Friday March 4 - Day 18

Backtracking — Knapsack Demo

Topic 8- Branch and Bound

Recall that our problems have two considerations:

(1) Constraints that must be satisfied

(2) A value/score that has to be minimized or maximized.

Backtracking boiled down to:
If you build your solutions a bit at a time, you can detect early if the constraints are violated, and rule out a big chunk of the search space all at once.

This never considered value.

Branch and Bound is just backtracking with an extra way to rule out a partial solution.

Assume for now we are maximizing

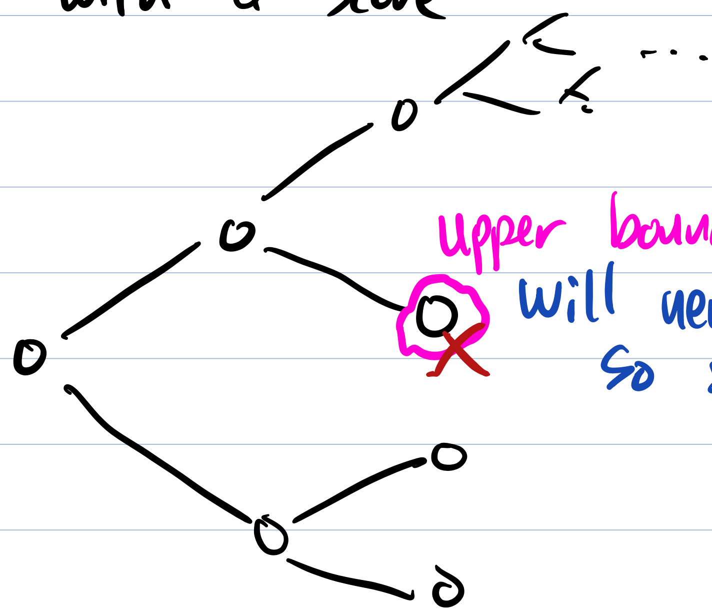
* If I have already seen a complete solution with a score of X, 50
  and I'm building a partial solution:
  if there is no way to complete this partial solution with a score $\geq X$,
  then give up on it (prune the branch and backtrack).

There's no way to know exactly the best you can do when completing a partial solution.

Want: A way to get an upper bound on the best you can do when completing a partial solution.

"I don't know how good I can do when completing this partial solution, but I know for sure that I can't do better than Y."

Have a complete solution with a score of 30.



upper bound of 25

will never beat 30, so prune.

Hard part: how to compute an upper bound
→ we'll come back to that

Mathematical Framework for Backtracking and B+B:

(1) "making decisions to build partial solutions"
⇒ splitting the search space into <u>disjoint</u> parts (subspaces)
↳ no overlap

<u>Ex:</u> Knapsack - Item 1 is in or out

{all subsets of items} ⟶
{subsets containing 1} and {subsets not cont. 1}

{subsets containing 1}

{subsets cont. 1 and not cont. 2}

{subsets cont. 1 and 2}

This is called <u>branching</u>.

(2) For any subspace S that we create with branching, we need to be able to bound(S), some upper bound on the best score possible for any candidate in S.

<u>Notes:</u>

* We're phrasing for maximization.
* bound(S) has to be an <u>upper</u> bound. Lower bounds are easy (e.g. greedy) but useless.

Ex: Job Assignment Problem.

You have n tasks that need to be done and n workers. Each task has a different cost to complete depending on which worker does it.
Goal: Minimize total cost.

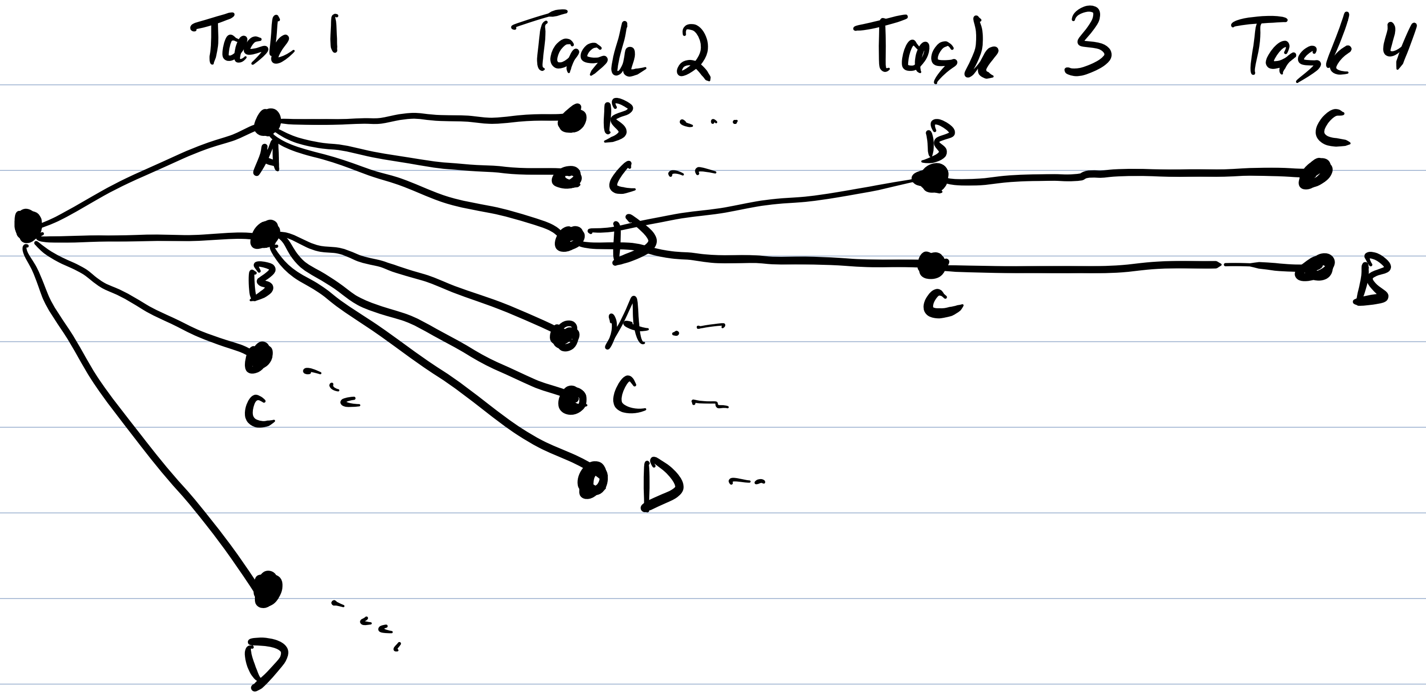
tasks

workers	1	2	3	4
A	3	5	2	2
B	6	8	10	8
C	2	6	4	9
D	10	4	7	5

Assign 1 task to each worker.

* Search space: All assignments of workers to tasks. How big?
$$n! = n \cdot (n-1) \cdot (n-2) \cdots 3 \cdot 2 \cdot 1$$
* No constraints, so backtracking alone is = brute force.

A B C D


Task 1
Task 2
Task 3
Task 4
A
B
C
D
B ...
C ...
D
A ...
C ...
D ...
B
C
C
B