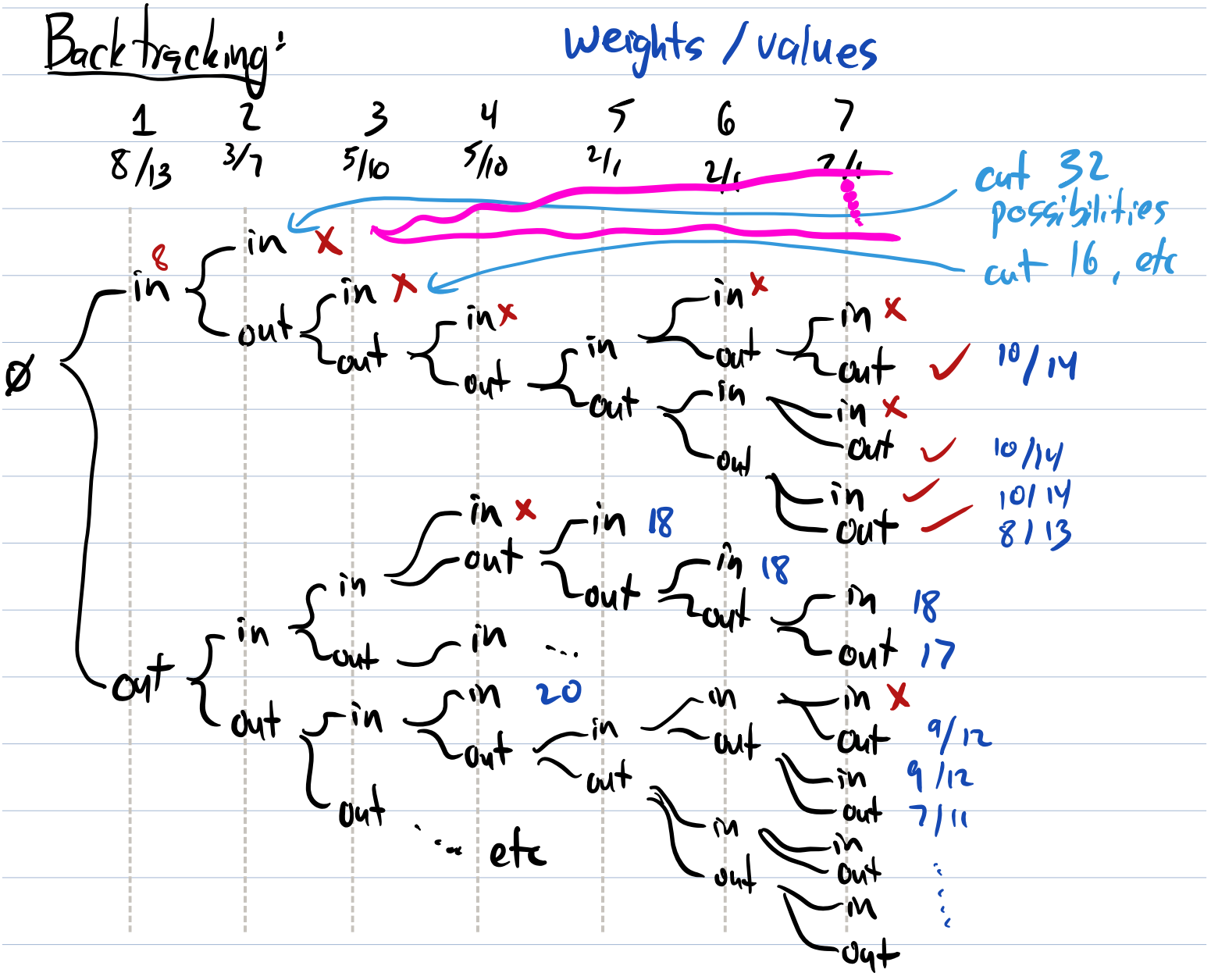
MSSC 6000
March 2 2022 - Day 17

Topic #7 - Backtracking (continued)



What are we doing?

- Putting a hierarchy on decisions that builds the whole search space,

with the critical property that if a partially built solution is bad, then every way of completing it must also be bad

Ex #2: Sudoku

1-9

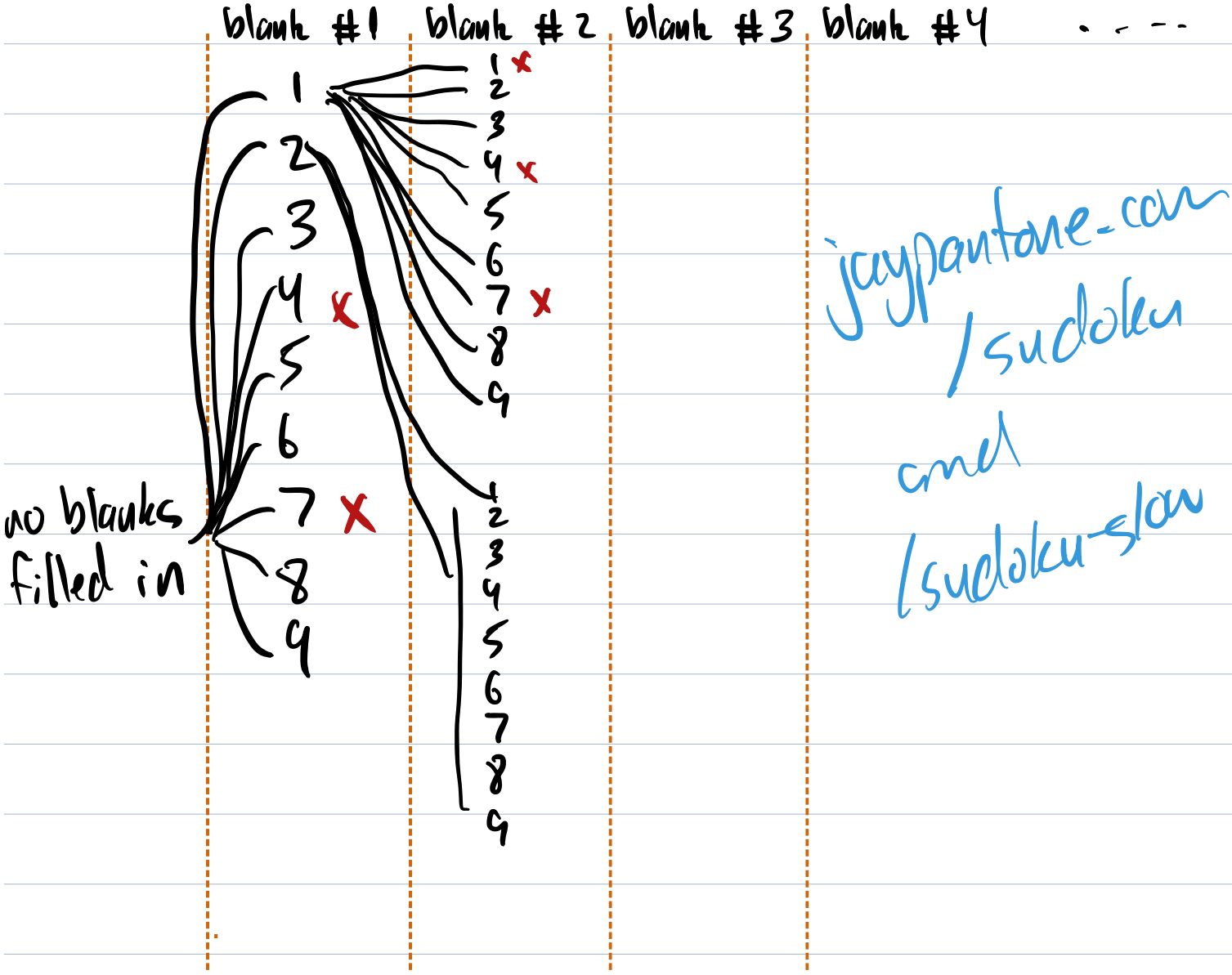
7,9 SAT-solver

- Start filling in blank cells left-to-right, then top-to-bottom.
- Start each blank cell with 1.
- If that cell doesn't violate a rule, move to the next cell and repeat.
- If it does violate, increase it by 1
- If you run out of possibilities, backtrack, go back to the previous cell, increment it by 1, and repeat.

4	7		6	2				
6		8		5	4			
		5		(9)	8	7		4
8			4	3	2			
	3			1			4	
			9	8	7			1
1		3	8	(7)		4		
			3	4		5		9
				6	9		1	8

9^{28}

- What would a brute force algo be? 30 blanks $\Rightarrow$ 9^{30} solutions to check



jaypantone-cpu/sudoku and /sudoku-slow

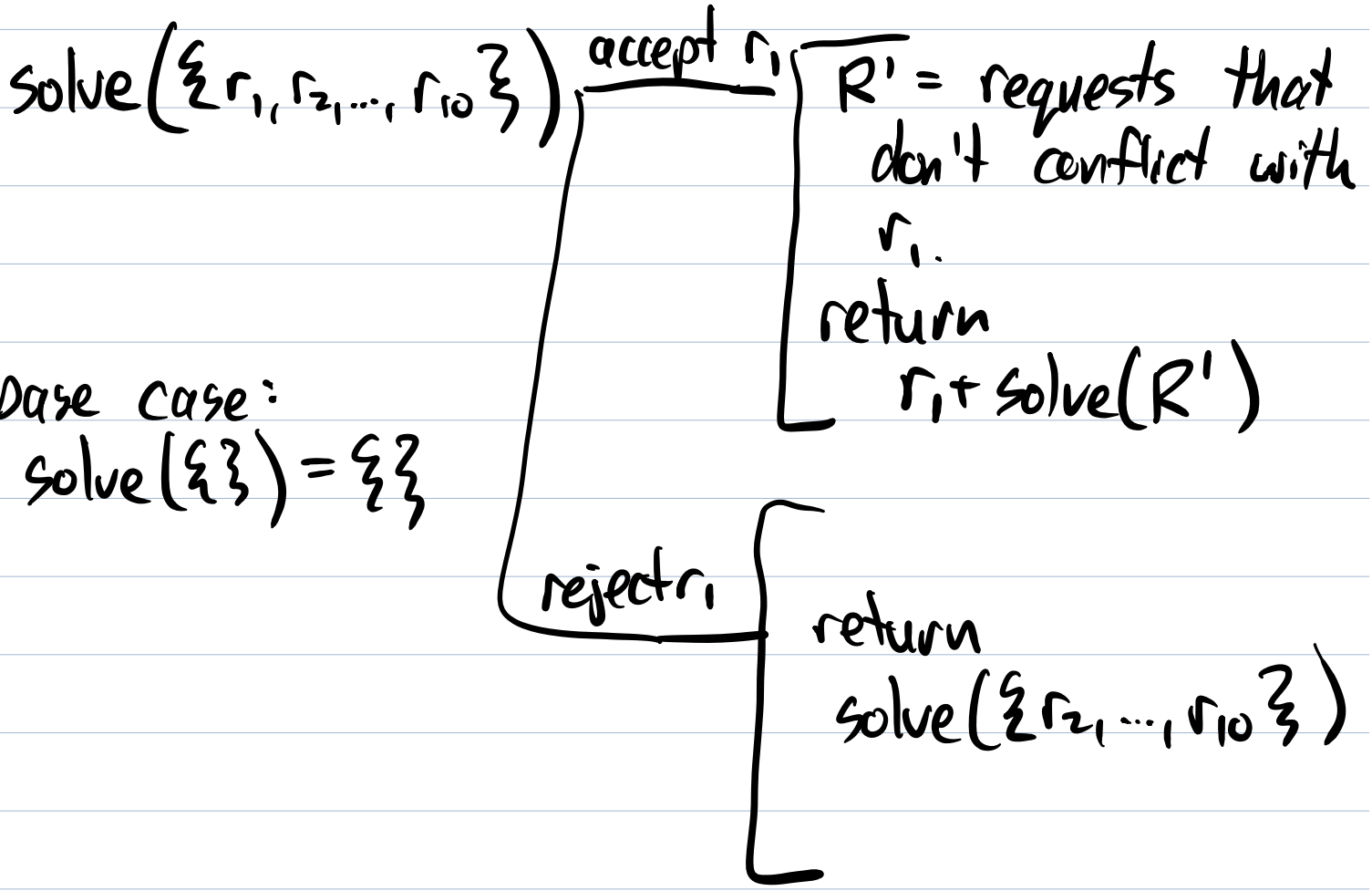
Ex 3: Weighted Interval Scheduling.

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

- You either accept or reject each request.

If you accept r_i, then in the future you can ignore any requests that conflict with r_i.

This is what recursion is great at because we're repeating the same logic on subproblems.



base case:
solve({}) = {}

Pseudo code:

```
function solve(requests):
    #goal: return the best subset of
           requests
    if len(requests) = 0:
        return []
    new_request = requests[0]
    compatible = the requests compatible with
```

new_request

accept_solution = [new_request] + solve(compatible)

reject_solution = solve(requests[1:])

return whichever of accept_solution and reject_solution has the higher score

Jupyter notebook demo