Linear Programming

The definitions of LP, and other pieces of the material appear in CLRS
Chapter 29

The linear-time algorithm for LP in 2D from MMOM

Slides courtesy of Craig Gotsman

Linear Programming - Example

Define: (amount amount consumed per day)

- \( i \) – types of foods (1 \( \leq \) \( i \) \( \leq \) \( d \)).
- \( j \) – types of vitamins (1 \( \leq \) \( j \) \( \leq \) \( n \)).
- \( x_i \) – the amount of food of type \( i \) consumed per day).
- \( a_{ji} \) – the amount of vitamin \( j \) in one unit of food \( i \).
- \( c_i \) – the number of calories in one unit of food \( i \).
- \( b_j \) – minimal required amount of vitamin \( j \).

Constraints (we need to consume some minimal amount of each vitamin):

\[ \begin{align*}
    \sum_{i=1}^{d} a_{ji} x_i & \geq b_j \\
    c_i x_i & \leq c_i
\end{align*} \]

Minimize: the total number of calories consumed:

\[ \sum_{i=1}^{d} c_i x_i \]

Subject to: Ax = b

More Geometry

The solution to the linear program is a point in the feasible region that is
extreme in the direction of the target function.

Theorem: Any bounded linear program that is feasible has a solution, which is a
vertex of the feasible region.

Proof: Convexity ...

Degenerate Cases

The feasible region may be:

- Empty
- Unbounded

The solution may be:

- Not unique

The Simplex Algorithm

Assume WLOG that the cost function points "downwards".

Construct (some of) the vertices of the feasible region.

Walk edge by edge downwards until

reaching a local minimum (which is also a
global minimum).

In \( \mathbb{R}^d \), the number of vertices might be \( \Theta (n^{\lceil d/2 \rceil}) \).
LP History

- Mid 20th century: Simplex algorithm, time complexity \( \Theta(n^4) \) in the worst case.
- 1980’s (Khachiyan) ellipsoid algorithm with time complexity \( \text{poly}(n,d) \).
- 1980’s (Karmakar) interior-point algorithm with time complexity \( \text{poly}(n,d) \).
- 1984 (Megiddo) – parametric search algorithm with time complexity \( O(C_d n^3) \) where \( C_d \) is a constant dependent only on \( d \).
- The holy grail: An algorithm with complexity independent of \( d \).
- In practice the simplex algorithm is used because of its linear expected runtime.

O(n log n) 2D Linear Programming

- Input:
  - \( n \) half planes.
  - Cost function that WLOG “points down”.
- Algorithm:
  1. Partition the \( n \) half-planes into two groups.
     - \( S \) are all half-planes contain the point \((0, \infty)\)
     - \( S' \) all other halfplanes contain the point \((0, -\infty)\)
  2. Sort them by slopes
  3. Compute the upper envelop \( U(S) \) and the lower envelop \( L(S') \)
  4. (using question from hw1)
  5. Scan simultaneously from left to right, and Computer intersection of two envelopes - they can intersect only at 2 points (why).
  6. Evaluate cost function at each vertex.

O(n^2) Incremental Algorithm

- The idea:
  - Start by intersecting two half-planes.
  - Add half-planes one by one and update optimal vertex by solving one-dimensional LP problem on new line if needed.

Incremental Algorithm - Symbols

- \( h_i \) the \( i \) half plane
- \( l_i \) the line that defines \( h_i \)
- \( C_i \) the feasible region after \( i \) constraints
- \( v_i \) the optimal vertex of \( C_i \)

Cost function to minimize: \( cx + py \)
Returns the leftmost point in feasible region

Incremental Algorithm

- Theorem:
  1. If \( v_{i-1} \in h_i \) then \( v_i = v_{i-1} \) // O(1) check, nothing to do
  2. If \( v_{i-1} \notin h_i \) then either
     - \( C_i \neq 3 \) // terminate
     - \( C_i = C_{i-1} \cap h_i \) and \( v_i \) lies on \( l_i \) // run 1D LP
- Proof:
  1. Trivial. Otherwise \( v_i \) would not have been optimum before.

Basic Theorem - Cont.

2. Assume that \( v_i \) is not on \( l_i \). \( v_i \) must be in \( C_{i+1} \). By convexity, also the segment \( v_{i-1}v_i \) is in \( C_{i+1} \).

Consider point \( v_i \) - the intersection of \( v_{i-1}v_i \) with \( l_i \). \( v_i \) is in both \( C_{i+1} \) and \( C_i \) and is better than \( v_i \).

Contradiction.
Finding \( v_i \) given \( l_i \)
(one-dimensional LP)

- Intersect each \( h_j \) \((j<i)\) with \( l_i \), generating \( i-1 \) rays representing (unbounded) intervals.
- Intersect the \( i-1 \) intervals in \( O(i) \) time.
- If the intersection is empty then report no solution, else report the lowest point.

Incremental Algorithm – \( O(n) \)
Randomized Version

- Exactly like the deterministic version, only the order of the lines is random.
- **Theorem**: The expected runtime of the random incremental algorithm (over all \( n! \) permutations of the input constraints) is \( O(n) \).

Probability Analysis

- **Question**: When given a solution after \( i \) half-planes, what is the probability that the last half-plane affected the solution?
- **Answer**: Exactly \( 2/i \), because a change can occur only if the last half-plane inserted is one of the two halfplanes thru \( v_i \), (note that \( v_i \) depends on the \( i \) halfplanes, but not on their order).

Complexity Analysis

- **Theorem**:
  \[
  T(n) = \sum_{i=1}^{n} O(i) = O(n^2)
  \]
Just to Make Sure …

- **False Claim:**
  - The probabilistic analysis is for the average input. Hence there exist bad sets of constraints for which the algorithm’s expected runtime is more than \( O(n) \), and there exist good sets of constraints for which the algorithm’s expected runtime is less than \( O(n) \).

- **True Claim:**
  - The probabilistic analysis is valid for all inputs. The expected complexity is over all permutations of this input.