Scientific Computing

Fri, Feb 20

Announcements

* HW 3 is due Friday, Feb 27
  covers brute force, search spaces,
  divide + conquer
  remember to let yourself struggle!

* Midterm exam
  Monday, March 2
  in class portion + takehome portion
  due Friday, March 6

Office Hours:

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

Cudahy 307

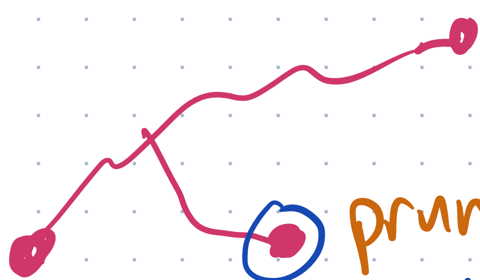
Backtracking boiled down to:

If you build your solutions a bit at a time, you can detect early if the constraints are violated, and rule out a chunk of the search space at once.

This never considered <u>value</u>.

Branch and Bound is just Backtracking with an <u>extra</u> way to rule out a partial solution: (assuming maximization for now)

* If I've already seen a complete solution with a score of X, and there is <u>no</u> way to complete this partial solution in a way that beats that, prune it (stop looking at this node).



There's no way to know exactly the best you can do on completing a partial solution — if you could do that, you could just do it from the start and solve the problem right away.

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

"I don't know how good I can do, but I know for sure I can't do better than Y."

Mathematical Framework for Backtracking and B+B:

(1) "making decisions to build partial solutions"
$\Rightarrow$ really splitting the search space into
disjoint parts (subspaces)
↪ no overlap

Ex: Knapsack - Item 1 is in or out
{all subsets of items} → {subsets containing 1} and
{subsets not containing 1}

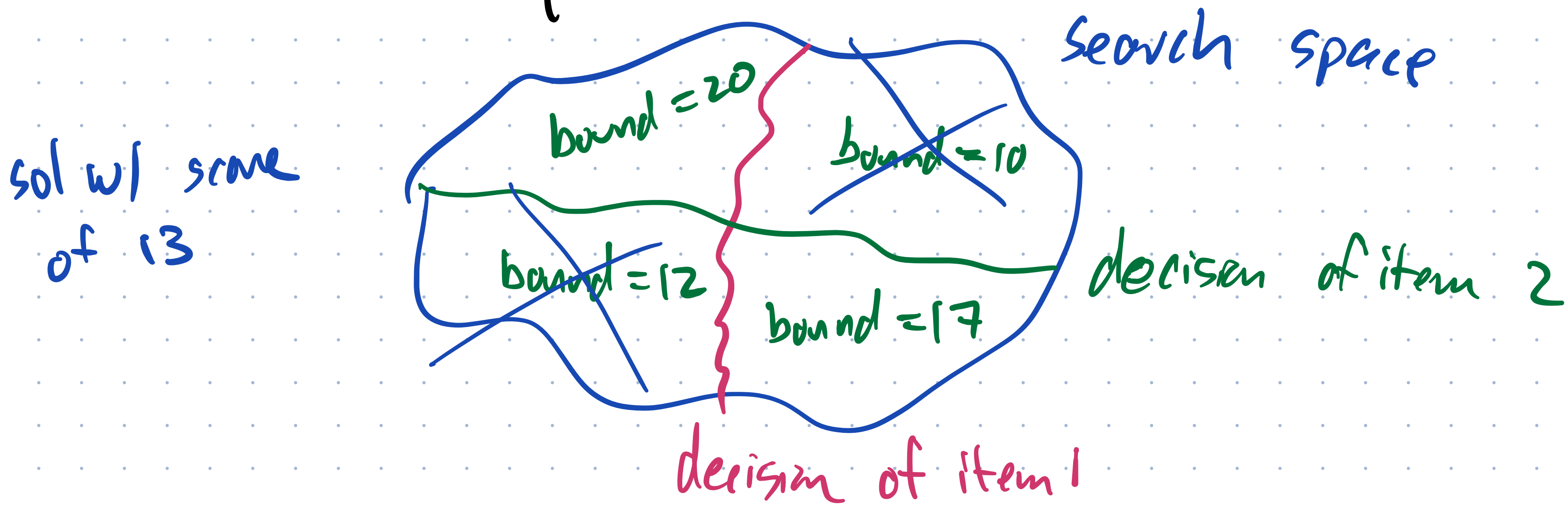
Another example:

{subsets containing 1}

→ {subsets containing 1 and 2} and

{subsets containing 1 and not containing 2}

This is called **branching**. It doesn't have to always be into two parts.



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

Notes:

* We're phrasing this all for maximization.
* bound(S) has to be an upper bound. Lower bounds are easy (greedy) but useless.

Ex: Problem #5: Job Assignment

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.

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

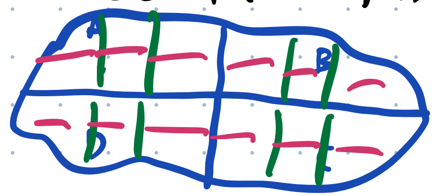
Score: $5+6+9+7=\boxed{27}$

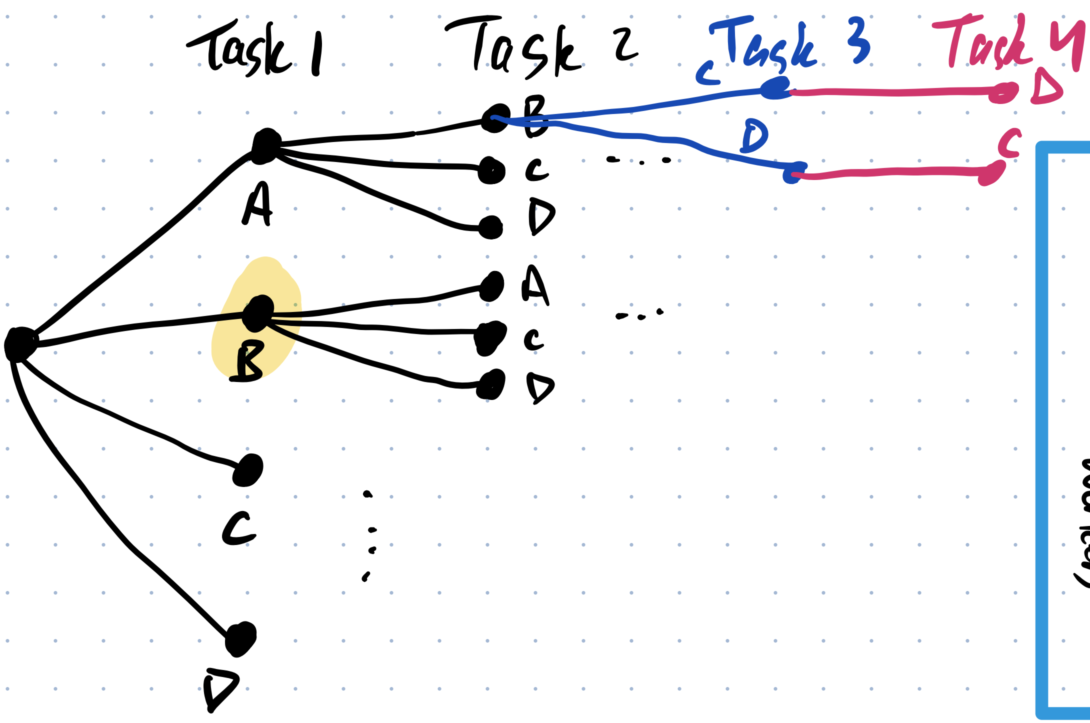
Many applications:

- Drivers picking up passengers
- Shipments from mines to factories

* Search Space: All assignments of workers to tasks.
  How big? $n! \approx n^n$
* No constraints, so backtracking alone is useless.
  (equiv. to brute force)

Branching - Pick which worker does a certain task





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

Bounding:

* Suppose you've already picked worker B to do Task 1.
* What is a lower bound on the best you can do to finish? (Has to be easier than actually solving the whole problem.)

tasks

workers	1	2	3	4
A	~~8~~	5	~~2~~	2
B	6	~~8~~	~~10~~	~~8~~
C	~~2~~	6	4	9
D	~~16~~	4	7	5

minimizing = lower bound to do B+B

We don't know how cheaply we can finish the other jobs, but it can't be cheaper than X.

LB: \$1 per remaining job (\$3)

LB: \$2 per job (\$6)

LB: \$8 => LB: \$10

One possibility: every task will cost at least its minimum remaning

Another possibility: every worker will incur a cost at least the minimum of the tasks remaning.

	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

So, finishing will cost at least ___.

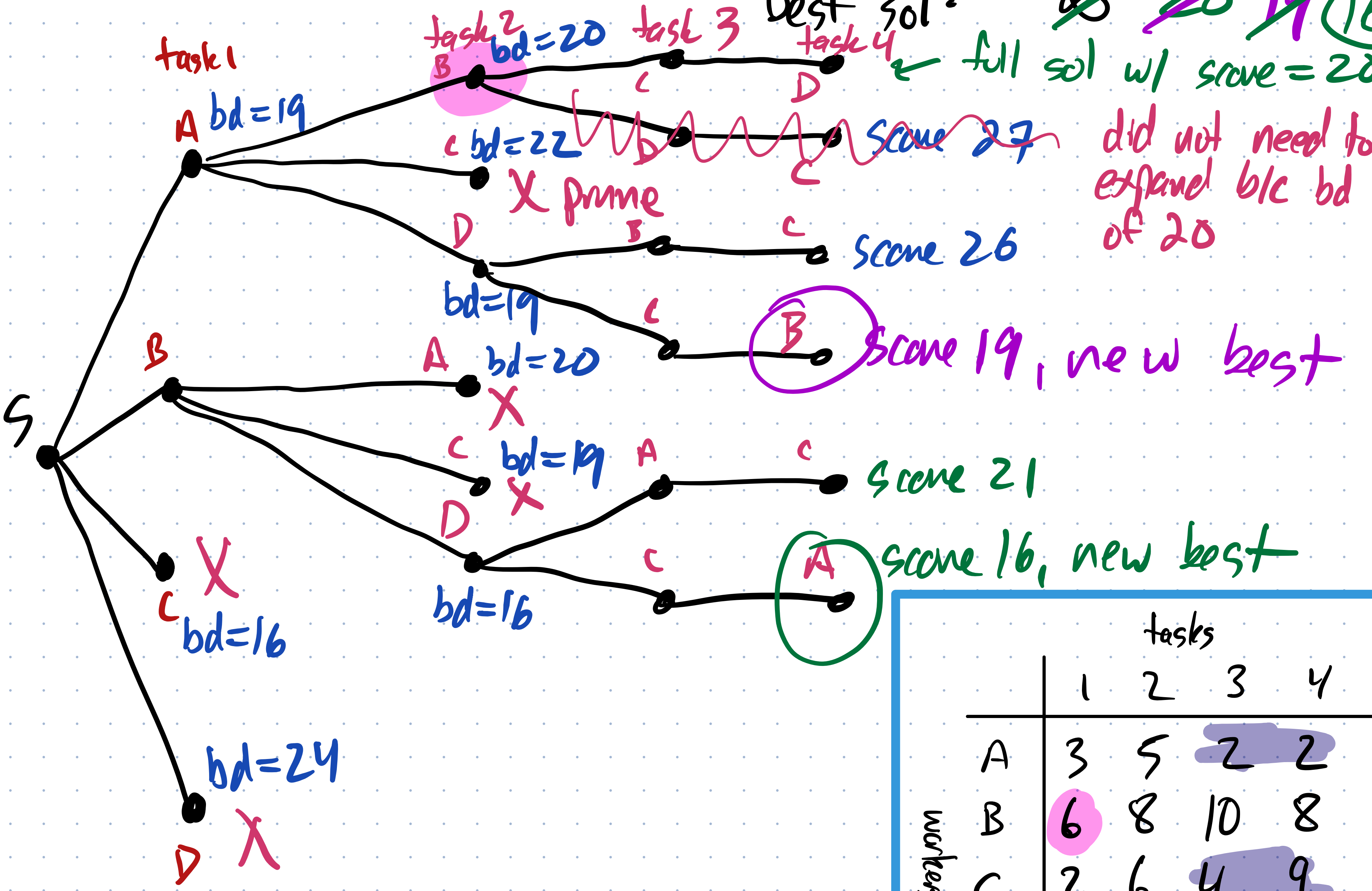
Which bound is better?

So, our lower bound will be

max(sum of smallest cost in each remaining row,
sum of smallest cost in each remaining col)
+ existing cost of selections.

Let's work through it.

best sol: ~~∞~~ ~~20~~ ~~19~~ 16



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

Notes:

* Again, the hardest part is finding a good bound! The stronger, the better.

* At the start, we didn't have a best_sol to do any pruning until we branched down to a single candidate. If we found a candidate before we started B+B, maybe there would have been more pruning.
    - Pick a few random solutions.
    - Pick a greedy solution.

Can get a greedy solution with score 16!

That would be proved optimal with very little branching.

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