Introduction to

Artificial Intelligence

Lecture 5 – CSP

CS/CNS/EE 154
Andreas Krause

To Print fast, saved in EMF

Announcements

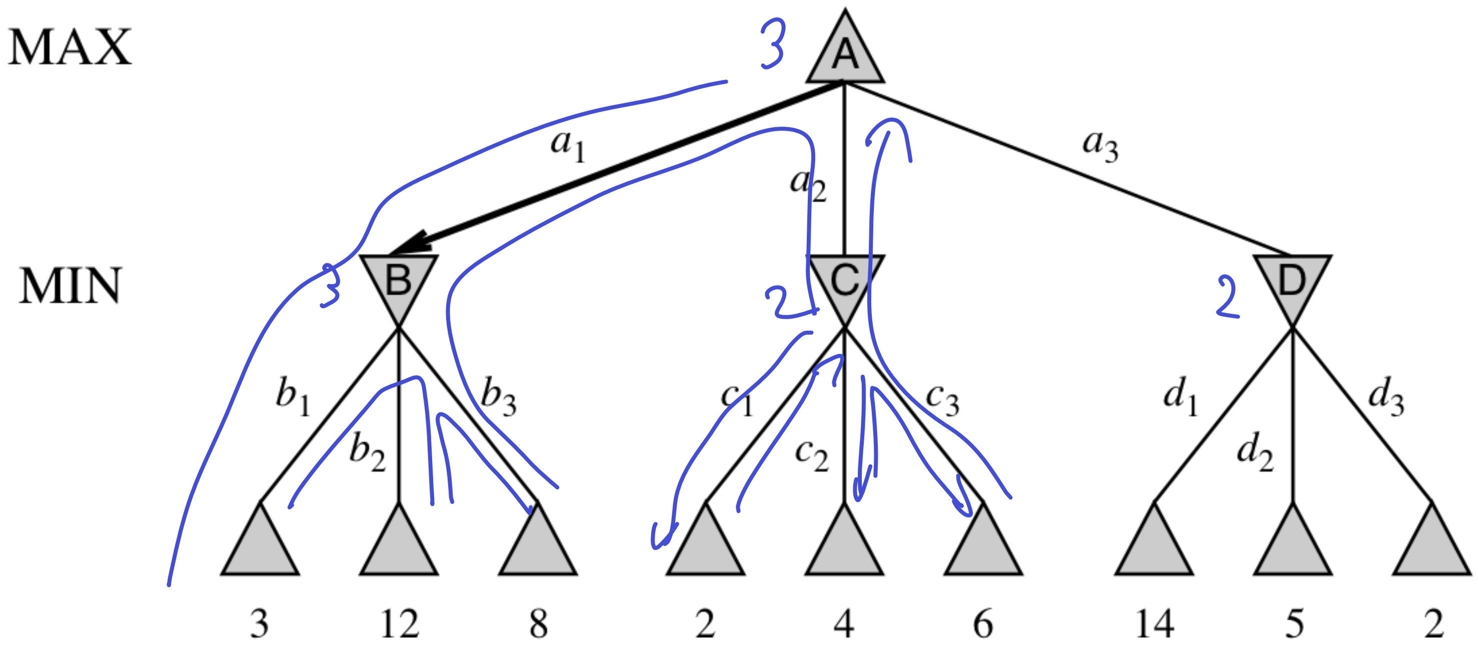
- Sign up for projects
  - Will make assignments tomorrow
- Homework 1 is out. Due Friday Oct 22

Games vs. search

- In games, actions are nondeterministic
  - Opponent can affect state of the environment
- Optimal solution no longer sequence of actions, instead a strategy (policy, conditional plan)
  - If you X I'll do Y, else if you do Y I'll do Z, ….

Minimax game tree

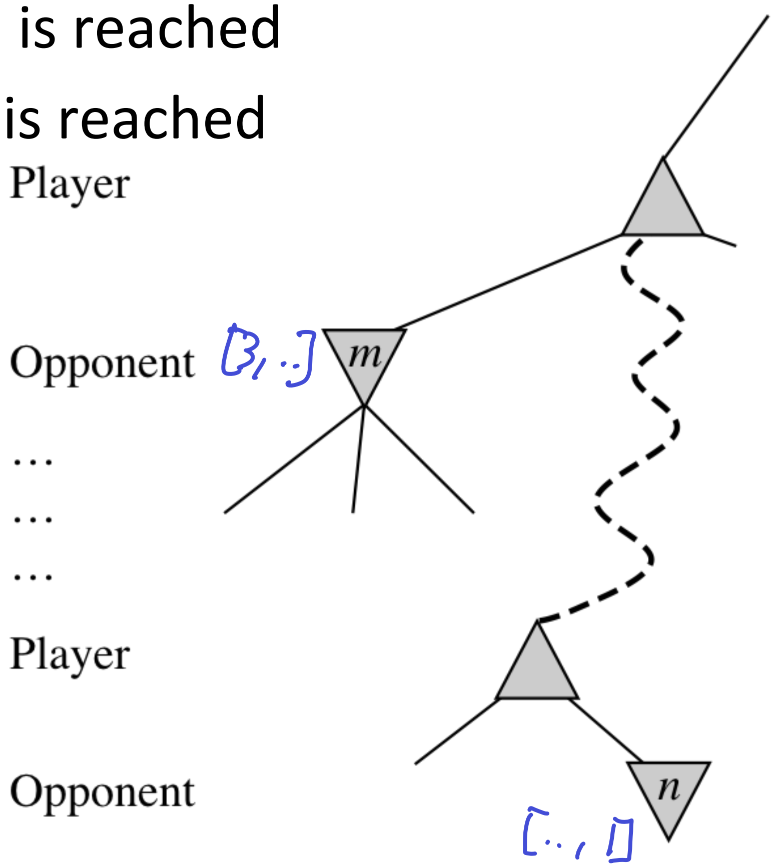
- Search for optimal move no matter what opponent does
- minimax value = best achievable payoff against best play



Can evaluate using DFS

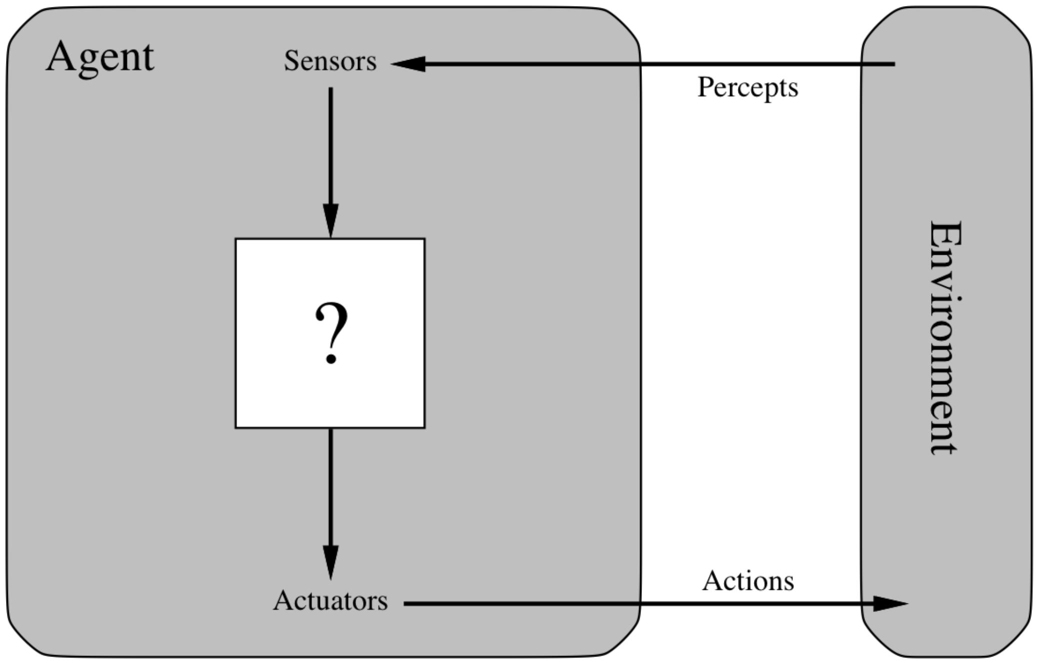
α-β-pruning

- Key idea: For each node n in minimax tree keep track of
  - α: Best value for MAX player if n is reached
  - β: Best value for MIN player if n is reached
- Never need to explore consequences of actions for which β<α
- Avoid exploring “provably suboptimal” parts of minimax tree



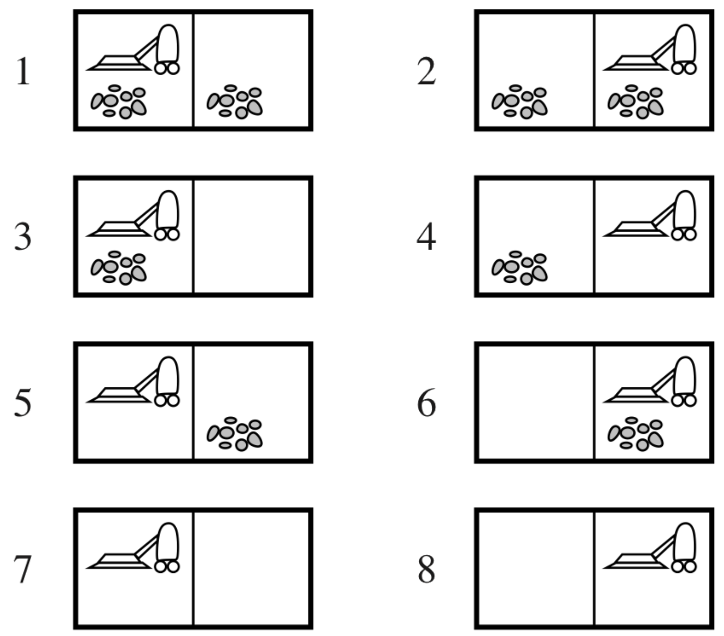
Nondeterministic and partially observable search

- Nondeterminism
  - Environment state not a function of current state and action
- Partial observability
  - Percept is not a function of environment state

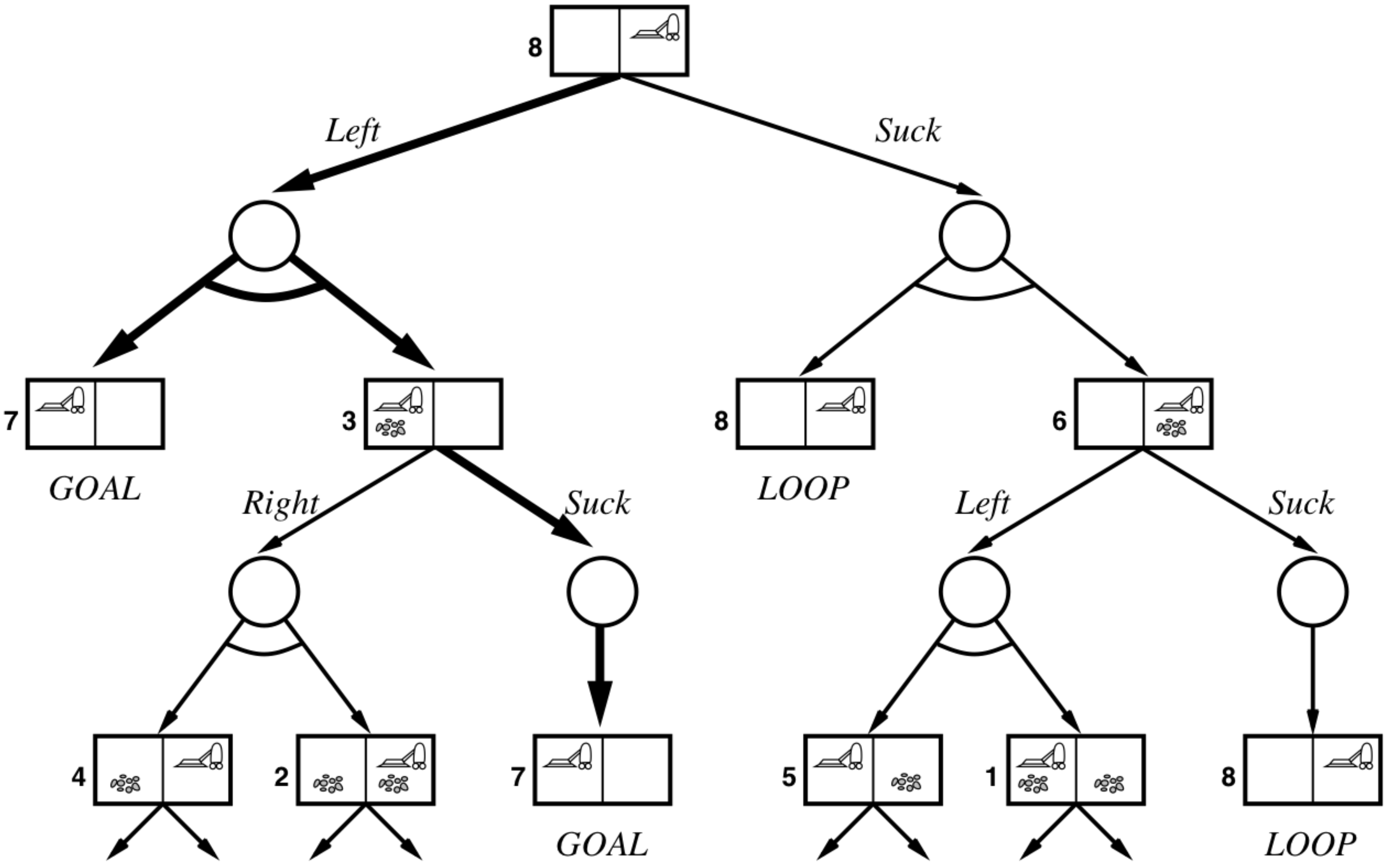


Nondeterminism

- So far, assumed that
  - all actions are deterministic
  - State is fully observable
- What if the actions are noisy?
- Example:
  - If applied on dirty square: “Suck” sometimes cleans up neighboring square as well
  - If applied on clean square: “Suck” sometimes dirties the square
- Solution?



AND-OR trees



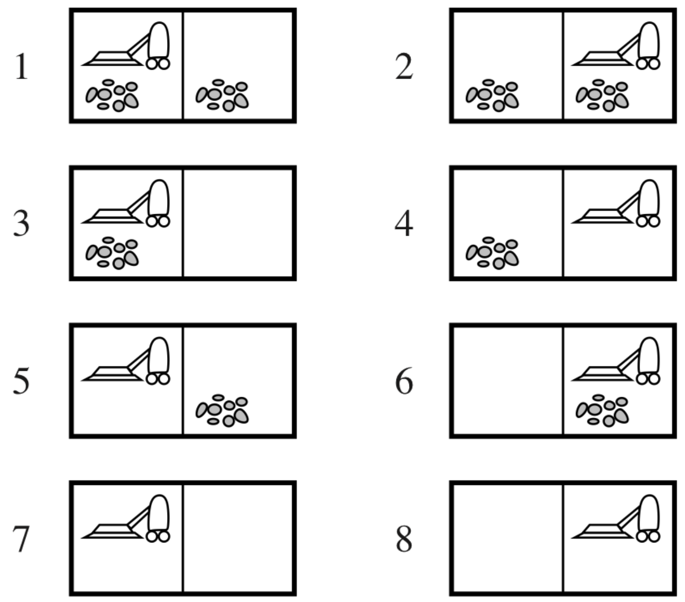
8
Left
Suck
7
3
8
6
GOAL
Right
Suck
LOOP
Left
Suck
4
2
7
5
1
8
GOAL
LOOP

Conditional plans

- For nondeterministic actions, optimal solution no longer a sequence, but a conditional plan (strategy)
- Can represent in AND-OR tree (like minimax)
- OR nodes: Agent chooses action
- AND nodes: Environment chooses next state
  - Need to specify what to do for every possible next state!
- Evaluate using backward induction (like minimax)
- Use IDS to grow tree until found solution (or tired)

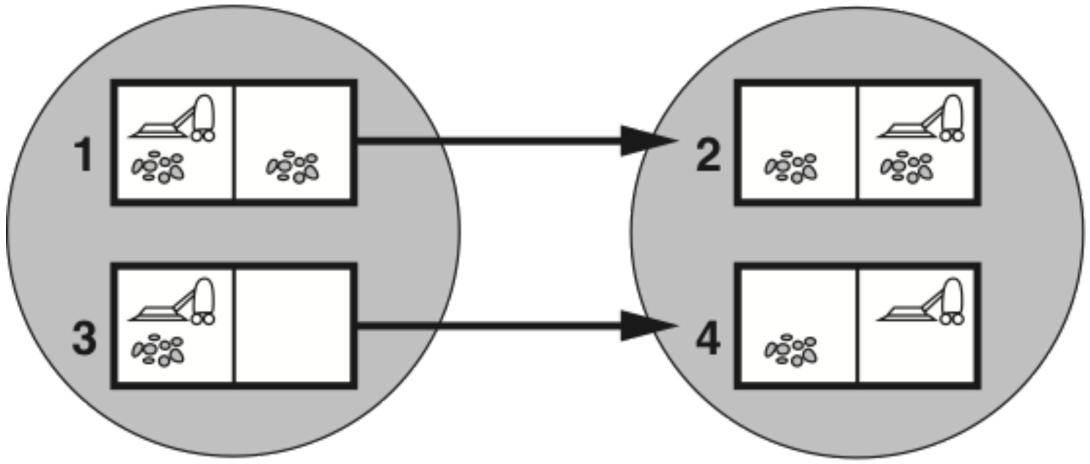
Partial observability

- Suppose our robot is sensorless.
  Can we still plan?

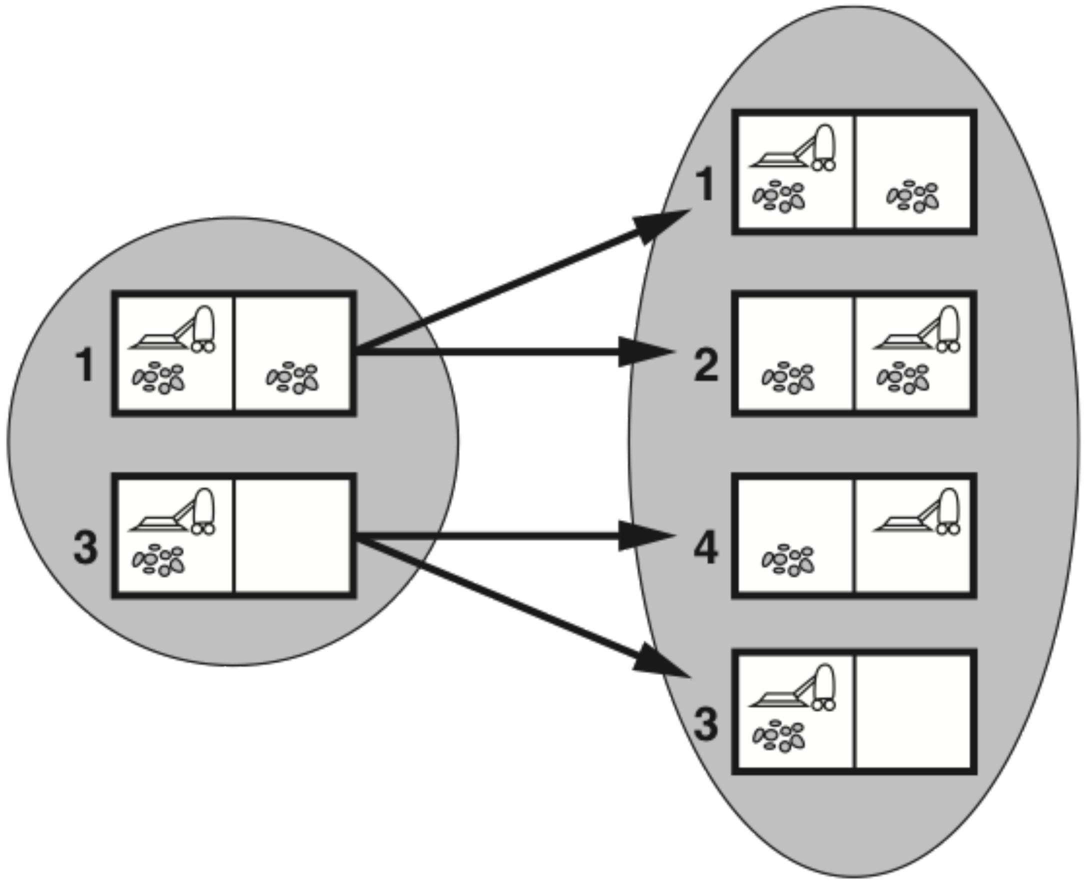


- Plan on “belief states” (sets of environment states), instead of individual states

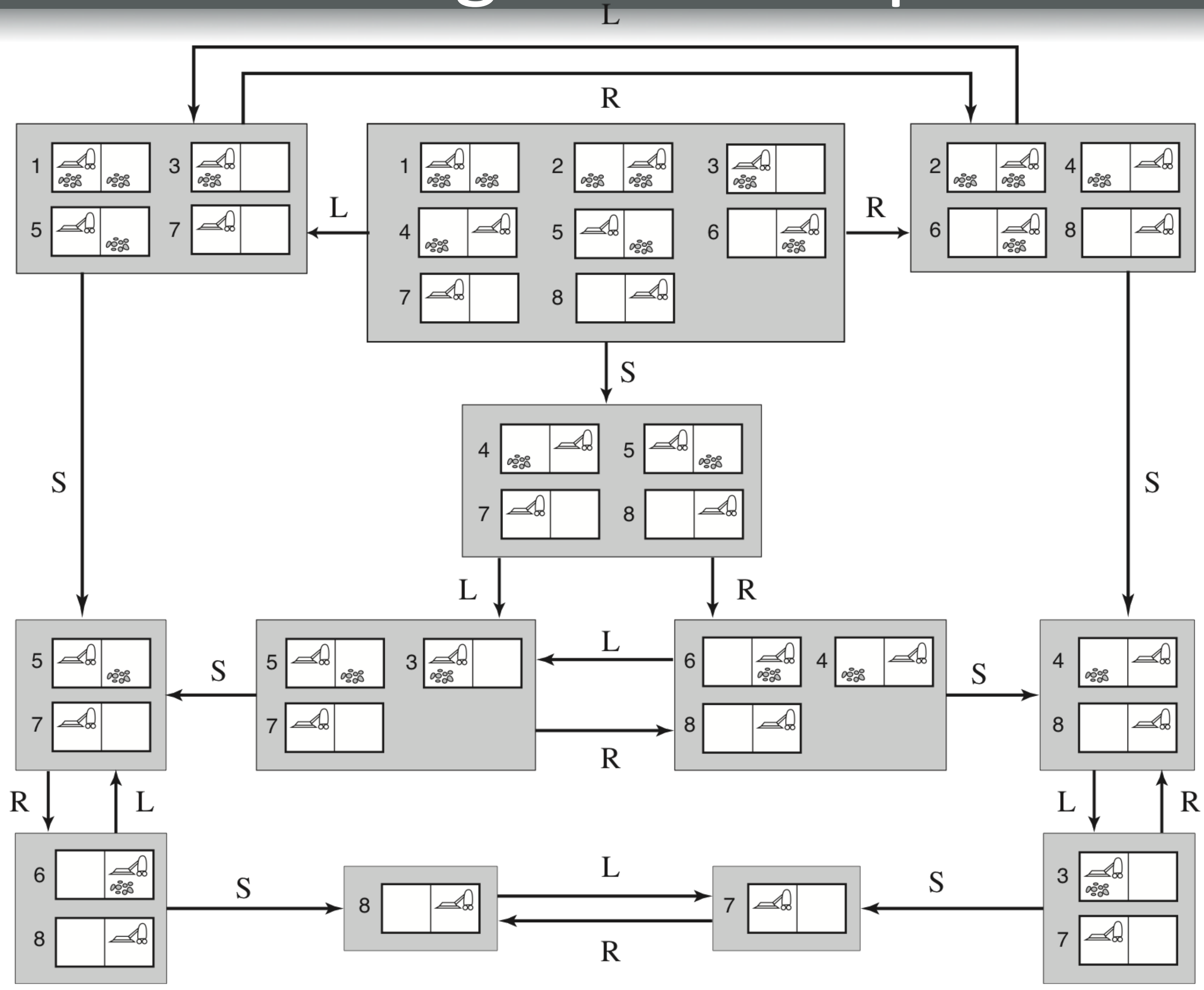
Working with belief states



Noisy actions (slippery vacuum world)



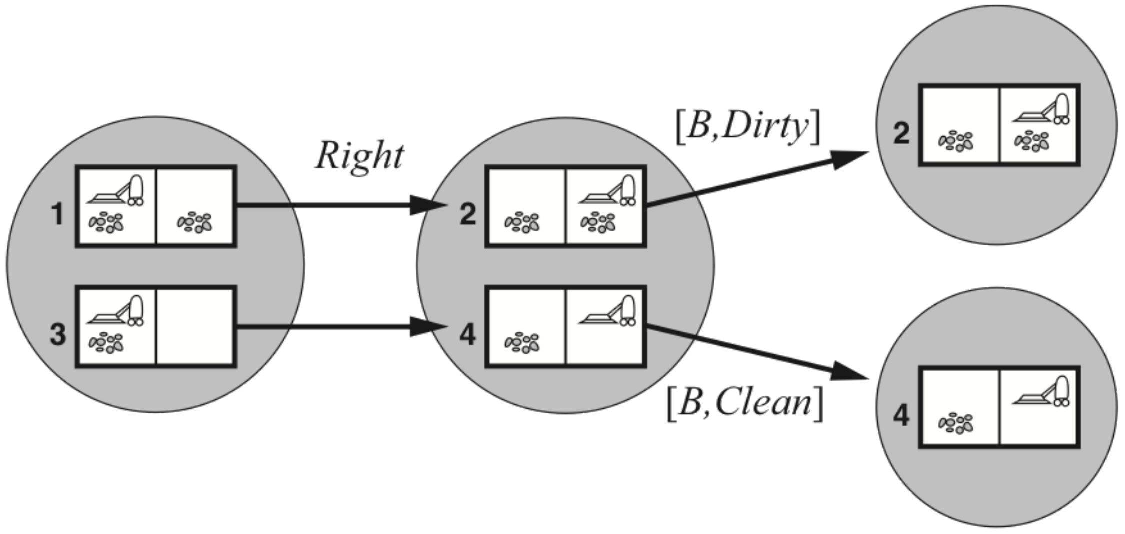
Planning in belief space



Planning in belief space

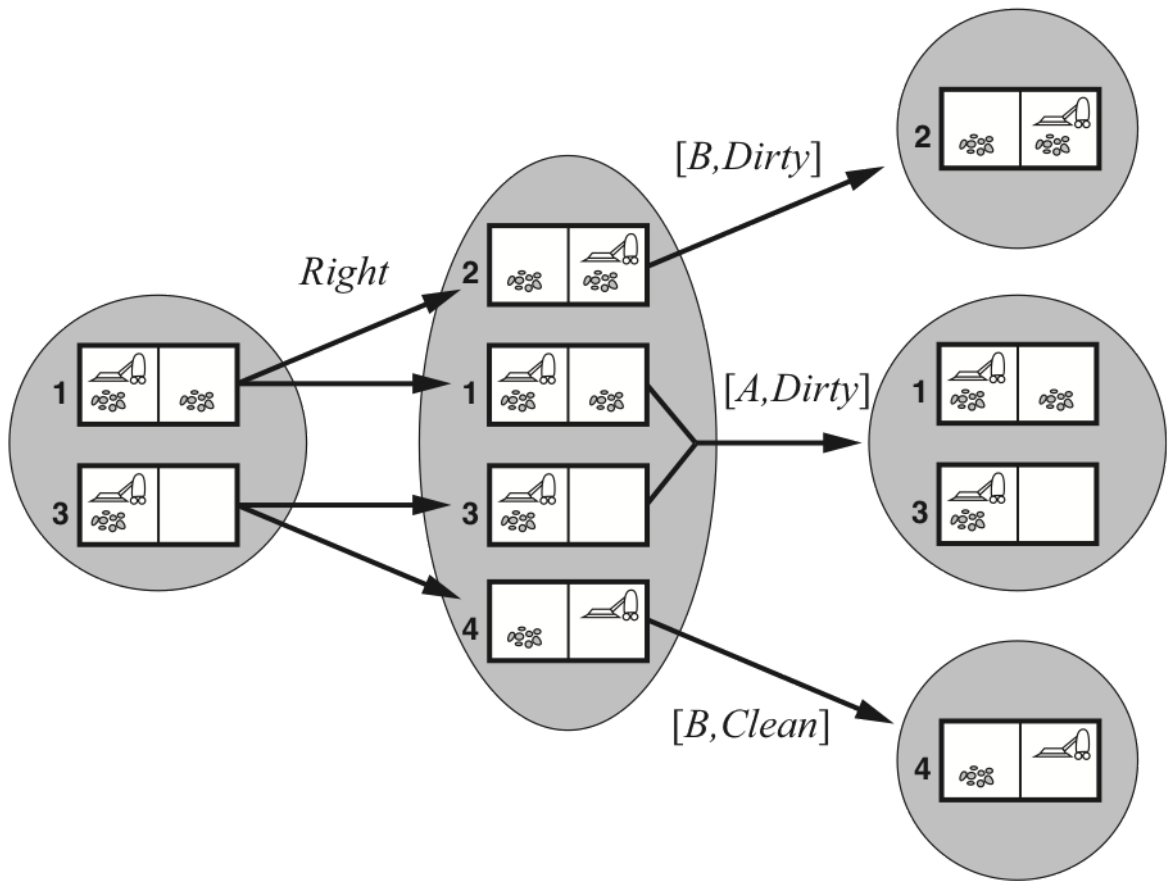
- Belief state = set of states agent could be in
- Belief state is a goal if all contained states are goals
- Successor function keeps track of all states the agent could be in after taking a particular action (“prediction”)

Sensing: Incorporating observations

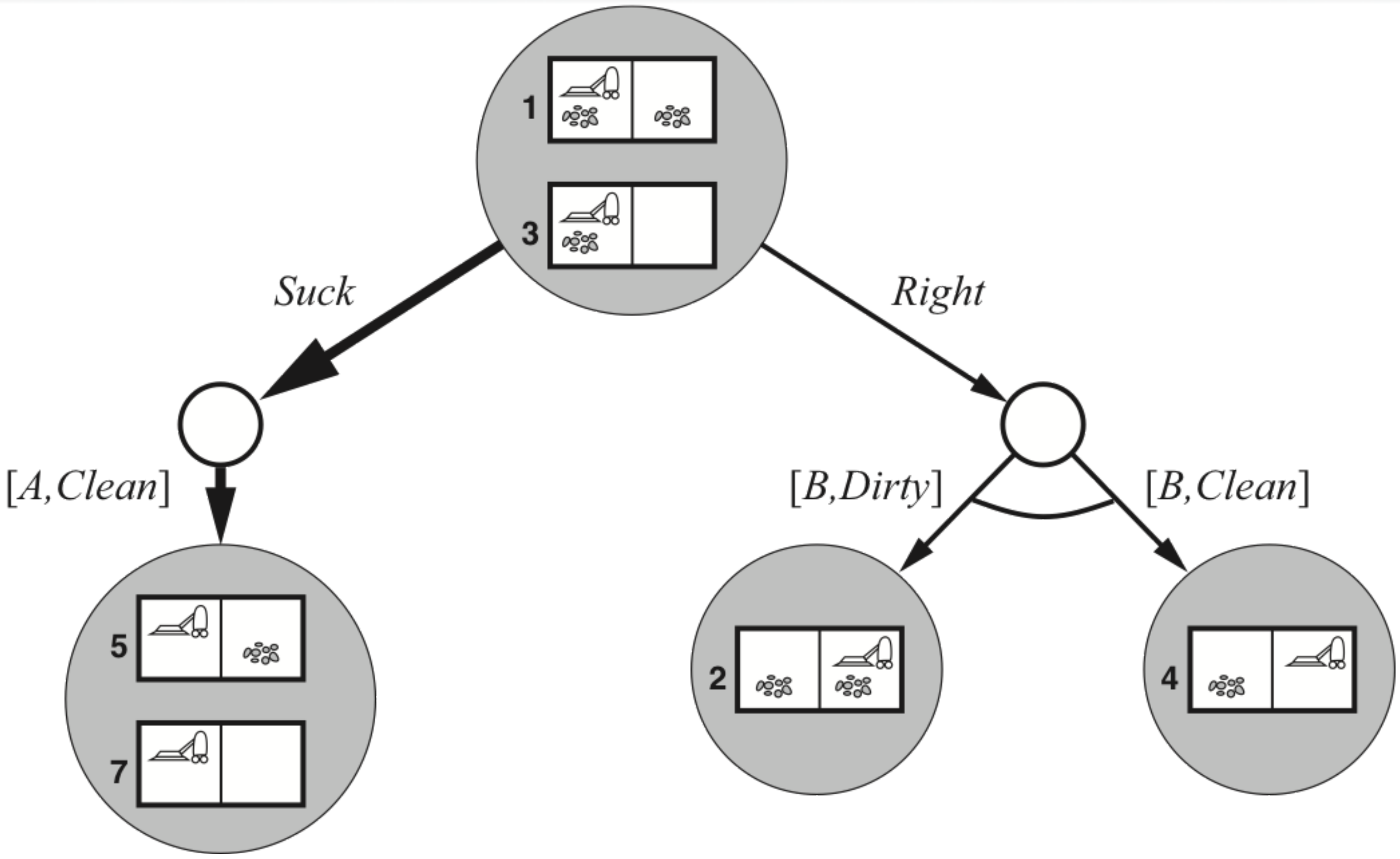


Even with deterministic actions,
Percepts can be nondeterministic
➔ Need conditional plan

Sensing: Incorporating observations



AND-OR trees with noisy observations



Summary: Partial observability

- Can reduce any partially observable problem to fully observable problem on belief states
- Belief state = set of states the environment could be in
- Use existing algorithms on belief states
  - Sensor-less case: Find opt. sequence using IDS, A*, etc.
  - Observations: Develop conditional plans using AND-OR search
  - Games: Use α-β pruning etc. on belief states

- # belief states exponential in # states…

- Need concise way to summarize the states we could be in
  ➔ logic! (coming up soon)

Constraint satisfaction problems

- So far: "black box search"
  - Environment state is arbitrary object
- CSPs:
  - state is defined by variables X_i taking values in domain D_i
  - goal test is a set of constraints
  - step cost is 0 – just need to find goal
    (or prove that constraints can't be satisfied)
- Can develop general purpose algorithms for large class of problems

Example: Map coloring

Variables:

WA, NT, SA, Q, NSW, V, T

Domains:

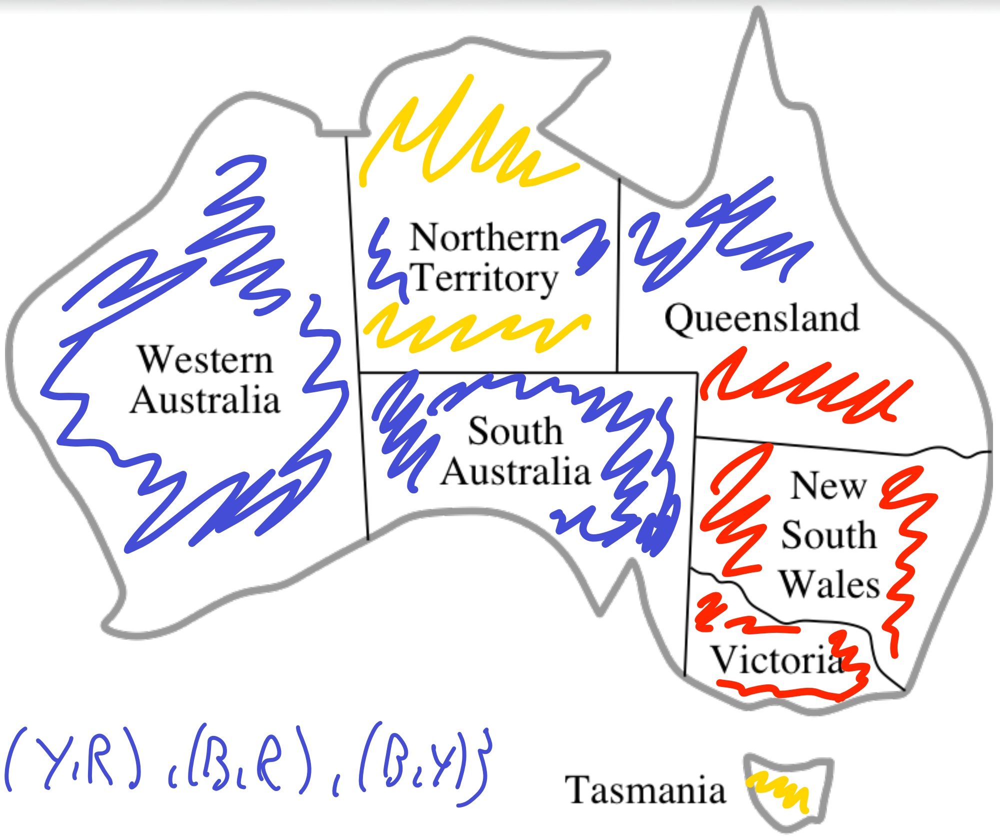
{R, B, Y}

Constraints

$WA \neq NT \wedge WA \neq SA \wedge \ldots$

$\equiv$

$(WA, NT) \in \{(R,Y), (R,B), (Y,B), (Y,R), (B,R), (B,Y)\}$



- Variables? Domains? Constraints?

Example: Sudoku

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

(a)

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

(b)

Variables: $X_{i,j}$; $i,j \in \{1..9\}$ Domain: $\{1, \dots, 9\}$

Constraints: AllDiff$(X_{1,1}, X_{2,1}, \dots, X_{9,1}) \wedge$ AllDiff ... -

(never have number occ. 2x per col./row/block)

Example: 8 queens

Variables. X_i : pos. of queen in col. i

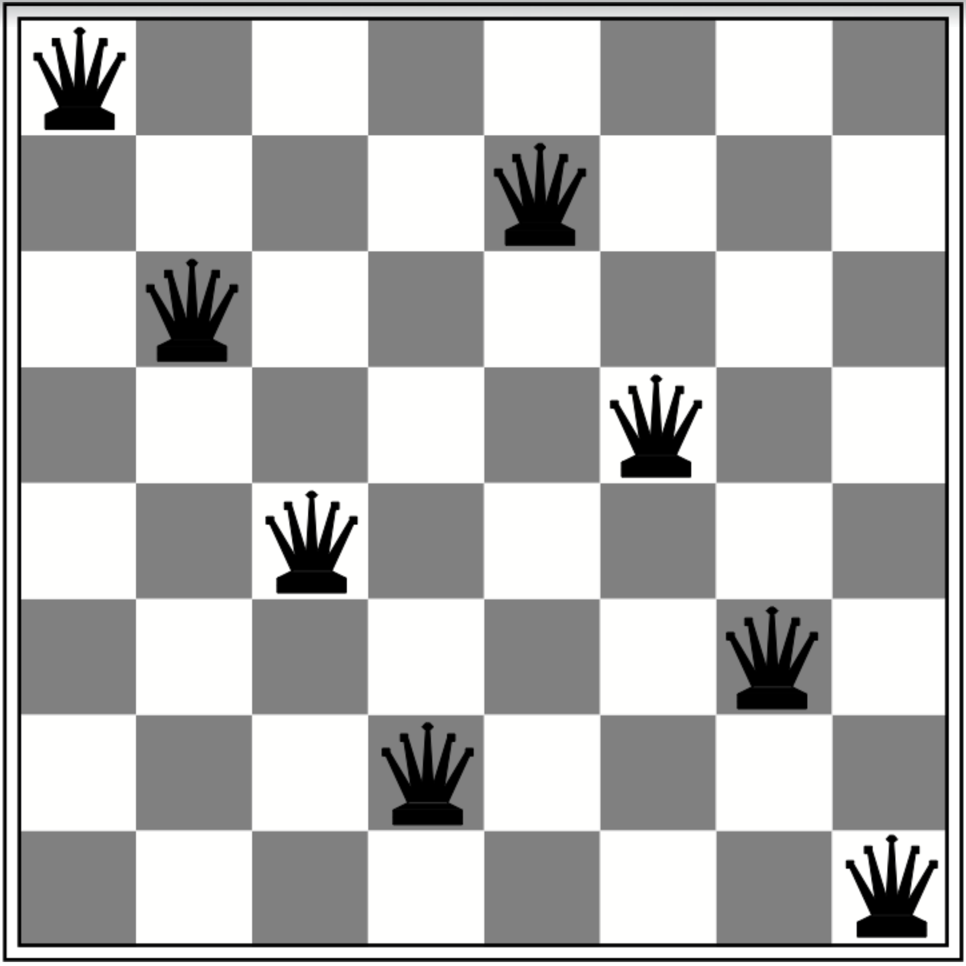
Domains: $\{1, \dots 8\}$

Constraint: No queens attack each other.

OR

Vars: X_i pos of queen i on board

Domains: $\{(1,1), (1,2), \dots (1,8)$
$(2,1), \dots \quad (2,8)$
$\vdots$
$(8,8)\}$



Constraints ...

Example: k-SAT

Disjunction of 3 literals

$$\overbrace{(X_1 \vee \neg X_3 \vee X_7)} \wedge (\neg X_1 \vee X_4 \vee X_8) \wedge \cdots \wedge (X_5 \vee \neg X_{17} \vee \neg X_{21})$$

Vars: $X_1, \dots X_n$, Domains: $\{$true, false$\}$, Constraints

- Fundamental special case
- All variables binary
- Constraints: disjunctions of k (negated) variables

- NP-complete for k>2
- Polynomial time solvable for k=2

Types of CSPs

- Discrete variables
  - This class → Finite domains : Map Coloring, Sudoku, 8 Queens, SAT
  - Infinite domains : $(X_2 \geq X_1 + 3) \wedge (X_5 \geq X_3 + 2)$

    Job scheduling

    X_i: start time of job i, Domains: $\{1, 2, \ldots\}$
- Continuous variables

  Robot/factory control, timetabling, ...

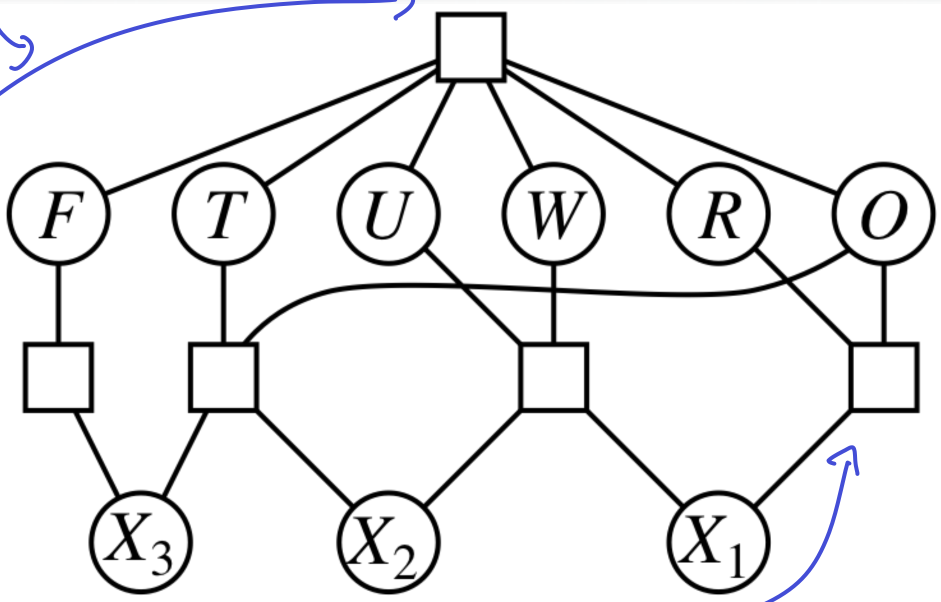
Types of constraints

- Unary: involve single variable E.g.: $NSW = B$
- Binary: involve pairs of variables $NSW \neq NT, \ldots$
- Higher-order: involve 3 or more variables
- Soft constraints: violation incurs cost
  - Constraint optimization instead of satisfaction

Example: higher-order constraints

$$\begin{array}{cccc} & T & W & O \\ + & T & W & O \\ \hline F & O & U & R \end{array}$$

order 6



order 3

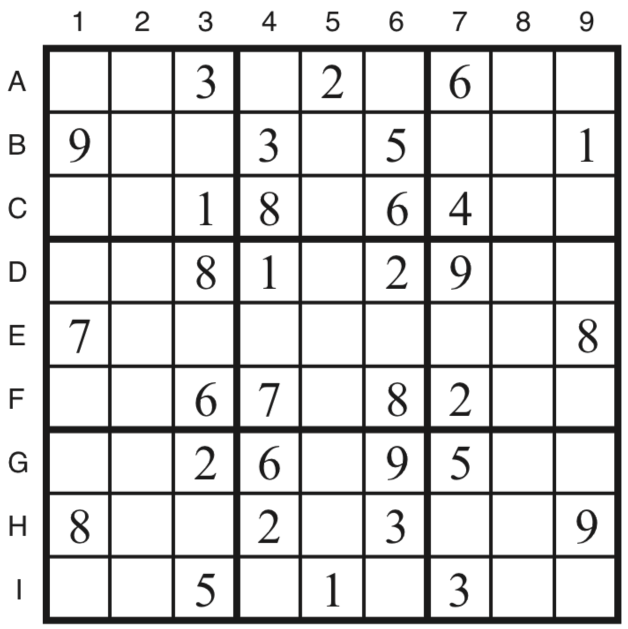
$O + O = R + 10 \cdot X_1$

$W + W + X_1 = U + 10 \cdot X_2$

$\vdots$

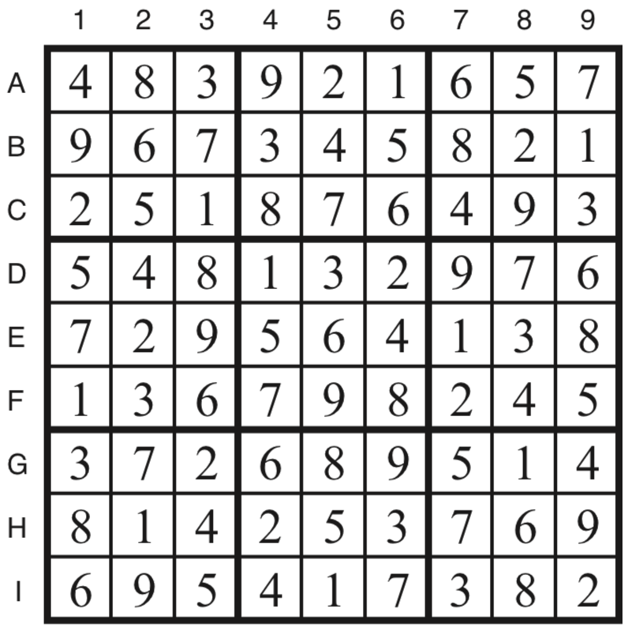
AllDiff(O, U, R, W, T, F)

Example: higher-order constraints



	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

(a)



	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

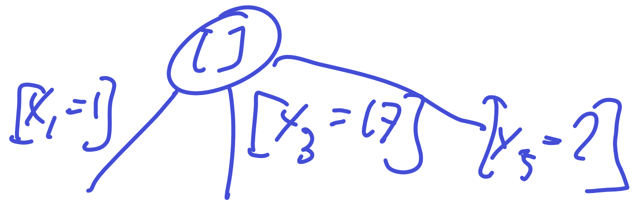
(b)

Real-world examples of CSPs

- Assignment problems
- Timetabling
- Hardware configuration
- Transportation scheduling
- Multi-agent coordination
- …

Solving CSP with search

- Naïve approach
  - State = Partial assignment to variables
  - Successor fn = Assign feasible value to some unassigned var
  - Goal test = check constraints



- Problems?

Size of search tree: $O(n!\ d^n)$

Backtracking search

- Variable assignments are commutative!

  [NSW = Y then NT = B] same as [NT = B then NSW = Y]

- Only need to consider assignments to single variable at each node

  Size of tree: $O(d^n) \ll n!\, d^n$

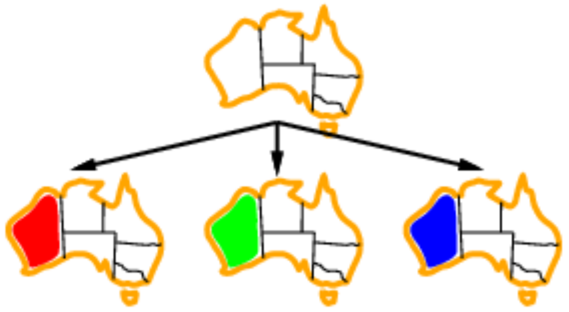
- Depth-first search with single var. assignments is called backtracking search

- Can solve 25-queens

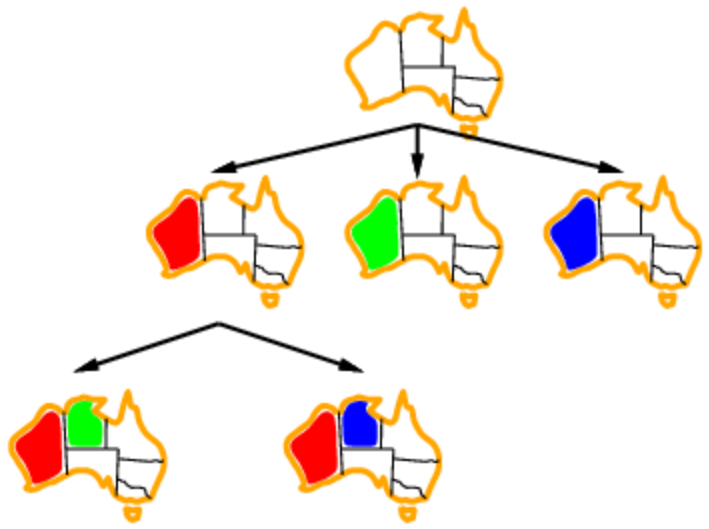
Backtracking example

Backtracking example

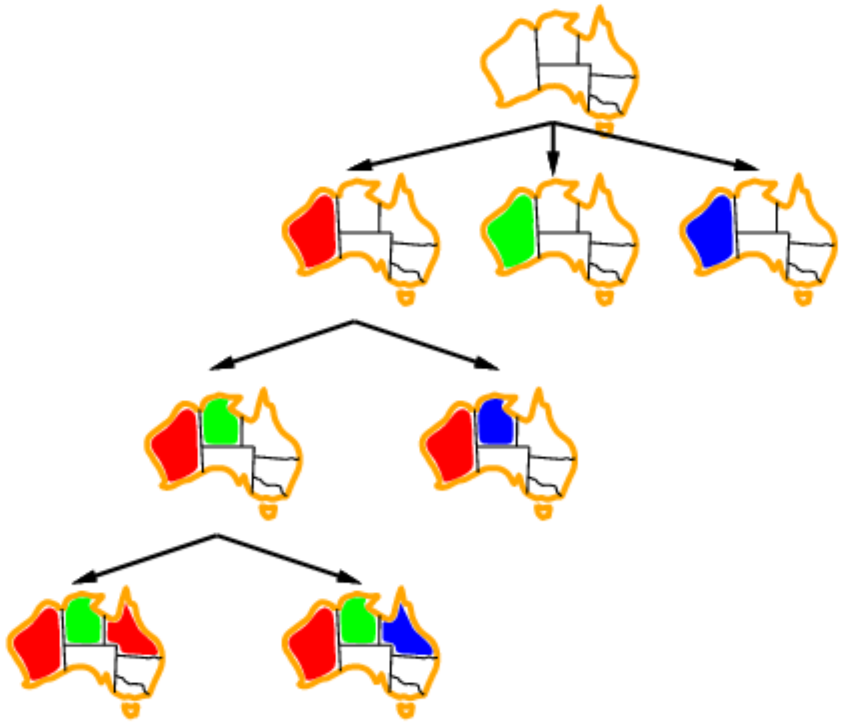
WT



Backtracking example



Backtracking example



Improving backtracking search

- General purpose methods can drastically improve speed

1. Which variable should be assigned next?
2. In what order should we try the values?
3. Can we detect inevitable failure early?
4. Can we take into account problem structure?