Combinatorial optimization, Decision trees

6.100 LECTURE 16

SPRING 2026

Announcements

- **Exam 2 next Monday 4/6 in Walker**
  - announcement email sent last night
- Make use of office hours!

Discrete Optimization

Recall framework of optimization

- Examples
  - Alyssa/Ben/Cindy **sell tickets**
  - **linear regression,** minimize squared error
- Problem structure
  - define **variables** of interest and their domains
  - satisfy **constraints** over them
  - minimize/maximize an **objective function** over them
- Special cases
  - **no constraints** → unconstrained optimization
  - **no objective** → constraint satisfaction problem

Solving optimization problems

- **Continuous variables**
  - can often consider **derivatives** of constraint and objective
  - typical strategy
    - guess an initial point (i.e., assignment to all variables)
    - evaluate derivatives
    - if infeasible, take a step towards feasible region
    - otherwise, take a steps towards more optimal
- **Discrete variables**
  - harder to step, not guaranteed to land on a valid assignment
  - typical strategy is based on **exhaustive enumeration**
    - generate all possible combinations of variable assignments
    - check each against constraints and objective
    - many clever ways to **check during generation**

Combinations of items

- Optimization framework is extremely general
  - many **subdisciplines** are dedicated to specific forms of constraints, and objectives
  - e.g., linear, convex, Boolean logic, higher-order logic, mixed discrete-continuous, stochastic
- Today and next week: consider the narrower situation of finding **combinations of items** of interest, i.e., finding **subsets** of a collection of items
- **Exhaustive enumeration** strategy
  - generate all possible combinations
  - evaluate them against constraints and objective
  - **today: ignore objective,** focus on generation and a basic constraint

Generating all combinations

- Pre-lecture code generated **some possible combination** with recursive strategy
  - decompose items into **first** and **rest**
  - recursively get **some combination** out of **rest**
  - stochastically decide whether to include **first**
- Generating the **entire power set**
  - still decompose items into **first** and **rest**
  - recursively get ***all combinations*** out of **rest, don't include first**
    - all are valid subsets of original
  - recursively get ***all combinations*** out of **rest** again, **include first** this time
- Given a set of $\boldsymbol{n}$ elements, power set has $\boldsymbol{2^n}$ subsets
  - on modern processors, for $\boldsymbol{n > \sim 25}$, becomes increasingly impractical to generate entire power set

Taking control of the recursion

- Consider the task of generating **all combinations of a certain size**
- Plain exhaustive enumeration
  - get all subsets from **`powerset()`**
  - keep only the ones with a given size (i.e., **`len()`**)
- Wastes time generating combinations of larger size
  - e.g., if you want all combinations of size 3, why bother building combinations of size 4, 5, 6, …
- Recursive process of powerset() is inherently a **depth-first search (DFS)-like** process over a **"tree graph" of subproblems**
  - if we had more control over the recursion, we could **cut off exploring** certain tree branches/subtrees

Decision tree with pruning

- **Idea:** include additional information in the recursive problem definition
  - still need **remaining items**
  - but also specify **remaining size of subset** you're looking for
- **Base case:** no more items
  - if looking for subset of size 0 (empty set), the empty set is the only valid subset
  - if still need a non-empty subset, then no valid subsets
- **Pruning** as an **early base case**
  - when there are still items, but remaining size has reached 0, no need to further explore subtree
- Additional **pruning when branching**
  - if all remaining items cannot possibly fill remaining size we're looking for, then no valid subsets

Next time

- **Today**
  - use recursion to generate all possible combinations of interest
  - prune infeasible branches within decision tree to save computation
  - practiced some list comprehensions along the way
  - enough to prepare you for **Pset 5**
- **Next Wednesday**
  - reuse computation within decision trees to save even more computation
  - what **Pset 6** will be based on