Computational Geometry

Chapter 4

Linear Programming

Linear Programming - Example

Define:
- \( i \) – 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 \( i \).
- \( a_{ji} \) – 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):

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

Minimize: \( \sum_{i=1}^{d} c_i x_i \)
Subject to \( \mathbf{Ax} \geq \mathbf{b} \)

Linear Programming – The Geometry

- Each constraint defines a half-space region in \( d \)-dimensional space.
- The feasible region is the (convex) intersection of these half-spaces.

- We will treat the case \( d = 2 \), where each constraint defines a half-plane.

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 unique 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

On the Agenda

- Linear programming
- Duality
- Smallest enclosing disk
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 R\(^d\), the number of vertices might be \(\Theta(2^{\frac{d}{2}})\).

LP History

- Mid 20\(^th\) century: Simplex algorithm, time complexity \(O(n^{\frac{d-1}{2}})\) in the worst case.
- 1980’s (Khachiyan) ellipsoid algorithm with time complexity \(poly(n,d)\).
- 1980’s (Karmarkar) interior-point algorithm with time complexity \(poly(n,d)\).
- 1984 (Megiddo) – parametric search algorithm with time complexity \(O(C_d n)\) where \(C_d\) is a constant dependent only on \(d\). E.g. \(C_d = 2^{\frac{d^2}{2}}\).
- 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.
  2. Compute, recursively, the feasible region for each group.
  3. Compute the intersection of the two feasible regions.
  4. Check the cost function on the region vertices.

Divide and Conquer – Complexity Analysis

- Stage 3:
  - Intersection of two convex polygons – plane sweep algorithm.
  - No more than four segments are ever in the SLS and no more than eight events in the EQ – \(O(n)\).
- Stage 4:
  - Find the minimal cost vertex - \(O(n)\).

\[ T(n) = 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n) \]

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 \(i\)\(^th\) half-planes
- \(l_i\): the line that defines \(h_i\)
- \(C_i\): the feasible region after \(i\) constraints
- \(v_i\): the optimal vertex of \(C_i\)
Theorem:
1. if $v_{i-1} \neq h_i$, then $v_i = v_{i-1}$. // O(1) check, nothing to do
2. if $v_{i-1} = h_i$, then either
   $C = \emptyset$ // terminate
   or
   $C = C_{i-1} \cap h_i$ and $v_i$ lies on \( l_i \) // run 1D LP

Proof:
1. Trivial. Otherwise $v_i$ would not have been optimum before.

Finding $v_i$ given \( l_i \)
(one-dimensional LP)
1. Intersect each $h_j$ (\( j < i \)) with \( l_i \), generating \( i-1 \) rays representing (unbounded) intervals.
2. Intersect the \( i-1 \) intervals in $O(i)$ time.
3. If the intersection is empty then report no solution, else report the lowest point.

Complexity Analysis
$T(n) = \sum_{i=1}^{n} O(i) = O(n^2)$

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 $O(n)$.

Complexity Analysis
The expected runtime is:
$$\sum_{i=1}^{n} O(1)(1 - E(x_i)) + O(i)E(x_i) = O(n) + \sum_{i=1}^{n} [O(i)E(x_i)]$$
where $x_i$ is a random variable:
$$x_i = \begin{cases} 
1 & v_i \neq v_{i-1} \quad \text{// run 1D LP} \\
0 & v_i = v_{i-1} \quad \text{// do nothing}
\end{cases}$$
**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 \(i\) half-planes, but not on their order).

**Complexity Analysis**

\[
E(x_i) = \Pr(v_i \neq v_{i-1}) = \frac{2}{i}
\]

\[
O(n) = \sum_i O(i)E(x_i) = O(n) + O\left(\sum_i \frac{2}{i}\right) = O(n)
\]

**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.

**Smallest Enclosing Disk**

**Input:** \(n\) points.

**Output:** Disk with minimal radius that contains all the points.

**Theorem:** For any finite set of points in general position, the smallest enclosing disk either has at least three points on its boundary, or two points which form a diameter. If there are three points, they subdivide the circle into three arcs of length no more than \(\pi\) each. **Prove**!

This immediately implies a \(O(n^2)\) algorithm (why?)

**Basic Theorem**

**Theorem:** Using an incremental algorithm, where \(D_i\) is the updated disk after seeing the first \(i\) points \(p_1, \ldots, p_i\):
- If \(p \notin D_{i-1}\), then \(p\) is on the boundary of \(D_i\).

**Proof:**

Observation: If \(r < s\), then \(a < b\)

- If \(p \notin D_{i-1}\) then \(r < s\) => \(a < b\)
- If \(p \notin D_{i-1}\) then \(q_1, q_2 \notin D_{i-1}\)
  - \(\text{Arc}(q_1, q_2)\) => Contradiction.

**Incremental \(O(n)\) Expected Time Algorithm**

Construct the procedures:
- \(\text{MinDisk}(P)\) – find a smallest enclosing disk for a set of points \(P\).
- \(\text{MinDisk1}(P, q)\) – find an enclosing disk for a set of points \(P\) which touches point \(q\).
- \(\text{MinDisk2}(P, q_1, q_2)\) – find an enclosing disk for a set of points \(P\) which touches points \(q_1\) and \(q_2\).
- \(\text{Disk}(q_1, q_2, q_3)\) – find a disk thru points \(q_1, q_2, q_3\) (easy).
Incremental Algorithm

- MinDisk(P)

- $D_2$ = the minimal disk through $p_1$ and $p_2$.
- For each point $p_i$ in random order (3≤i≤n):
  - If $p_i$ ∈ $D_2$, then $D_i = D_2$. // do nothing
  - Else $D_i$ = MinDisk1($P_{i-1}$, $p_i$). // look for other two points on disk
- Return $D_n$

Incremental Algorithm

- MinDisk1(P,q)

- $D_1$ = the minimal disk through $q$ and $p_1$.
- For each point $p_i$ (2≤i≤n):
  - If $p_i$ ∈ $D_1$, then $D_i = D_1$. // do nothing
  - Else $D_i$ = MinDisk2($P_{i-1}$, $p_i$). // look for other one point on disk
- Return $D_n$

Incremental Algorithm

- MinDisk2(P,q1,q2)

- $D_0$ = the minimal disk through $q_1$ and $q_2$.
- For each point $p_i$ (1≤i≤n):
  - If $p_i$ ∈ $D_0$, then $D_i = D_0$. // do nothing
  - Else $D_i$ = Disk($q_1$, $q_2$, $p_i$). // form disk
- Return $D_n$

Complexity Analysis

- Use backward analysis on point ordering.
- Total time complexity:
  \[
  \sum_{i=1}^{n} \frac{3}{i} = O(n)
  \]
- Linear expected runtime.
- Worst case: $O(n^2)$. 