MSSC 6000
Feb 21, 2022 - Day 13

Lecture 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 our list of constraints?
and/or
(2) Which one is optimal? "

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

They don't check every [blank] - they only check a single one.

The search space of a problem is the set of all possible "things" that may or may not satisfy your constraints and that

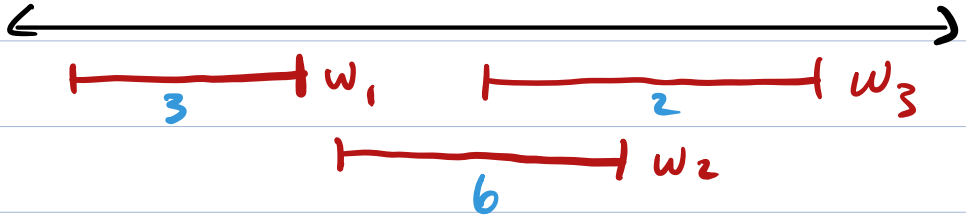
all have some score that you want to minimize or maximize.

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

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

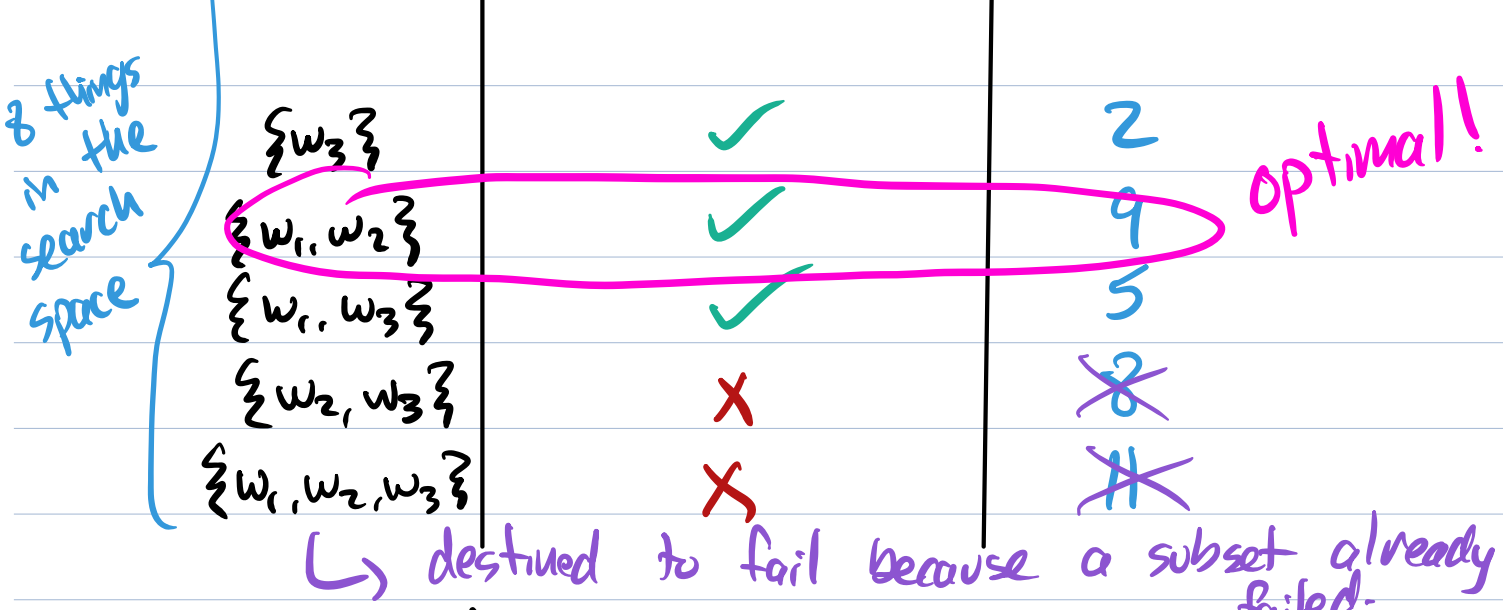
Ex 1: Weighted Interval Scheduling

3 requests:



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

Candidates	satisfies constr.	score
always suboptimal → $\{\}$	✓	0
$\{w_1\}$	✓	3
$\{w_2\}$	✓	6



↳ destined to fail because a subset already failed.

Optimal sol: highest score and satisfies the constraints

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

Brute force is very slow, but usually very easy to code.

In python: a library called "itertools"
Google: "all subsets with itertools"
"powerset"

Pseudocode

```
R = set of requests
b = 0          best_subset = None
for each subset r of R:        loops 2^n times
```

```
if r satisfies constraints:        takes O(n) time
    s = score(r)                   takes O(n) time
    if s > b:                      } takes constant
        b = s                          O(1)
        best_subset = r
return b, r
```

Total time $\approx 2^n \cdot (2n+1) = 2 \cdot n \cdot 2^n + 2^n$

$= O(n \cdot 2^n)$

Knapsack Problem: Same situation:

n items

search space = all subsets of n items

size is 2^n again

Closest Pair Problem:

Input: n points in xy-plane

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

Search Space = all unordered pairs of points

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, the # of pairs is the binomial coefficient $\binom{n}{2}$ (or ${}_nC_2$), which is $\frac{n \cdot (n-1)}{2}$. ← size of our search space

$= \frac{n^2}{2} - \frac{n}{2} = O(n^2)$ not exponential!

* This is surprising: this can be solved in $O(n \cdot \log(n))$ time, so without checking every pair.

Gamestop Problem from HW 2:

What is each configuration you're checking?

You have 60 timeslots. (ignoring for simplicity)

A solution assigns one ~~or zero~~ people to each time slot.

60 time slots

Slot 1: 100 people
Slot 2: 99 people
⋮
Slot 60: 41 people

} $100 \cdot 99 \cdot 98 \cdots \cdot 41$

n people: $n(n-1)(n-2)\cdots(n-59)$

$= n^{60} + ?n^{59} + ?n^{58} + \cdots$

$= O(n^{60})$

Good news: not exponential

Bad news: very big

Summary:

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

Cons - Slow, can usually only do small cases

→ weighted interval / knapsack
n up to ≈25 in a few minutes

→ pairs of points, ≈ 100,000 in a

minute.