# Scientific Computing

Wed, Feb 18

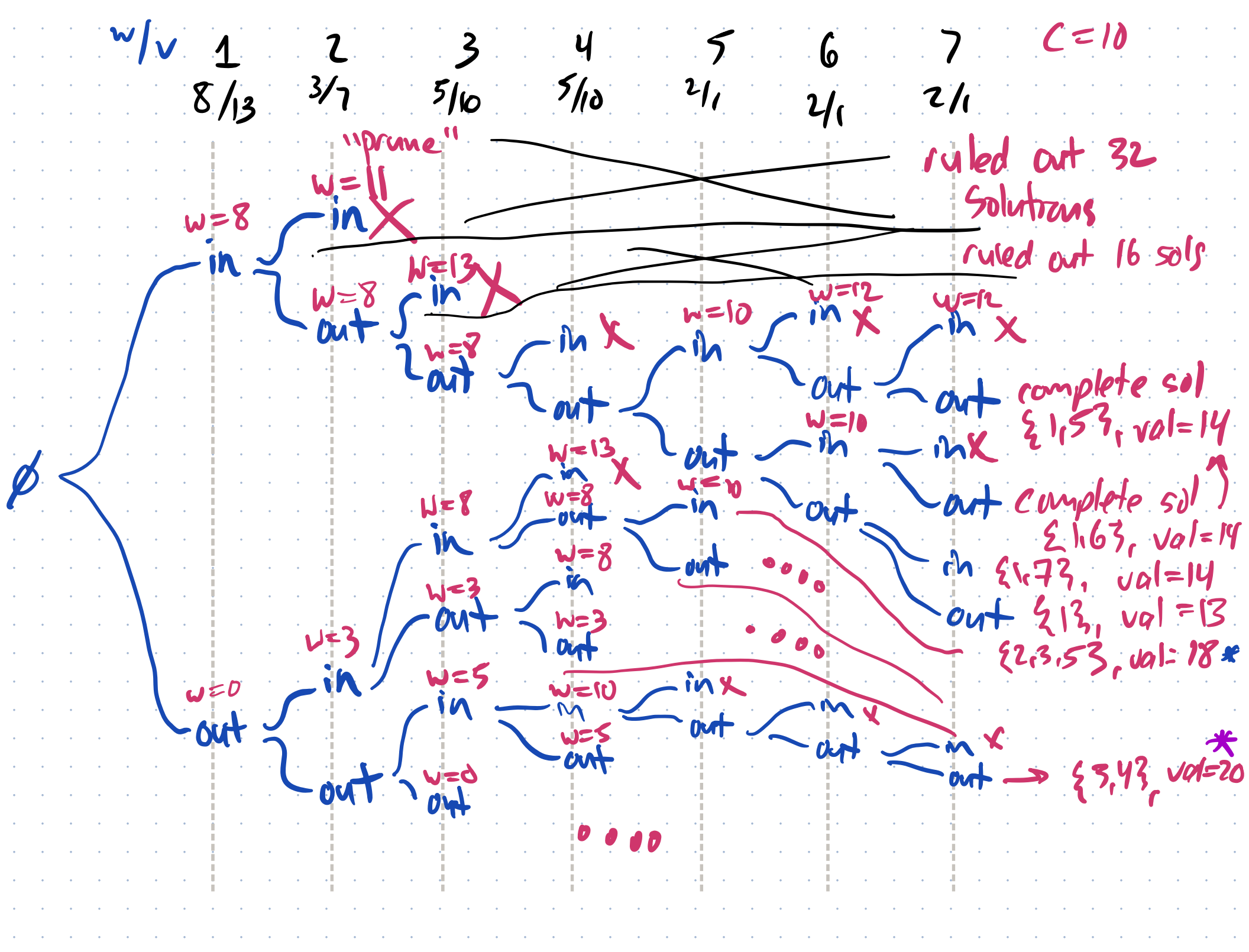
## Announcements

* 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


w/v
1
8/13
2
3/7
3
5/10
4
5/10
5
2/1
6
2/1
7
2/1
C=10
"prune"
ruled out 32 solutions
ruled out 16 sols
∅
w=8
in
out
w=11
in X
w=13
in X
w=8
out
w=7
out
in X
out
w=10
in
out
w=12
in X
out
w=12
in X
out
complete sol
{1,5}, val=14
w=13
in X
w=8
out
w=10
in
out
in X
out
complete sol
{1,6}, val=14
w=8
in
w=8
in
out
w=3
out
w=3
in
w=3
out
w=8
in
out
in {1,7}, val=14
out {1}, val=13
{2,3,5}, val=18*
w=0
out
w=3
in
out
w=5
in
w=0
out
w=10
in X
w=5
out
in X
out
in X
out
in X
out
{3,4}, val=20*

## Ex #2: Sudoku

- Start filling in blank cells L-to-R then T-to-bottom.
- Start each cell at 1.
- If the cell doesn't violate a rule, move to the next cell.
- If not, bump up the value.
- If you run out of possibilities, go back to the previous cell.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 4 | 7 | 1 | 6 | 2 | 3 | 8 | 9 | 5 |
| 6 | | 8 | | 5 | 4 | | | |
| | | 5 | | | 8 | 7 | | 4 |
| 8 | | | 4 | 3 | 2 | | | |
| | 3 | | | 1 | | | 4 | |
| | | | 9 | 8 | 7 | | | 1 |
| 1 | | 3 | 8 | | | 4 | | |
| | | | 3 | 4 | | 5 | | 9 |
| | | | | 6 | 9 | | 1 | 8 |

* online demo – jaypantone.com/sudoku
/ sudoku-slow

"Hardest Sudoku Ever"

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | | | | | 7 | | 9 | |
| | 3 | | | 2 | | | | 8 |
| | | 9 | 6 | | | 5 | | |
| | | 5 | 3 | | | 9 | | |
| | 1 | | | 8 | | | | 2 |
| 6 | | | | | 4 | | | |
| 3 | | | | | | | 1 | |
| | 4 | | | | | | | 7 |
| | | 7 | | | | 3 | | |

## Sudoku

Let's say there are 30 empty boxes

A solution is built from 30 decisions

Box 1: $1, 2, 3, \ldots, 9$

Box 2: $1, 2, 3, \ldots, 9$

$\vdots$

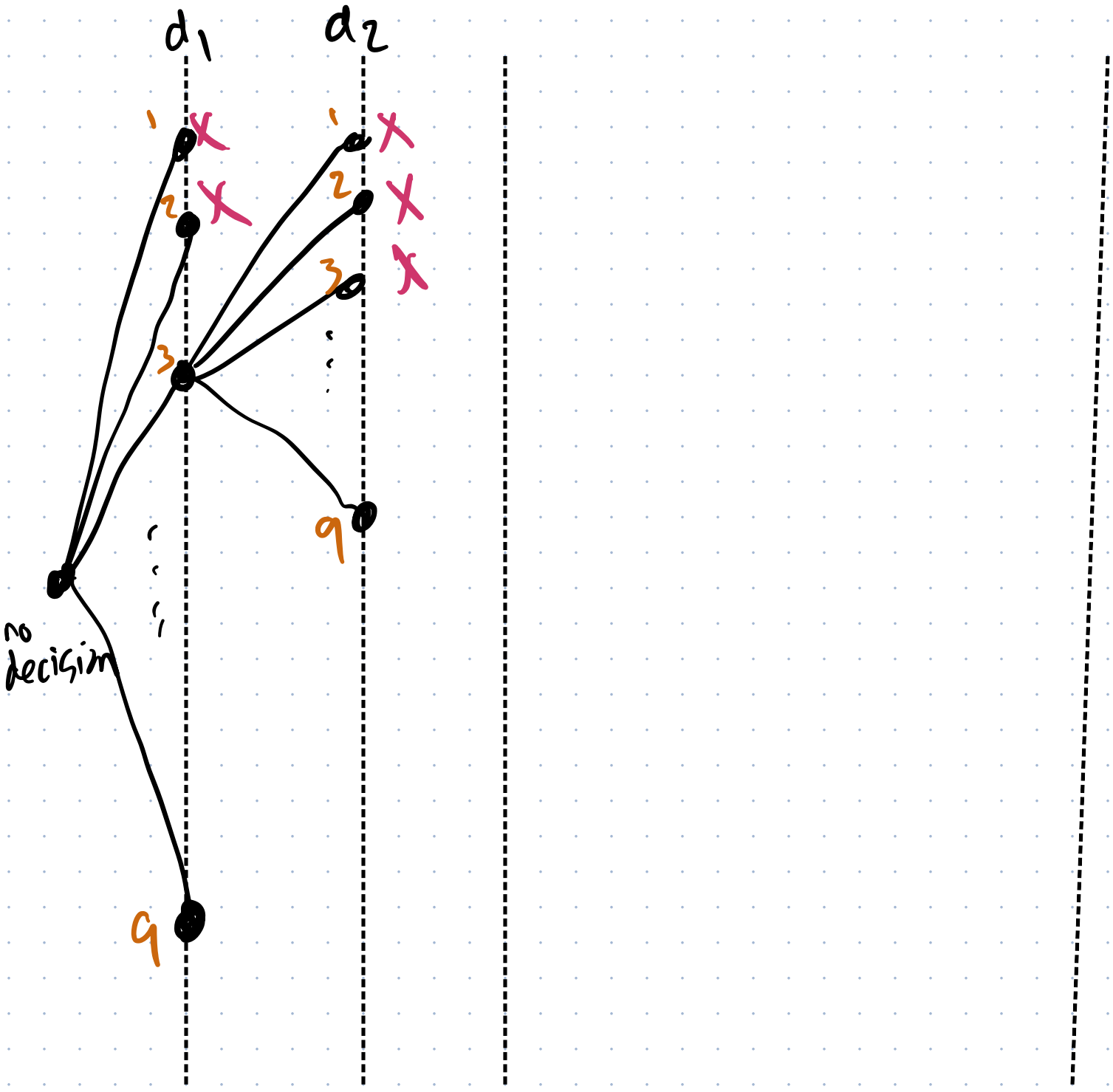
Box 30: $1, 2, 3, \ldots, 9$

Search Space: All things of the form

$$(d_1, d_2, \ldots d_{30})$$

where $d_i \in \{1, 2, \ldots, 9\}$

Size is $9^{30}$


$d_1$
$d_2$
1
2
3
1
2
3
X
X
X
X
X
q
q
no decision

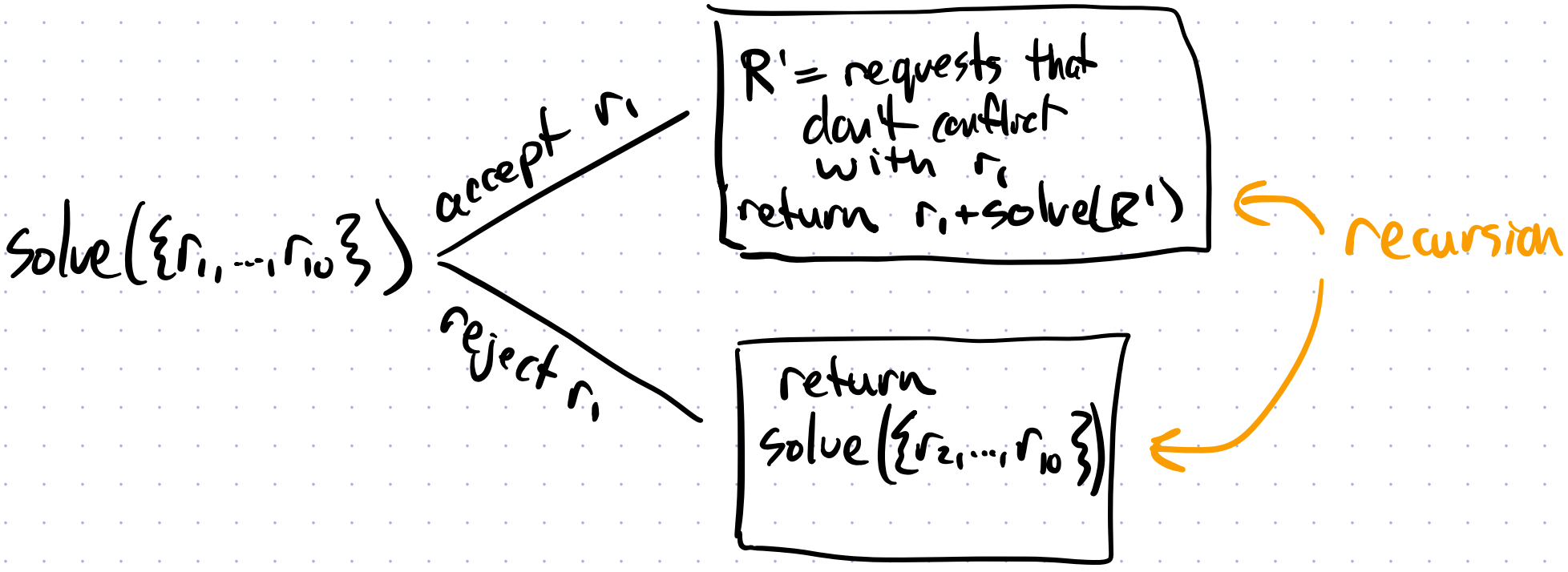
## Ex 3: Weighted Interval Scheduling

Requests $R = \{r_1, r_2, r_3, \ldots\}$

You either accept or reject each request.
If you accept $r_i$, then in the future you can ignore requests that conflict with $r_i$.

This is exactly the kind of situation that recursion is perfect for because we're repeating the same logic repeatedly on subproblems.

$R = \{r_1, \ldots, r_{10}\}$



# Pseudocode

```
function solve(requests):
    # goal: return best solution that
            can be made from [requests]
    if len(requests) = 0:
        return [ ]
    new_request = requests[0]
    compatible = requests compatible with new_requests
    accept_solution = [new_request] + solve(compatible)
    reject_solution = solve(requests[1:])
    return whichever of accept_solution and reject_solution
        has the highest value
```

recursion

[demo!]

# Topic 9 - Branch and Bound

Recall that our problems usually have two considerations:

(1) Constraints that must be satisfied

ex: capacity of the knapsack

choosing interval requests that don't conflict

row/col/square sudoku conditions

(2) A value/score that we want to maximize or minimize among all candidates that satisfy the constraints.

Some problems are only about constraints (Sudoku, NFL scheduling).

Some problems don't really have constraints and are only about optimizing – it can depend on how you define your search space (traveling salesman)

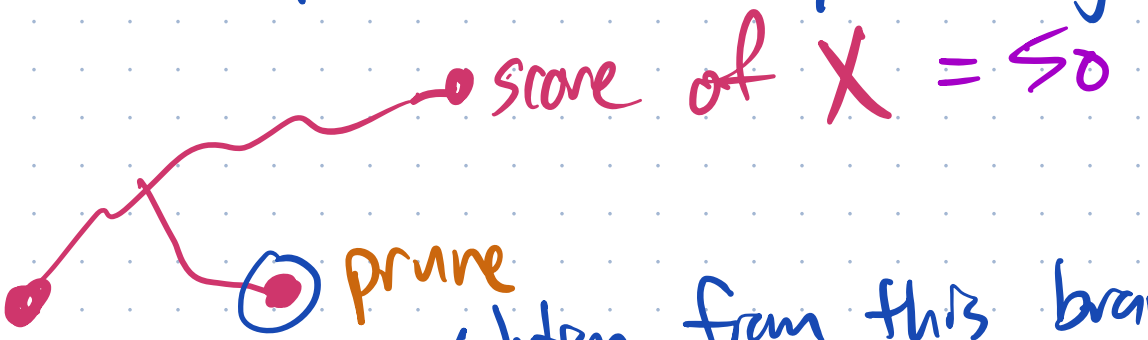
Backtracking boiled down to:

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 value.

Branch and Bound is just Backtracking with an extra 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 no way to complete this partial solution in a way that beats that, prune it (stop looking at this node).

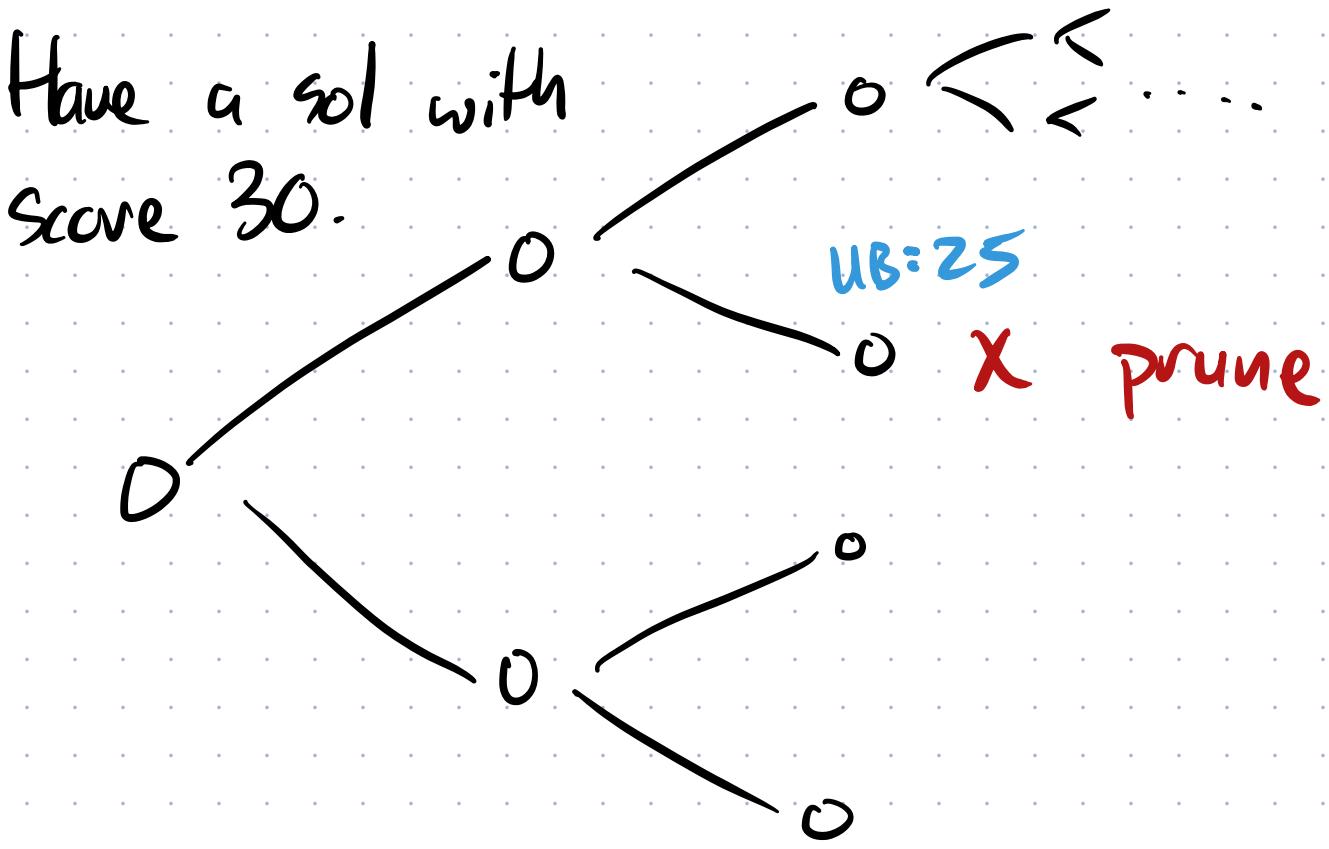



optimal sol: ≥0

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."




We're not addressing the hard part yet: how to compute an upper bound. We'll come back to that.