Knapsack problems, Dynamic programming

6.100 LECTURE 18

SPRING 2026

Announcements

- **Exam 2 scores** to be released tomorrow
- **Pset 5** due next Monday 4/13
- **AI-generated hints** on finger exercises
  - announcement tomorrow

Review
Decision Trees

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

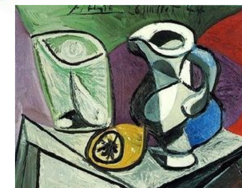
- You have limited strength, so there is a maximum weight knapsack that you can carry
- You want to take more stuff than you can carry
- How do you choose what to take vs. leave behind?
  - want to optimize "value" of things to take
- Two variants
  - continuous or fractional knapsack problem
  - discrete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

- $\mathbf{2^n}$ 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 $n\sim\mathbf{35}$ 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

Dynamic Programming

Avoid re-solving overlapping subproblems

- Consider items $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}, \ldots$ with weights $\mathbf{2}, \mathbf{3}, \mathbf{2}, \mathbf{1}, \ldots$ and capacity $\mathbf{20}$
- After going down the branch $+\boldsymbol{A}, +\boldsymbol{B}, -\boldsymbol{C}$, need to solve a subproblem for items $\boldsymbol{D}, \boldsymbol{E}, \ldots$ with capacity $\mathbf{15}$
- After going down the branch $-\boldsymbol{A}, +\boldsymbol{B}, +\boldsymbol{C}$, need to solve the **exact same subproblem!**
- ***Idea (dynamic programming)***
  - Remember the result the first time
  - Recognize if we re-encounter, use saved result

Dynamic programming

- Broadly applicable algorithm strategy
  - Principles were invented by Richard Bellman in the 1950s
  - Name was chosen for "marketing"
    - "programming" in the sense of "creating a schedule/solution"
    - "dynamic" because it sounds cool
    - https://en.wikipedia.org/wiki/Dynamic_programming#History_of_the_name
- **Correctness requirement: optimal substructure**
  - optimal solutions involve optimal solutions to subproblems
- **Efficiency requirement: overlapping subproblems**
  - different subproblems rely on the same subproblems
  - how efficient depends on amount of subproblem overlap

Memoization approach

- When solve any subproblem for the first time, store its solution in a memo
- When solving any subproblem again, look up its solution in the memo
- Memo can be implemented as a **`dict`,** map problem spec to solution
  - in Python dicts, the problem spec needs to be hashable
- Use recursion to build the memo **"top-down"**
- **Pro:** small implementation change, matches top-down / DFS exploration order decision tree
- **Con:** overhead of recursion

Tabularization approach

- Define the entire space of problem specs as a **grid/table**
  - in Python, typically lists, so indices represent the problem parameters
- Iterate through the the table “bottom-up”, solving and storing the answer to each subproblem
- By the time a solution needs the solution to a particular subproblem, the subproblem is guaranteed to have been solved
- **Pro:** can run faster, no recursion overhead
- **Con:** may solve unnecessary subproblems

Next time

- **Last week and today**
  - many optimization problems come down to finding a best solution out of a combinatorial solution space
  - decision trees provide structure to exploring that space and offer opportunities for pruning and reusing computation for overlapping subproblems
- **Next week**
  - more Python: custom object types
  - object-oriented programming
  - **Pset 6** will be solving knapsack problems using object-oriented code