Solving linear programs

Ben Rosenberg

February 9, 2026

Overview

- Complexity notation review (big-O primer)
- GILP
- Simplex algorithm and intuition
- Limitations of simplex/adversarial cases
- Interior point methods

Big-O notation

Informally, big-O notation is a way of specifying how much "work" an algorithm performs as the size of the input we pass to it increases.

Some examples:

$O(n)$ [fast]:

```
def print_list_contents(L):
    for i in L:
        print(i)
```

$O(n^2)$ [fast-ish]:

```
def print_all_differences(L):
    for i in L:
        for j in L:
            print(i - j)
```

$O(2^n)$ [really slow]:

```
def bad_fibonacci(n):
    return (
        bad_fibonacci(n-1)
        + bad_fibonacci(n-2)
    )
```

$O(1)$ [instant]:

```
def is_even_length(L):
    return len(L) % 2 == 0
```

GILP

Geometric Interpretation of Linear Programs

Useful website/Python package for understanding LPs and the simplex algorithm geometrically

https://gilp.henryrobbins.com/en/latest/index.html

Solving LPs

Two methods typically used in practice:

1. Simplex algorithm (or "revised" simplex)
2. Interior point methods

Simplex

A simplex is an n-dimensional convex polyhedron having exactly $n-1$ vertices. The boundary of the simplex contains simplices of lower dimension which are called simplicial faces. A simplex in zero dimension is a point; in one dimension it is a line; in two dimensions it is a triangle; and in three dimensions it is a tetrahedron.

– Saul I. Gass. 1985. Linear programming: methods and applications (5th ed.). p. 37

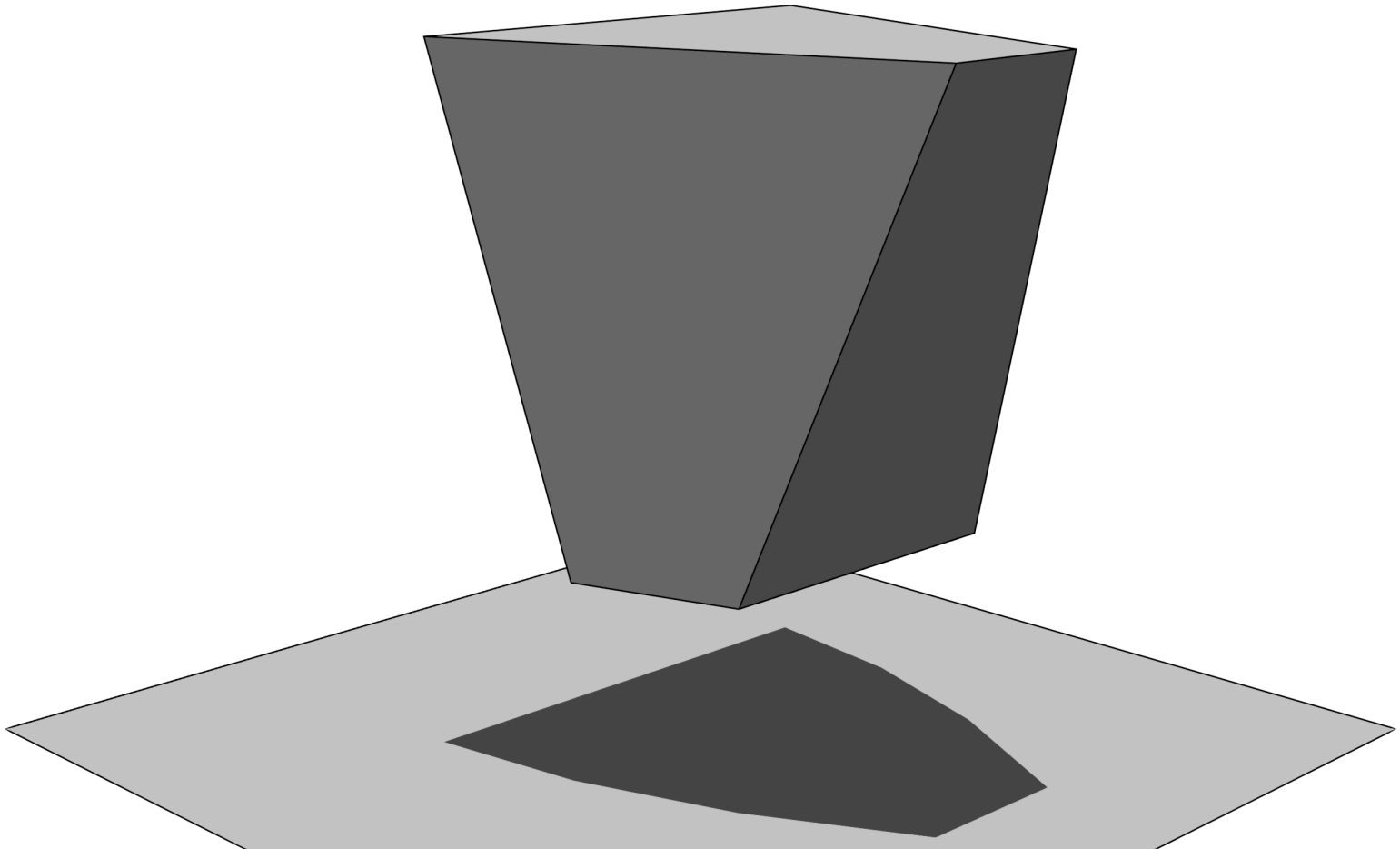
Simplex algorithm

1. Convert the LP to standard form
   - Turn the LP's constraint equation into $Ax = b$ instead of $Ax \leq b$, by introducing "slack variables"
2. Identify an initial "basic feasible solution" (BFS)
   - Typically we use the newly introduced slack variables in the initial BFS
3. Create a "simplex tableau"
   - Essentially a table indicating which variables are "basic" (in the "basis") and which are not
4. Iteratively update the tableau, with entering and leaving variables
   - This reflects moves along the edges of the feasible region from vertex to vertex
5. Terminate
   - We are done when we can't improve the objective value any more

Limitations of simplex

Simplex "usually" runs in polynomial time, but can run in exponential time in the worst case (Klee, V. and Minty, G.J., 1972)

Klee-Minty LP: adversarially constructed LP for which the simplex algorithm will take exponential time proportional to problem size:



Ellipsoid method

In the 70's Yudin and Nemirovki and Shor (independently) discovered the ellipsoid method, a less efficient (in general, around $O(n^8)$) but guaranteed polynomial time method of solving LPs

Input: Matrix A, vector b; sphere $S(c_0, R)$ containing P

Output: feasible $\overline{x}$ or proof that $P = \emptyset$

// Initialize

1: $k := 0$

2: $k^* = 2n(2n+1) < C > +2n^2(\log(R) - n^2 + 1)$

// Max number of iterations

3: $x^0 = c_0$, $B_0 = R^2 \cdot I$ // Initial ellipsoid

4: $\tau := \frac{1}{n+1}$ // Parameter for ellipsoid: step

5: $\sigma := \frac{2}{n+1}$ // Parameter for ellipsoid: dilation

6: $\delta := \frac{n^2}{n^2-1}$ // Parameter for ellipsoid: expansion

// Check if polytope P is empty

7: **if** $k = k^*$ **return** $P = \emptyset$

// Check feasibility

8: **if** $A\,x^k \leq b$ **return** $\overline{x} := x^k$ // $\overline{x}$ is a feasible point

9: **else** let $a_j^{\mathrm{T}} x^k > b_j$

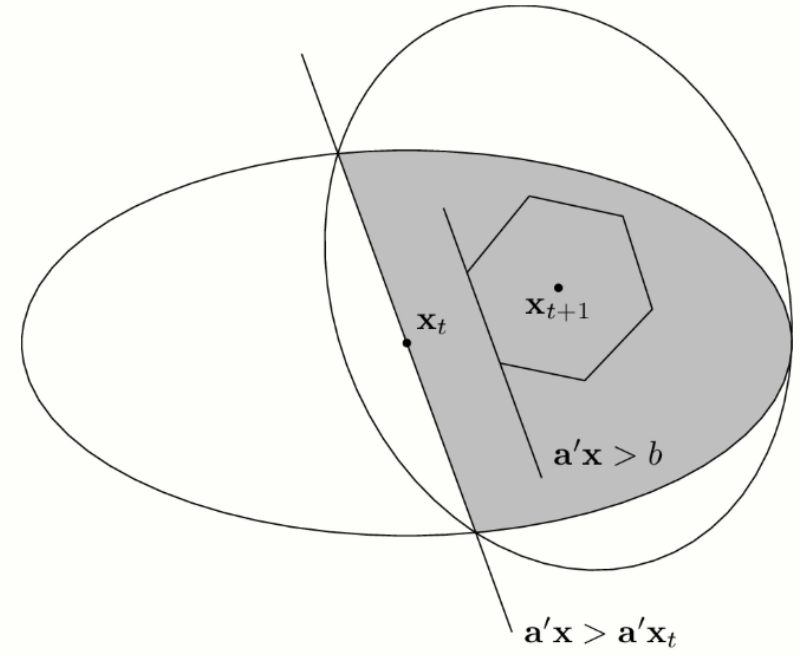
// Construct new ellipsoid

10: $B_{k+1} := \delta(B_k - \sigma \frac{B_k a_j (B_k a_j)^{\mathrm{T}}}{a_j^{\mathrm{T}} B_k a_j})$

11: $x^{k+1} := x^k - \tau \frac{B_k a_j}{\sqrt{a_j^{\mathrm{T}} B_k a_j}}$

// Loop

12: $k \leftarrow k+1$ and goto step 7

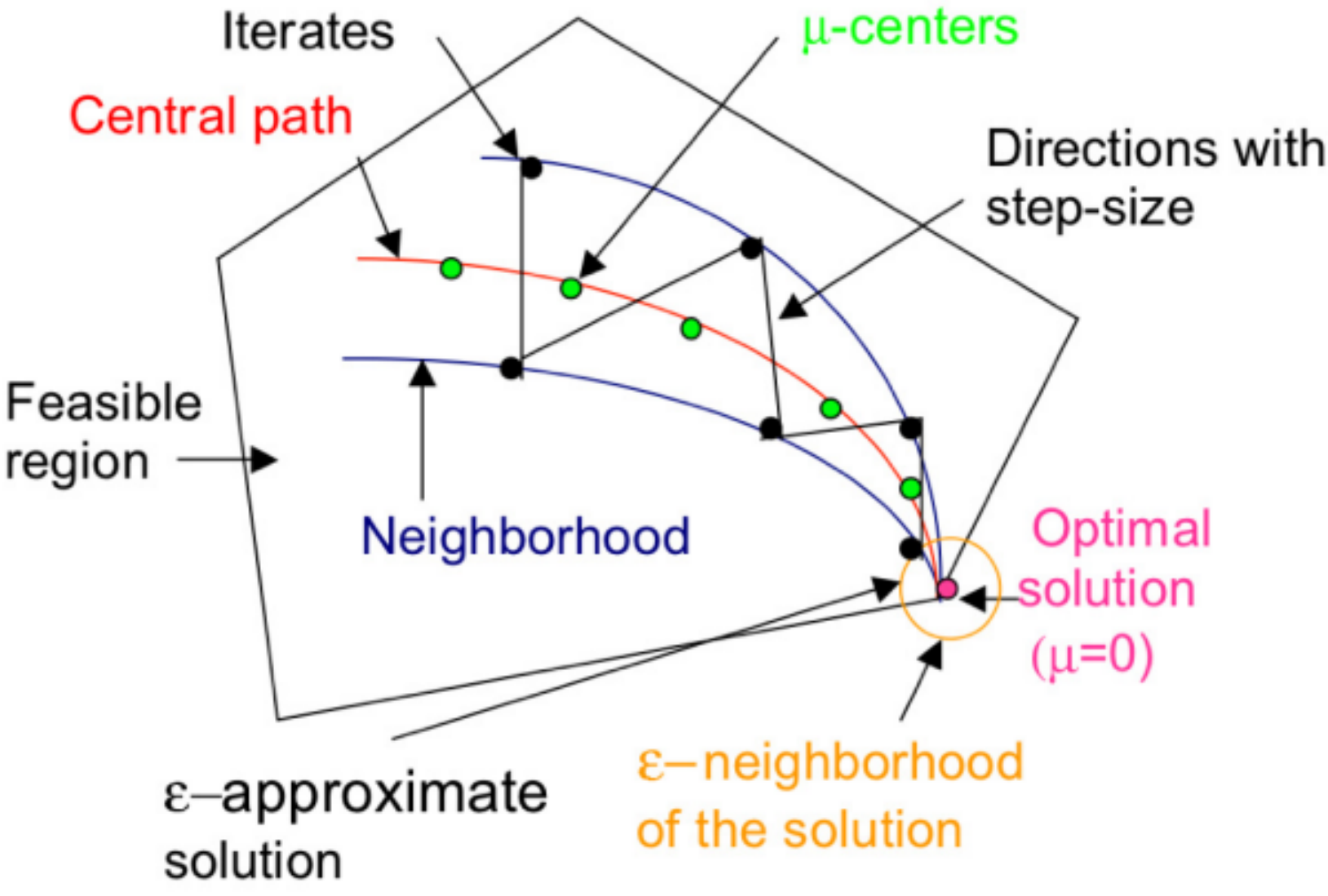


Interior point methods

Interior point methods: both guaranteed polynomial time and efficient in practice

- 1967: Dinkin (Soviet mathematician) discovered interior point methods (generally)
- 1980's: Interior point methods reinvented in the US
- 1984: Karmarkar develops Karmarkar's algorithm for linear programming using interior point methods, which runs in $O(n^{3.5}L)$ time for L-bit numbers and is efficient in practice

Interior point methods (diagram)



Interior point methods (general algorithm)

1. Pick some point in the feasible region
   - Different from simplex, where we wanted to choose one of the vertices of the feasible region
   - In interior point methods, we want to choose a point that is in the "middle" of the region instead
2. Determine the direction that gets us closest to the the objective value
   - Typically done using Newton's method if we are doing general convex optimization; for linear programming can be simpler
3. Combine the direction with the impact of the barrier function(s)
   - Barrier functions typically incorporate some log term so that as we get closer to the edges of the feasible region, the "repelling force" increases exponentially
   - This prevents us from reaching the boundary of the feasible region too quickly, which would give us a suboptimal solution
4. Terminate when we're sufficiently close to the boundary (and therefore the optimal solution)

In practice

Simplex is very fast most of the time (except for when it isn't), and interior point methods are generally pretty fast and more consistent

Solvers may run both simplex and some interior point method method in parallel on different cores, and take the result of whichever method terminates first

Top commercial solvers (LP and ILP): CPLEX, Gurobi, MOSEK

Open source alternatives: GLOP (Google, used by OR-Tools), GLPK (GNU), HiGHS, SCIP

References

Klee, V. and Minty, G.J. (1972) How Good Is the Simplex Method. In: Shisha, O., Ed., Inequalities III, Academic Press, New York, 159-175.

Yudin, D.B. and Nemirovski, A.S. (1976). Informational complexity and effective methods of solution for convex extremal problems. Ekonomika i Matematicheskie Metody, 12, 357–369.

Shor, N. Z. (1977). Cut-off method with space extension in convex programming problems. Cybernetics, 13(1), 94–96.

I. I. Dikin, Sov. Math., Dokl. 8, 674–675 (1967; Zbl 0189.19504); translation from Dokl. Akad. Nauk SSSR 174, 747–748 (1967)

N. Karmarkar. 1984. A new polynomial-time algorithm for linear programming. In Proceedings of the sixteenth annual ACM symposium on Theory of computing (STOC '84). Association for Computing Machinery, New York, NY, USA, 302–311. https://doi.org/10.1145/800057.808695