Data Structures and Algorithms

Lecture 8: Hard problems

Heikki Peura

h.peura@imperial.ac.uk

Previously

Graphs recap:

- Weighted/unweighted graphs
- Dijkstra and BFS

Plan for today:

- Lecture: Greedy algorithms and the knapsack problem
- Workshop
- 11:10am: Prize ceremony, parting words

Designing algorithms is tricky!

No single recipe solves all our problems...

- ...but there are some useful principles

Designing algorithms is tricky!

No single recipe solves all our problems...

- ...but there are some useful principles

We've used the **divide-and-conquer** paradigm

- Divide into smaller subproblems
- Eg binary search

Designing algorithms is tricky!

No single recipe solves all our problems...

- ...but there are some useful principles

We've used the **divide-and-conquer** paradigm

- Divide into smaller subproblems
- Eg binary search

But there are others:

- Greedy algorithms (today)
- Dynamic programming (for you to discover...)

Greedy algorithms

Rough idea: make **myopic** decisions iteratively

- Make a choice for immediate benefit without worrying about future consequences
- Attractive but dangerous

Greedy algorithms

Rough idea: make **myopic** decisions iteratively

- Make a choice for immediate benefit without worrying about future consequences
- Attractive but dangerous

Greedy algorithms are:

- Easy to come up with
- Easy to analyse running time
- Difficult to show to be correct
- **DANGER**: a greedy choice is often **not** correct

Knapsack problem

Problem: fill a bag with the most valuable items available.

Input:

- Set of n items, with values v_i and sizes w_i (integer)
- Capacity W

Knapsack problem

Problem: fill a bag with the most valuable items available.

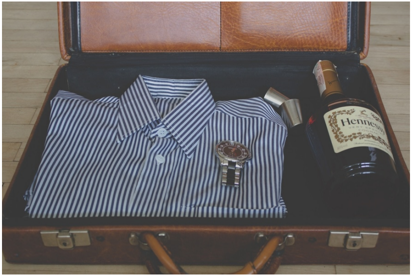
Input:

- Set of n items, with values v_i and sizes w_i (integer)
- Capacity W

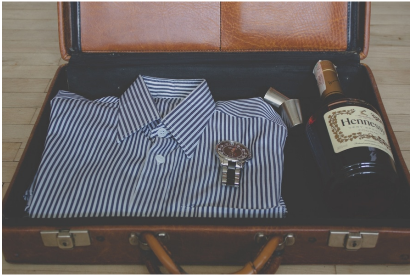
Output: subset S of items that maximizes the sum of values subject to a capacity constraint:

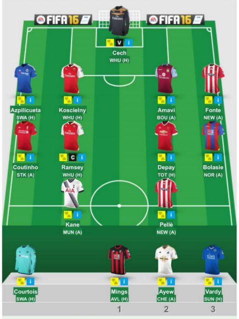
- $\max \sum_{i \in S} v_i$
- subject to $\sum_{i \in S} w_i \leq W$

Knapsack problem applications

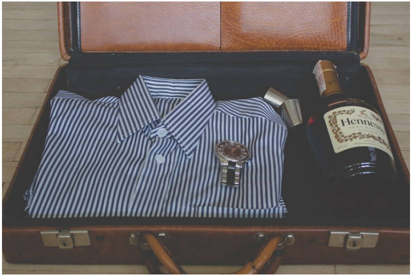


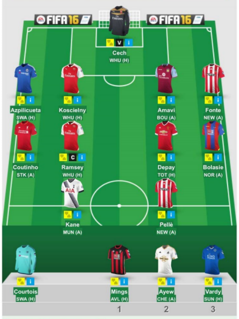
Knapsack problem applications





Knapsack problem applications





Many problems with budget constraints are versions of knapsack

- Selecting portfolios (eg projects to invest in)
- Lots of “operational” problems...

Two-item example

Items to pack

- **Shirt**: value 5, weight 5
- **Bottle**: value 10, weight 5

Two-item example

Items to pack

- **Shirt**: value 5, weight 5
- **Bottle**: value 10, weight 5

Subsets:

- {},{**Shirt**},{**Bottle**},{**Shirt**,**Bottle**}

Two-item example

Items to pack

- **Shirt**: value 5, weight 5
- **Bottle**: value 10, weight 5

Subsets:

- {},{**Shirt**},{**Bottle**},{**Shirt**,**Bottle**}

Knapsack size limits feasible solutions

- $W < 5$: can pick neither
- $5 \leq W < 10$: neither or just one
- $W \geq 10$: all subsets feasible

Go through all possibilities?

Input:

- Set of n items, with values v_i and sizes w_i (integer)
- Capacity W

Output: subset S of items that maximizes the sum of values subject to capacity constraint:

- $\max \sum_{i \in S} v_i$
- subject to $\sum_{i \in S} w_i \leq W$

Go through all possibilities?

Input:

- Set of n items, with values v_i and sizes w_i (integer)
- Capacity W

Output: subset S of items that maximizes the sum of values subject to capacity constraint:

- $\max \sum_{i \in S} v_i$
- subject to $\sum_{i \in S} w_i \leq W$

Exhaustive (brute-force) search?

- Go through all subsets of $\{1, 2, 3, ..., n\}$
- Suppose we have 50 items: $O(2^n)$ subsets...

Greedy approaches for knapsack problem

Input:

- Set of n items, with values v_i and sizes w_i
- Capacity W

Output: subset S of items that maximizes the sum of values subject to capacity constraint:

- $\max \sum_{i \in S} v_i$
- subject to $\sum_{i \in S} w_i \leq W$

Greedy approaches? — Pick myopically without worrying about future choices

Greedy approaches for knapsack problem

Input:

- Set of n items, with values v_i and sizes w_i
- Capacity W

Output: subset S of items that maximizes the sum of values subject to capacity constraint:

- $\max \sum_{i \in S} v_i$
- subject to $\sum_{i \in S} w_i \leq W$

Greedy approaches? — Pick myopically without worrying about future choices

- **Highest-value** item first?

Greedy approaches for knapsack problem

Input:

- Set of n items, with values v_i and sizes w_i
- Capacity W

Output: subset S of items that maximizes the sum of values subject to capacity constraint:

- $\max \sum_{i \in S} v_i$
- subject to $\sum_{i \in S} w_i \leq W$

Greedy approaches? — Pick myopically without worrying about future choices

- **Highest-value** item first?
- **Lowest-size** item first?

Greedy approaches for knapsack problem

Input:

- Set of n items, with values v_i and sizes w_i
- Capacity W

Output: subset S of items that maximizes the sum of values subject to capacity constraint:

- $\max \sum_{i \in S} v_i$
- subject to $\sum_{i \in S} w_i \leq W$

Greedy approaches? — Pick myopically without worrying about future choices

- **Highest-value** item first?
- **Lowest-size** item first?
- Some easy way of **combining value and size**?

Greedy algorithms for knapsack

Greedy approaches?

- **Highest-value** item first?

Greedy algorithms for knapsack

Greedy approaches?

- **Highest-value** item first?

Example: capacity $W = 20$, three items with values $v = [10, 10, 11]$, weights $w = [10, 10, 11]$.

- Picking highest value first is **bad**...

Greedy algorithms for knapsack

Greedy approaches?

- **Highest-value** item first?
- **Lowest-size** item first?

Example: capacity $W = 20$, three items with values $v = [10, 10, 11]$, weights $w = [10, 10, 11]$.

- Picking highest value first is **bad**...

Greedy algorithms for knapsack

Greedy approaches?

- **Highest-value** item first?
- **Lowest-size** item first?

Example: capacity $W = 20$, three items with values $v = [10, 10, 11]$, weights $w = [10, 10, 11]$.

- Picking highest value first is **bad**...

Example: capacity $W = 20$, three items with values $v = [10, 10, 50]$, weights $w = [10, 10, 11]$.

- Picking lowest weight first is **bad**...

Greedy algorithms for knapsack

Greedy approaches?

- **Highest-value** item first?
- **Lowest-size** item first?

Example: capacity $W = 20$, three items with values $v = [10, 10, 11]$, weights $w = [10, 10, 11]$.

- Picking highest value first is **bad**...

Example: capacity $W = 20$, three items with values $v = [10, 10, 50]$, weights $w = [10, 10, 11]$.

- Picking lowest weight first is **bad**...

Some easy way of **combining value and size**?

- Eg **sort items by unit weight** v_i / w_i and pick them in this order

Greedy algorithm for knapsack

Greedy algorithm:

- Sort items in decreasing bang-for-buck v_i/w_i
- Pick items until capacity full

Greedy algorithm for knapsack

Greedy algorithm:

- Sort items in decreasing bang-for-buck v_i/w_i
- Pick items until capacity full

Running time?

- Sorting?
- Loop?

Greedy algorithm for knapsack

Greedy algorithm:

- Sort items in decreasing bang-for-buck v_i/w_i
- Pick items until capacity full

Running time?

- Sorting?
- Loop?
- Total $O(n \log n)$

Is the algorithm correct?

Example: capacity W=510, three items with values $v = [10, 10, 500]$, weights $w = [10, 10, 501]$.

- Greedy picking is **bad**...

Is the algorithm correct?

Example: capacity W=510, three items with values $v = [10, 10, 500]$, weights $w = [10, 10, 501]$.

- Greedy picking is **bad**...

It would be optimal if we could **divide items into fractions**

Is the algorithm correct?

Example: capacity W=510, three items with values $v = [10, 10, 500]$, weights $w = [10, 10, 501]$.

- Greedy picking is **bad**...

It would be optimal if we could **divide items into fractions**

Here we would need a different approach to find correct solution — dynamic programming

- Exploit problem structure to loop through all items and possible capacities
- Correct solution, $O(nW)$ time ("pseudo-polynomial")

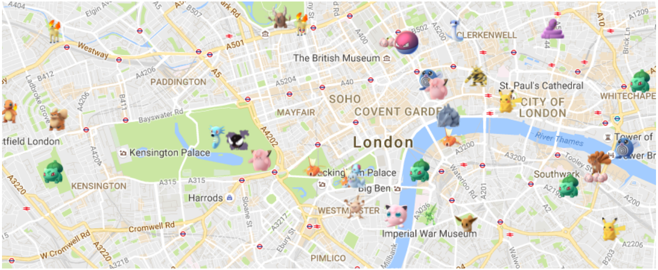
Many important problems are intractable

Tractable problem = Solvable in polynomial time $O(n^k)$ for some k

Many important problems are intractable

Tractable problem = Solvable in polynomial time $O(n^k)$ for some k

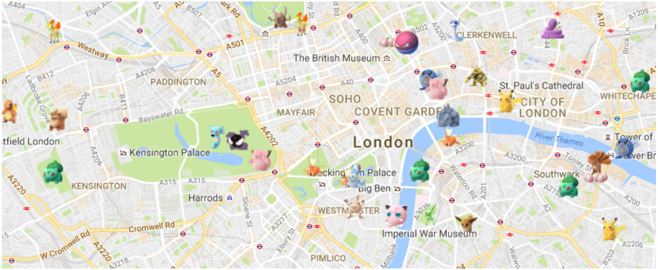
Finding the **shortest route to a Pikachu nest** (Dijkstra's algorithm) vs. finding the **shortest tour of all Pokemon nests**



Many important problems are intractable

Tractable problem = Solvable in polynomial time $O(n^k)$ for some k

Finding the **shortest route to a Pikachu nest** (Dijkstra's algorithm) vs. finding the **shortest tour of all Pokemon nests**

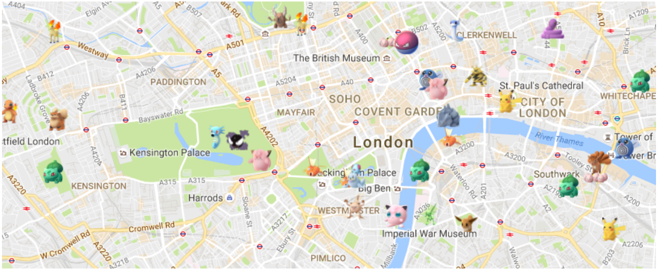


Example of intractability: **traveling salesperson problem** (TSP)

Many important problems are intractable

Tractable problem = Solvable in polynomial time $O(n^k)$ for some k

Finding the **shortest route to a Pikachu nest** (Dijkstra's algorithm) vs. finding the **shortest tour of all Pokemon nests**



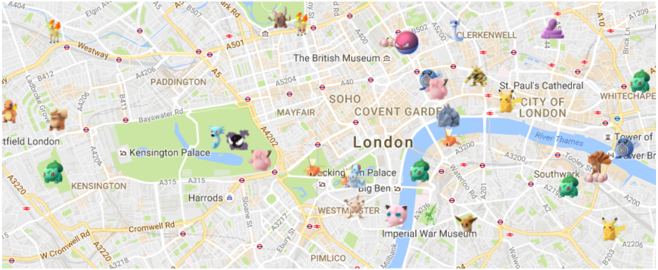
Example of intractability: **traveling salesperson problem** (TSP)

- **Input**: undirected graph with non-negative edge costs

Many important problems are intractable

Tractable problem = Solvable in polynomial time $O(n^k)$ for some k

Finding the **shortest route to a Pikachu nest** (Dijkstra's algorithm) vs. finding the **shortest tour of all Pokemon nests**



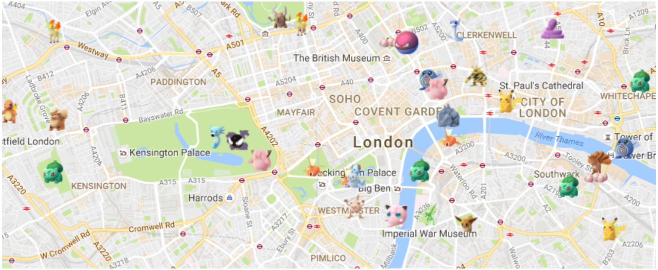
Example of intractability: **traveling salesperson problem** (TSP)

- **Input**: undirected graph with non-negative edge costs
- **Goal**: find minimum cost tour visiting every node

Many important problems are intractable

Tractable problem = Solvable in polynomial time $O(n^k)$ for some k

Finding the **shortest route to a Pikachu nest** (Dijkstra's algorithm) vs. finding the **shortest tour of all Pokemon nests**



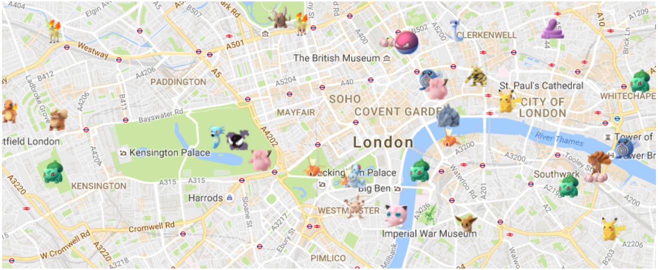
Example of intractability: **traveling salesperson problem** (TSP)

- **Input**: undirected graph with non-negative edge costs
- **Goal**: find minimum cost tour visiting every node
- **Conjecture**: no polynomial-time algorithm

Many important problems are intractable

Tractable problem = Solvable in polynomial time $O(n^k)$ for some k

Finding the **shortest route to a Pikachu nest** (Dijkstra's algorithm) vs. finding the **shortest tour of all Pokemon nests**



Example of intractability: **traveling salesperson problem** (TSP)

- **Input**: undirected graph with non-negative edge costs
- **Goal**: find minimum cost tour visiting every node
- **Conjecture**: no polynomial-time algorithm
- Many other important problems too...

My problem is intractable!

What can you do?

1. There may be **tractable special cases** (small knapsack DP)

My problem is intractable!

What can you do?

1. There may be **tractable special cases** (small knapsack DP)
2. Get an **approximate solution using heuristics** — fast but not “correct” (next slide)

My problem is intractable!

What can you do?

1. There may be **tractable special cases** (small knapsack DP)
2. Get an **approximate solution using heuristics** — fast but not “correct” (next slide)
3. Solve in exponential time but try to **improve on brute force** (large knapsack DP)

Greedy knapsack heuristic

Greedy knapsack was incorrect but very fast: $O(n \log n)$

Greedy algorithm:

- Sort items in decreasing bang-for-buck v_i / w_i
- For each item, pick the item until reach total W

Greedy knapsack heuristic

Greedy knapsack was incorrect but very fast: $O(n \log n)$

Greedy algorithm:

- Sort items in decreasing bang-for-buck v_i/w_i
- For each item, pick the item until reach total W

How bad is it?

- **Example**: capacity W=510, three items with values $v = [10, 10, 500]$, weights $w = [10, 10, 501]$.

Greedy knapsack heuristic

Greedy knapsack was incorrect but very fast: $O(n \log n)$

Greedy algorithm:

- Sort items in decreasing bang-for-buck v_i/w_i
- For each item, pick the item until reach total W

How bad is it?

- **Example**: capacity W=510, three items with values $v = [10, 10, 500]$, weights $w = [10, 10, 501]$.
- "Worst-case scenario": leave out (a single) extremely valuable object that would fit into knapsack

Better greedy knapsack heuristic

Modified algorithm: pick **either** the greedy solution **or** the most valuable item

Better greedy knapsack heuristic

Modified algorithm: pick either the greedy solution or the most valuable item

- Sort items in decreasing v_i/w_i
- For each item, pick the item until reach total W
- Return either greedy solution or the most valuable item, whichever is better

Better greedy knapsack heuristic

Modified algorithm: **pick either the greedy solution or the most valuable item**

- Sort items in decreasing v_i / w_i
- For each item, pick the item until reach total W
- Return either greedy solution or the most valuable item, whichever is better

Claim: This gets at least 50% of maximum possible value.

Better greedy knapsack heuristic

Modified algorithm: **pick either the greedy solution or the most valuable item**

- Sort items in decreasing v_i/w_i
- For each item, pick the item until reach total W
- Return either greedy solution or the most valuable item, whichever is better

Claim: This gets at least 50% of maximum possible value.

- "Worst-case scenario": greedy leaves out two extremely valuable objects that would both fit into knapsack
- Still pick the better of them: at least 50%
- Often items are small $\rightarrow$ the greedy choice cannot leave out many of them

Review

Some problems are inherently intractable

- Knapsack, traveling salesperson
- Greedy algorithms, approximations and heuristics can help

Workshop after the break

- Knapsack and fantasy football

11am: Parting words

- Hacker Challenge
- What next?

Workshop

Workshop zip file on the Hub

- HTML instructions
- At some point, you'll need the `.py`-file with skeleton code (open in Spyder)

11am: Parting words

- Hacker Challenge
- What next?