Scientific Computing

Wed, Feb 4

Announcements

* Homework 2 due Fri, Feb 13, 11:59pm
  pdf + zip file on D2L
  start early!!
  covers Greedy Algorithms

Don't forget to keep track of and cite any external resources you use - friends, websites, AI, etc.

* Friday - we will allocate half of class time for me troubleshooting your Python installations - bring your laptop

Office Hours:

Mon, 9:30-10:30
Fri, 2:00-3:00

Cudahy 307

Topic 5 - Search Spaces and Brute Force

Most of our problems can be summarized as:

"Out of all ways to do [blank]:

(1) Do any of them satisfy this list of constraints?

and/or

(2) Which one is optimal? "

"Out of all ways to do [blank]:
(1) Do any of them satisfy this list of constraints?
and/or
(2) Which one is optimal? "

Greedy Algos gave us a quick way to get a [blank] that might be decent, but in most cases is not at all guaranteed to be optimal.

They don't check every [blank] - in fact they only check a single one!

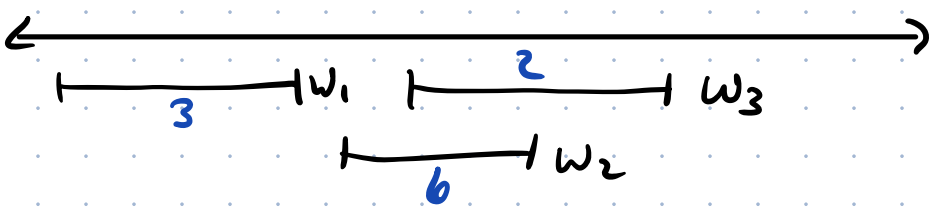
The search space of a problem is the set of all possible "things" that may or may not satify your constraints and that all have some score that you want to minimize or maximize.

The next few lectures are focused on ways to actually check the entire search space to find the optimal solution.

The most obvious way to do this is brute force: generate every single element of the search space, and check whether it satisfies the constraints and if so, what its score is.

Ex 1: Weighted Interval Scheduling

3 requests



Search space: all subsets of $\{w_1, w_2, w_3\}$

candidate	satisfies constraints	score
$\{\}$	✓	0
$\{w_1\}$	✓	3
$\{w_2\}$	✓	6
$\{w_3\}$	✓	2
$\{w_1, w_2\}$	✓	9
$\{w_1, w_3\}$	✓	5
$\{w_2, w_3\}$	X	8
$\{w_1, w_2, w_3\}$	X	11

Fact: There are 2^n subsets of a set of size n.

b = "best score we've seen so far"

Pseudocode

```
R = set of requests
b = 0
best_sol = None
for each subset r of R:
    if r is valid:
        s = score(r)
        if s > b:
            b = s
            best_sol = r
return (b, best_sol)
```

loop over every element of the search space

loops 2^n times
validity check takes $O(n)$
scoring takes $O(n)$

total time $\sim 2^n \cdot (2n) = O(n2^n)$.

↳ returning best score

Knapsack: Same situation. n items

Search space = all subsets of those n items

size is 2^n again

Closest Pair: Input n points in the xy-plane

Goal: Find the closest pair (normal Euclidean distance).

Search space = all unordered pairs of distinct points

$p_1 = (x_1, y_1)$

$p_2 = (x_2, y_2)$

$d(p_1, p_2) = d(p_2, p_1)$

$n = 3$ $\{p_1, p_2, p_3\}$

$\{(p_1, p_2), (p_1, p_3), (p_2, p_3)\}$

Suppose our points are $\{p_1, p_2, p_3, p_4\}$.

The search space is: $\{(p_1,p_2), (p_1,p_3), (p_1,p_4), (p_2,p_3), (p_2,p_4), (p_3,p_4)\}$ 6 pairs

In general, it's the binomial coefficient $\binom{n}{2}$ "n choose 2" which is the # of ways of picking 2 things out of n. (order doesn't matter)

$$\binom{n}{2} = \frac{n(n-1)}{2} = \frac{1}{2}n^2 - \frac{1}{2}n = O(n^2).$$

ignores multiples and vanishing terms

* It may seem surprising, but this can actually be done in $O(n \cdot \log(n))$ time, so without checking every pair! (next lecture)

GPU Problem from HW2:

What really is each configuration you're checking made up of? You have 60 transaction slots, and you need to assign a person to each of them. How many possibilities?

Slot 1: n people

Slot 2: n-1 people

Slot 3: n-2 people

⋮

Slot 60: n-59 people

(order matters!)

Search space: all ordered lists of 60 people (out of the n total)

Size: $n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-59) = n^{60} + ?n^{59} + ?n^{58} + \ldots$

$= O(n^{60})$.

Good news: Polynomial, not Exponential!

Bad news: Yikes...

NFL Schedules: search space for 1 week = all ways of putting 32 teams in pairs. For 17 weeks = all ways of picking 17 one-week schedules.

$\approx 6.5 \times 10^{294}$ (ignoring bye weeks)

# of atoms in the universe $\approx 10^{80}$

Summary of Brute Force

Pros - very easy to code
fewer bugs
guaranteed optimal
find all solutions
good to test other methods against

Cons - SLOW, can usually only do small cases

→ weighted interval / knapsack (2^n)
n up to 20-30 in a few minutes

→ pairs of points
n ≈100,000 in a minute (not bad!)

So, how can we find optimal solutions?

(1) Don't even bother - greedy algos

(2) Wander around the search space randomly, keeping track of the best you've seen. (random search)

(3) Wander around the search space cleverly, keeping track of the best you've seen. (meta heuristics)

So, how can we find optimal solutions?

(4) Check everything in the search space one-by-one (brute force)

(5) Check or otherwise rule out everything in the search space (divide-and-conquer, backtracking, branch-and-bound).

(6) Do some clever computations that allow you to score big chunks of the search space all at once (dynamic programming).

So, how can we find optimal solutions?

(4) Check everything in the search space one-by-one (brute force)

(5) Check or otherwise rule out everything in the search space (divide-and-conquer, backtracking, branch-and-bound).

(6) Do some clever computations that allow you to score big chunks of the search space all at once (dynamic programming).

Topic 7 - Divide and Conquer

"Divide and Conquer" is an algorithmic paradigm that is roughly

1) Split the input in half
2) Solve the problem on each half separately (recursively)
3) Combine your two answers into one big answer.

Classic Example: Sorting a list (easy)

* You can phrase this as a constraint satisfaction problem.
* Input: n numbers
* Search space: All orderings of n things. These are called permutations, and the # of them is $n(n-1)(n-2)(n-3)\cdots 3\cdot 2\cdot 1 = n!$
* Goal: Find the rearrangement that puts things in the right order.

* Obvious optimal algorithm: (greedy-ish)
  - Pick the smallest thing, put it first
  - Pick the next smallest thing, put it second, etc.

How many steps does this take?
- Finding the k^{th} smallest thing takes n steps (have to search the whole list)
- We have to do this n times.

Thus, $O(n^2)$. Fine for a few thousand things, but not more.

* Divide-and-conquer can do it in $O(n \log(n))$.

$n \to n^2$

$(2n) \to (2n)^2 = 4 \cdot n^2$

$n \to n^3$

$(2n) \to 8n^3$

$n \to 2^n$

$n+1 \to 2^{n+1} = 2 \cdot 2^n$