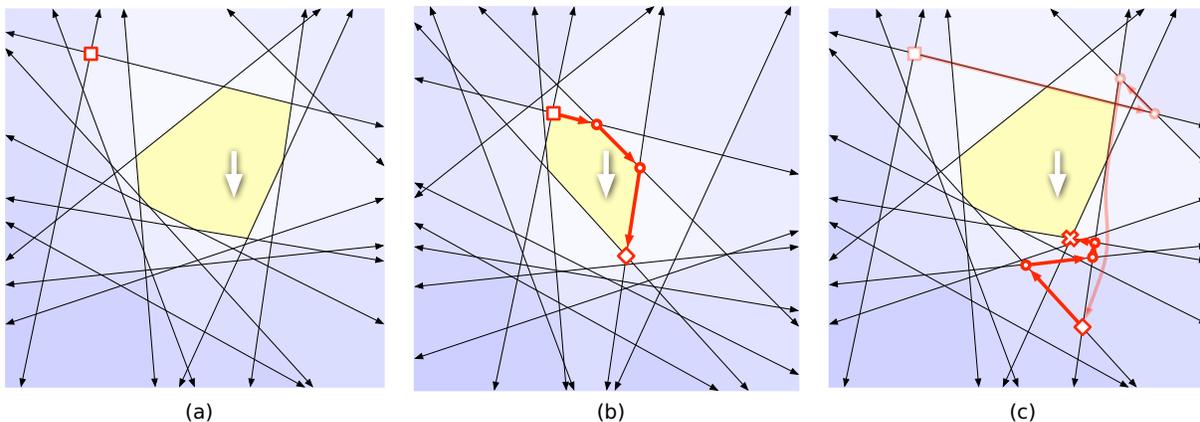
(a) Choose any basis. (b) Translate constraints to make it feasible, and pivot downward to find a locally optimal basis.
(c) Pivot upward to the optimum basis for the original constraints.

Here are more complete descriptions of the simplex algorithm with this initialization rule, in both primal and dual forms. As usual, the input is a set H of halfspaces, and the algorithms either return the lowest vertex in the intersection of these halfspaces or report that no such vertex exists.

$\underline{\text{SIMPLEX1}(H):}$
  $x \leftarrow$ any vertex
  $\tilde{H} \leftarrow$ any rotation of H that makes x locally optimal

  while x is not feasible
        if every locally optimal neighbor of x is lower (wrt $\tilde{H}$) than x
              return INFEASIBLE
        else
              $x \leftarrow$ any locally optimal neighbor of x that is higher (wrt $\tilde{H}$) than x
  while x is not locally optimal
        if every feasible neighbor of x is higher than x
              return UNBOUNDED
        else
              $x \leftarrow$ any feasible neighbor of x that is lower than x
  return x

$\underline{\text{SIMPLEX2}(H):}$
  $x \leftarrow$ any vertex
  $\tilde{H} \leftarrow$ any translation of H that makes x feasible

  while x is not locally optimal
        if every feasible neighbor of x is higher (wrt $\tilde{H}$) than x
              return UNBOUNDED
        else
              $x \leftarrow$ any feasible neighbor of x that is lower (wrt $\tilde{H}$) than x
  while x is not feasible
        if every locally optimal neighbor of x is lower than x
              return INFEASIBLE
        else
              $x \leftarrow$ any locally-optimal neighbor of x that is higher than x
  return x

26.5   Linear Expected Time for Fixed Dimensions

In most geometric applications of linear programming, the number of variables is a small constant, but the number of constraints may still be very large.

The input to the following algorithm is a set H of n halfspaces and a set B of b hyperplanes. (B stands for *basis*.) The algorithm returns the lowest point in the intersection of the halfspaces in H and the hyperplanes B. At the top level of recursion, B is empty. I will implicitly assume that the linear program is both feasible and bounded. (If necessary, we can guarantee boundedness by adding a single halfspace to H, and we can guarantee feasibility by adding a dimension.) A point x **violates** a constraint h if it is not contained in the corresponding halfspace.

```
SEIDELLP(H, B):
    if |B| = d
        return ⋂B
    if |H ∪ B| = d
        return ⋂(H ∪ B)
    h ← random element of H
    x ← SEIDELLP(H \ h, B)          (∗)
    if x violates h
        return SEIDELLP(H \ h, B ∪ ∂h)
    else
        return x
```

The point x recursively computed in line $(\ast)$ is the optimal solution if and only if the random halfspace h is *not* one of the d halfspaces that define the optimal solution. In other words, the probability of calling SEIDELLP$(H, B \cup h)$ is exactly $(d - b)/n$. Thus, we have the following recurrence for the expected number of recursive calls for this algorithm:

$$T(n, b) = \begin{cases} 1 & \text{if } b = d \text{ or } n + b = d \\ T(n - 1, b) + \dfrac{d - b}{n} \cdot T(n - 1, b + 1) & \text{otherwise} \end{cases}$$

The recurrence is somewhat simpler if we write $\delta = d - b$:

$$T(n, \delta) = \begin{cases} 1 & \text{if } \delta = 0 \text{ or } n = \delta \\ T(n - 1, \delta) + \dfrac{\delta}{n} \cdot T(n - 1, \delta - 1) & \text{otherwise} \end{cases}$$

It's easy to prove by induction that $T(n, \delta) = O(\delta! \, n)$:

$$\begin{aligned}
T(n, \delta) &= T(n - 1, \delta) + \frac{\delta}{n} \cdot T(n - 1, \delta - 1) \\
&\leq \delta!(n - 1) + \frac{\delta}{n}(\delta - 1)! \cdot (n - 1) && \text{[induction hypothesis]} \\
&= \delta!(n - 1) + \delta! \frac{n - 1}{n} \\
&\leq \delta! \, n
\end{aligned}$$

At the top level of recursion, we perform one violation test in $O(d)$ time. In each of the base cases, we spend $O(d^3)$ time computing the intersection point of d hyperplanes, and in the first base case, we spend $O(dn)$ additional time testing for violations. More careful analysis implies that the algorithm runs in $O(d! \cdot n)$ *expected time*.

Exercises

1. Fix a non-degenerate linear program in canonical form with d variables and $n + d$ constraints.

   (a) Prove that every *feasible* basis has exactly d *feasible* neighbors.

   (b) Prove that every *locally optimal* basis has exactly n *locally optimal* neighbors.

2. Suppose you have a subroutine that can solve linear programs in polynomial time, but only if they are both feasible and bounded. Describe an algorithm that solves *arbitrary* linear programs in polynomial time. Your algorithm should return an optimal solution if one exists; if no optimum exists, your algorithm should report that the input instance is UNBOUNDED or INFEASIBLE, whichever is appropriate. *[Hint: Add one variable and one constraint.]*

3. (a) Give an example of a non-empty polyhedron $Ax \le b$ that is unbounded for every objective vector c.

   (b) Give an example of an infeasible linear program whose dual is also infeasible.

   In both cases, your linear program will be degenerate.

4. Describe and analyze an algorithm that solves the following problem in $O(n)$ time: Given n red points and n blue points in the plane, either find a line that separates every red point from every blue point, or prove that no such line exists.

5. The single-source shortest path problem can be formulated as a linear programming problem, with one variable d_v for each vertex $v \ne s$ in the input graph, as follows:

$$
\begin{aligned}
\text{maximize} \quad & \sum_v d_v \\
\text{subject to} \quad & d_v \le \ell_{s \to v} && \text{for every edge } s \to v \\
& d_v - d_u \le \ell_{u \to v} && \text{for every edge } u \to v \text{ with } u \ne s \\
& d_v \ge 0 && \text{for every vertex } v \ne s
\end{aligned}
$$

This problem asks you to describe the behavior of the simplex algorithm on this linear program in terms of distances. Assume that the edge weights $\ell_{u \to v}$ are all non-negative and that there is a unique shortest path between any two vertices in the graph.

   (a) What is a basis for this linear program? What is a feasible basis? What is a locally optimal basis?

   (b) Show that in the optimal basis, every variable d_v is equal to the shortest-path distance from s to v.

   (c) Describe the primal simplex algorithm for the shortest-path linear program directly in terms of vertex distances. In particular, what does it mean to pivot from a feasible basis to a neighboring feasible basis, and how can we execute such a pivot quickly?

   (d) Describe the dual simplex algorithm for the shortest-path linear program directly in terms of vertex distances. In particular, what does it mean to pivot from a locally optimal basis to a neighboring locally optimal basis, and how can we execute such a pivot quickly?

   (e) Is Dijkstra's algorithm an instance of the simplex method? Justify your answer.

   (f) Is Shimbel's algorithm an instance of the simplex method? Justify your answer.

6. The maximum (s, t)-flow problem can be formulated as a linear programming problem, with one

variable $f_{u \to v}$ for each edge $u \to v$ in the input graph:

$$
\begin{aligned}
\text{maximize} \quad & \sum_{w} f_{s \to w} - \sum_{u} f_{u \to s} \\
\text{subject to} \quad & \sum_{w} f_{v \to w} - \sum_{u} f_{u \to v} = 0 \qquad \text{for every vertex } v \neq s, t \\
& \phantom{\sum_{w}} f_{u \to v} \leq c_{u \to v} \quad \text{for every edge } u \to v \\
& \phantom{\sum_{w}} f_{u \to v} \geq 0 \qquad \text{for every edge } u \to v
\end{aligned}
$$

This problem asks you to describe the behavior of the simplex algorithm on this linear program in terms of flows.

(a) What is a basis for this linear program? What is a feasible basis? What is a locally optimal basis?

(b) Show that the optimal basis represents a maximum flow.

(c) Describe the primal simplex algorithm for the flow linear program directly in terms of flows. In particular, what does it mean to pivot from a feasible basis to a neighboring feasible basis, and how can we execute such a pivot quickly?

(d) Describe the dual simplex algorithm for the flow linear program directly in terms of flows. In particular, what does it mean to pivot from a locally optimal basis to a neighboring locally optimal basis, and how can we execute such a pivot quickly?

(e) Is the Ford-Fulkerson augmenting path algorithm an instance of the simplex method? Justify your answer. *[Hint: There is a one-line argument.]*

7. (a) Formulate the minimum spanning tree problem as an instance of linear programming. Try to minimize the number of variables and constraints.

   (b) In your MST linear program, what is a basis? What is a feasible basis? What is a locally optimal basis?

   (c) Describe the primal simplex algorithm for your MST linear program directly in terms of the input graph. In particular, what does it mean to pivot from a feasible basis to a neighboring feasible basis, and how can we execute such a pivot quickly?

   (d) Describe the dual simplex algorithm for your MST linear program directly in terms of the input graph. In particular, what does it mean to pivot from a locally optimal basis to a neighboring locally optimal basis, and how can we execute such a pivot quickly?

   (e) Which of the classical MST algorithms (Borůvka, Jarník, Kruskal, reverse greedy), if any, are instances of the simplex method? Justify your answer.