Decision trees, Knapsack Problem

6.100 LECTURE 17

FALL 2026

Pruning Decision Trees

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

Decisions and pruning

- **Task 1:** generating **all combinations** of a collection of items
  - model **`powerset()`** using a decision tree
- **Task 2:** generating **all combinations of a certain size**
  - exhaustive enumeration: filter output of **`powerset()`**
  - **pruning:** keep information about remaining size, don't explore unnecessary branches

Knapsack Problem

Some common optimization problems

- **Knapsack**
  - Task: fill a bag with subset of available items
  - Objective: maximize total value
  - Constraints: stay under bag's weight limit
  - Applications: packing delivery trucks, managing power allocation, election flipping
- **Bin packing**
  - Task: minimize number of containers needed to ship items
- **Edit distance**
  - Task: change one sequence of letters into another sequence
  - Objective: minimize total cost of edit actions
  - Constraints: available actions (insert, delete, substitute)
  - Applications: tracking DNA mutations, file compression and error correction

Knapsack Problems



Also known as a rucksack, haversack, backpack

- You have limited strength, so there is a maximum weight knapsack that you can carry
- You want to take more stuff than you can carry
- How do you choose what to take vs. leave behind?
  - want to optimize "value" of things to take
- Two variants
  - continuous or fractional knapsack problem
  - discrete or 0/1 knapsack problem

Straightforward

Much more interesting

Quanta are infinitesimal relative to available space



versus



Quanta are large relative to available space

Example: Participatory Budgeting

- Cambridge has annual cycles of participatory budgeting
  - https://pb.cambridgema.gov/pbcycles
- Set aside about $1 million
- Release proposed list of projects, each with a cost
- Collect votes from residents, score each project on collective benefit

Winning PB projects	Planning/Design	Procurement of Supplies/ Equipment	Installation
Trees for Danehy Park and Cambridge	✓	Locations for plantings have been identified. DPW is waiting until digging can take place at the park.	
Technology for Youth Centers	Human Services is planning the distribution process for technology in city youth centers.		
Look and Listen: Safer Crossing for Cambridge	The Department of Transportation is planning for installation and identifying locations for safer crosswalks.		
Smart Recycling and Trash Compactors	✓	Public Works is in the process of ordering the new recycling and trash compactor units.	
Electric Vehicle Charging Stations	The City is identifying locations with highest demand and planning for installation of EV charging stations.		
E-Cargo Bikes for Watering Trees	✓	✓	✓
More Outdoor Public Wi-fi	The IT Department is planning for installation of necessary wifi in Clement Morgan Park.		
Three New Public Art Murals	✓ Planning for the mural in the Port is underway Planning for the mural in Central Square is underway	✓	✓

Solving knapsack

- **Approach 1**
  - enumerate from powerset()
  - filter to meet capacity constraint
  - maximize total value of selected items
- **Approach 2**
  - prune decision tree
  - if choose current item would exceed the remaining capacity, do not explore that branch

Runtime performance of enumeration

- $\boldsymbol{2^n}$ combinations, grows exponentially in number of items
  - $2^{10} \sim 10^3$, $2^{20} \sim 10^6$, $2^{30} \sim 10^9$
  - Processor speeds $\sim 10^9$ Hz (cycles per second)
  - $\sim 10^3$ cycles to process each combination
- Solution times become user-unfriendly around $\boldsymbol{n \sim 25}$
  - Each increment in $\boldsymbol{n}$ will **double** the size of the solution space to consider
  - Not scalable
  - Are we doomed to rely on greedy solutions?



Pruning branches in decision tree

- While generating the combinations, keep track of remaining capacity as we go descend branches in the decision tree
- If taking the next object would cause our selection to go over capacity, do not consider that branch
- Testing result
  - Can solve up to $\boldsymbol{n \sim 35}$ in less than a minute
  - Better, but still not practical
- Exercise
  - Decision-tree implementation makes list copies when slicing
  - Rewrite so that it recurses on ***index of next item*** to consider, rather than on list of remaining items

