Linear Programming

- Optimization over $x=(x_1,\ldots,x_n)$
- Solve:

  maximize $\sum_j x_j c_j$
  subject to $\sum_j x_j a_{ij} \leq b_i$ for $i=1\ldots m$
  and $x_j \geq 0$

- Today: algorithms for solving LP
  - LP recap
  - Randomized incremental algorithm
  - Overview of other algorithms
Linear Programming in 2D

- Variables: \((x_1, x_2)\)
- Constraints: \(a_{i1}x_1 + a_{i2}x_2 \leq b_i\)
- Opt. direction: \((c_1, c_2)\)
- Opt. solution?
- Must lie on the boundary of the feasible region
  - Vertex optimum
  - Edge optimum (next slide)
- Other cases?
Edge optimum
An Infeasible Linear Program
An Unbounded LP
Types of LPs

• Generic case: vertex optimum
  – Want to find it

• “Pathological” cases:
  – Optimal edge
  – Unbounded LP
  – Infeasible

• In this lecture, we assume vertex optimum
How can we efficiently solve the LP?
Incremental algorithm
Adding a New Constraint
Adding a New Constraint
Adding a New Constraint
Adding a New Constraint
Algorithm

• Choose two constraints and initialize the solution

• For each new constraint:
  – If the current solution is not feasible, find optimum on new constraint (line)

• Analysis ?
Initialization

- Choose two constraints and initialize the solution
  - A simple hack: impose additional "box constraint" $x \in [-M,M]^2$ for $M$ “large enough”
    (possible to make it formal, but here we leave it as is)
  - Then one can just take a corner as the initial solution
    - $O(1)$ time
- If the current solution is not feasible, find optimum on new constraint
  - How much time does this take?
Analysis

• If we encounter the constraints in the bottom-up order, we need to update the solution every time
  – Time $= \Theta(\sum i_i) = \Theta(m^2)$
• If we encounter them in the top-down order, we never need to update the solution
  – Time $= \Theta(m)$
• How to ensure “good” order?
• Consider the constraints in random random order!
  – Each permutation equally likely
Expected time spent updating

- Let $T_i$ be the time spent at time $i$
  - If we need an update, time $= O(i)$
  - If no need to update, time $= O(1)$
- We have

\[
E[\sum_i T_i] \\
= \sum_i E[T_i] \\
= O(m) + \sum_i P(\text{update at round } i)O(i)
\]
Probability of update at round $i$

- Consider the set of the first $i$ constraints in the random order (excluding the initial box constraints)
  - Those constraints still arrive in a random order
    - Each permutation of $i$ constraints equally likely
- Consider the solution $v$ for these $i$ constraints (defined by two constraints)
- Update occurs in the $i$th step only if the $i$th constraint is one of the two defining constraints
- The probability of that is at most $\frac{2}{i}$
Expected Run-Time Analysis

- Expected time spent updating:

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

- Linear time (for \( n=2 \))
- What about \( n>2 \) ?
What about dimension > 2?

- Incrementally add new constraints
- Probability of update: \( n/i \)
- On update: solve \( n-1 \) dimensional LP

\[
T(m,n) \leq O(mn) \sum_{i=1}^{m} \frac{n}{i} T(i-1,n-1) \leq O(n!m)
\]
Other algorithms

• Weakly polynomial-time algorithms:
  – Ellipsoid algorithm
  – Interior point algorithm
  – …

• Assume coefficients are integers of absolute value < L

• Running time polynomial in m, n and log L

• Reason: algorithms are *iterative*
Ellipsoid algorithm

- Solves feasibility, i.e., whether there exists $x$ such that $Ax \leq b$
  - Add $cx \geq T$
  - Binary search on $T$
- Find an initial ellipsoid that contains the feasible region
- Iterate to find smaller and smaller ellipsoids
$x(0)$

Credit: Steven Boyd, via Wikipedia
Conclusions

• Algorithms for LP
  – Simplex, incremental
  – Weakly polynomial

• Open problem: strongly polynomial algorithm?