

Local Search: Goal Versus Path Search

CS3243: Introduction to Artificial Intelligence – Lecture 5a

7 February 2022

Contents

1. Administrative Matters
2. Goal Versus Path Search
3. Local Search via Hill-Climbing
4. Local Beam Search
6. Constraint Satisfaction Problems (CSPs)
7. CSP Formulation
8. A First Look at an Algorithm for CSPs

Administrative Matters

Midterm Quiz

- Particulars
 - Week 7 Lecture Slot - 28 February, 1000
 - Online via LumiNUS Quiz
 - Duration: 90 minutes
 - Topics: Lectures 1-5 (i.e., everything up to and including this lecture)
- Requirements
 - Environment video on secondary device
 - Screen recording on primary device
- Practice Session
 - Saturday, 12 February, 1000-1030

Upcoming...

- Deadlines
 - DQ4
 - *Due today (7 February), 2359 hrs*
 - DQ5 (released today)
 - *Due this Friday (11 February), 2359 hrs*
 - TA3
 - *Due this Sunday (13 February), 2359 hrs*
 - TA4 (released today)
 - *Due next Sunday (20 February), 2359 hrs*
 - *Refer to the tutorial assignment instructions document on LumiNUS*
 - Project 1
 - *Due next Sunday (20 February), 2359 hrs*

DQ5 and TA4
are only on
Local Search

Goal Versus Path Search

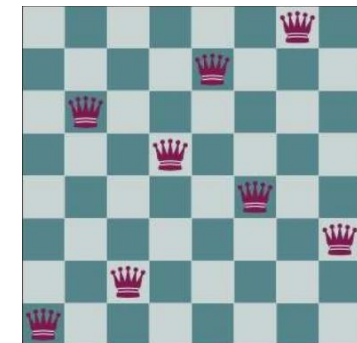
Slightly Different Problems

- Thus far: finding a path to a goal
 - Algorithms track paths
 - Systematically search paths
- What if only interested in goal state?
 - Have goal test, but not values to satisfy it
 - Only want goal state values
- Optimisation problems
 - Vertex cover problems
 - Boolean satisfiability problems (SAT)
 - Travelling salesman problem
 - Timetabling / scheduling problems

- Sudoku

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

- n-queens



Path Versus Goal

- Search problems – path planning
 - Path to a goal necessary
 - Path cost is important
- Local search – goal determination
 - Abandon systematic search – ignore path (and path cost)
 - Focus on the goal state composition – greedy approach
- Advantages
 - Only store current and immediate successor states
 - Space complexity: $O(b)$
 - Applicable to very large or finite search spaces

Local Search is incomplete

Local Search via Hill-Climbing

Hill-Climbing Algorithm

```
current = initial state
while true:
    neighbour = highest-valued successor of current
    if value(neighbour) ≤ value(current) return current
    current = neighbour
```

- How it works (steepest ascent – greedy strategy)
 - Only store the current state
 - In each iteration, find a successor that *improves* on current state
 - Requires **actions** and **transition** to determine successors
 - Requires **value**; a way to value each state – e.g., $f(n) = -h(n)$
 - If none exists, return current state as the best option
 - This algorithm *can fail*; may return a non-goal state

8-Queens Example

- Given an 8×8 chess board
 - Place 8 queens
 - No queen must threaten another
 - Use h: *pairs of queens threatening each other*
- Search problem
 - State: 1 queen per column
 - Action: move 1 queen to different col. position
 - Goal: 0 pairs threatening
- Example h
 - Consider top-most left-most cell (18)
 - C1 attacks C2, C3, C5
 - C2 attacks C3, C4, C6, C8
 - C3 attacks C5, C7
 - C4 attacks C5, C6, C7
 - C5 attacks C6, C7
 - C6 attacks C7, C8
 - C7 attacks C8

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♔	13	16	13	16
♔	14	17	15	♔	14	16	16
17	♔	16	18	15	♔	15	♔
18	14	♔	15	15	14	♔	16
14	14	13	17	12	14	12	18

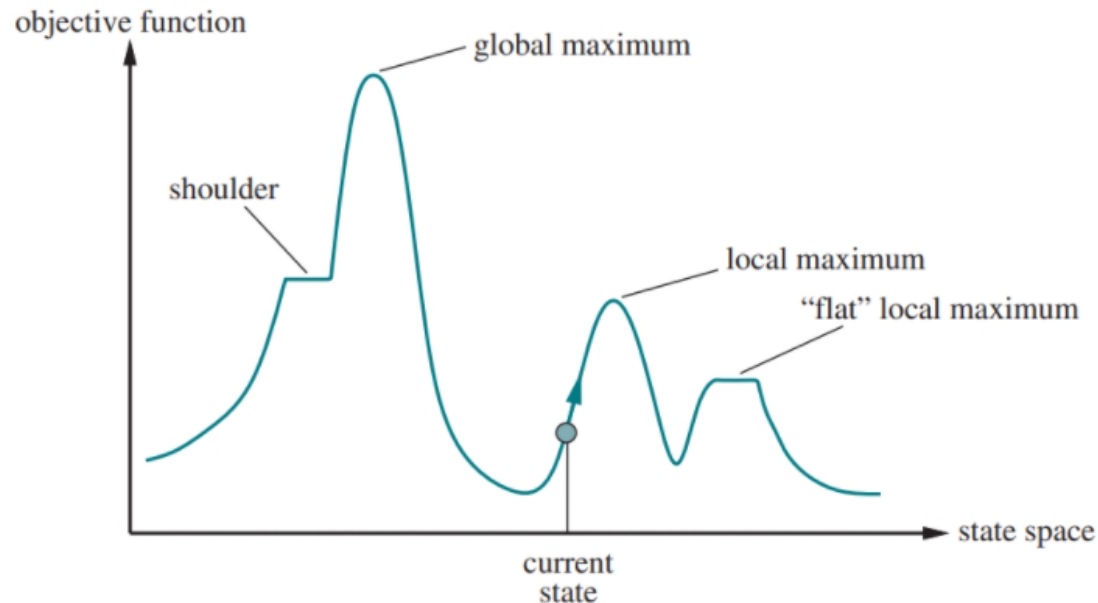
Complete-State Formulations

- States in the 8-Queens search problem have all 8 queens present
- Every state has all components of a solution
 - No partially completed states
- Each state is a potential solution
 - Apt for problems where path is not important
 - Simply “guess” a solution
 - “Check” its value
 - Make a “systemic guess” by moving to states of higher value (e.g., via $f(n) = -h(n)$)
 - Assumes that states with higher f values are closer to the goal (i.e., more likely to reach a goal)
- Most local search problems may be formulated in this manner

Practically, it is fine to use $f(n) = h(n)$ and seek a local minima as well. In such cases, we simply replace the \leq in the algorithm with \geq .

Issues & the Potential for Failure

- Hill-climbing may not return a solution



- May get stuck at
 - Local Maxima
 - Shoulder or Plateau
 - Ridge (sequence of local maxima)
- Require strategies to counter these problems

Hill-Climbing Variants

- Sideways move
 - Replaces \leq with $<$; allows continuation when **value(neighbour) == value(current)**
 - Can traverse shoulders / plateaus
- Stochastic hill climbing
 - Chooses randomly among states with values better than current
 - May take longer to find a solution but sometimes leads to better solutions
- First-choice hill climbing
 - Handles high b by randomly generating successors until one with better value than current is found (instead of generating all possible successors)
- Random-restart hill climbing
 - Adds an outer loop which randomly picks a new starting state
 - Keeps attempting random restarts until a solution is found

Back to 8-Queens: Analysis

- Hill climbing (via steepest-ascent) with random restarts
 - Solution: $p_1 = 14\%$ (expected solution in 4 steps; expected failure in 3 steps)
 - Expected computation = $1(\text{steps for success}) + ((1 - p_1) / p_1)(\text{steps for failure})$
 $= 1(4) + (0.86/0.14)(3)$
 $= 22.428571428571427$ steps
- Adding sideways moves
 - Solution: $p_2 = 94\%$ (expected solution in 21 steps; expected failure in 64 steps)
 - Expected computation = $1(\text{steps for success}) + ((1 - p_1) / p_1)(\text{steps for failure})$
 $= 1(21) + (0.06/0.94)(64)$
 $= 25.085106382978722$ steps
- 8-Queens possible states = $8^8 = 16777216$

Extremely efficient for such a large space

Expected values taken
from AIMA pp. 131

Local Beam Search

Local Beam Search

- Store k states instead of 1
 - Hill climbing just stores the current state
 - Beam (window) stores k
- Algorithm
 - Begins with k random starts
 - Each iteration generate successors for all k states
 - Repeat with best k among successors unless goal found
- Better than k parallel random restarts
 - Since best k among ALL successors taken (not best from each set of successors, k times)
- Stochastic beam search
 - Original variant may still get stuck in a local cluster
 - Adopt stochastic strategy similar to stochastic hill climbing to increase state diversity

Questions on the Lecture so far?

- Was anything unclear?
- Do you need to clarify anything?
- Channels
 - Verbally on Zoom
 - On Archipelago
 - Via Zoom Chat



OR <https://archipelago.rocks/app/resend-invite/29794382657>

Constraint Satisfaction Problems: Generalising Goal Search I

CS3243: Introduction to Artificial Intelligence – Lecture 5b

7 February 2022

Systematic Goal Search

- With local search we apply greedy search strategies
 - Are there more *systematic* search strategies applicable?
- Issues with systematic searching
 - Systematic approaches tend to be computationally expensive
 - Incorporating domain knowledge via heuristics helped direct the search such that less was searched
 - Need to reduce the search space to make a systematic search more viable
- A general solution
 - Use a factored representation for each state
 - State: set of variables $X = \{x_1, \dots, x_n\}$, where each variable x_i has a domain $D_i = \{d_1, \dots, d_m\}$
 - Divide the goal test into a set of constraints
 - If a state satisfies all constraints, it is a goal state
 - Constraint satisfaction problem (CSP)
 - Any state that does not satisfy a constraint should not be further explored

CSPs systematically search for goal states by pruning invalid subtrees as early as possible

CSP Formulation

Formulating CSPs

- State representation
 - Variables: $X = \{x_1, \dots, x_n\}$
 - Domains: $D = \{d_1, \dots, d_k\}$
 - Such that x_i has a domain d_i
 - Initial state: all variables unassigned
 - Intermediate state: partial assignment
- Goal test
 - Constraints: $C = \{c_1, \dots, c_m\}$
 - Defined via a constraint language
 - Algebra, Logic, Sets
 - Each c_i corresponds to a requirement on some subset of X
 - Objective is a **complete** and **consistent** assignment
 - Find a legal assignment (y_1, \dots, y_n)
 - $y_i \in d_i$ for all $i \in [n]$
 - Complete: all variables assigned values
 - Consistent: all constraints C satisfied
- Actions, costs and transition
 - Assignment of values (within domain) to variables
 - Costs are not utilised

CSP Formulation Example 1: Graph Colouring

- Colour each state of Australia such that no two adjacent states share the same colour
- Variables
 - $X = \{ WA, NT, Q, NSW, V, SA, T \}$
- Domains
 - $d_i = \{ \text{Red, Green, Blue} \}$
- Constraints
 - $\forall (x_i, x_j) \in E, \text{colour}(x_i) \neq \text{colour}(x_j)$



CSP Formulation Example 2: Cryptarithmic Puzzle

- Given that each letter represents a digit, determine the letter-digit mapping that solves the given sum
- Variables
 - $X = \{ T, W, O, F, U, R, B_1, B_2, B_3 \}$
 - Where B_1, B_2, B_3 are carry bits for $(2O, 2W, 2T)$ respectively)
- Domains
 - $d_i = \{ 1, 2, 3, 4, 5, 6, 7, 8, 9 \}$
 - Strictly, B_1, B_2, B_3 should have domain $\{0, 1\}$

	<i>T</i>	<i>W</i>	<i>O</i>	
+	<i>T</i>	<i>W</i>	<i>O</i>	
<hr/>				
<i>F</i>	<i>O</i>	<i>U</i>	<i>R</i>	

- Constraints
 - **alldiff**(T, W, O, F, U, R)
 - $O + O = R + 10.B_1$
 - $B_1 + W + W = U + 10.B_2$
 - $B_2 + T + T = O + 10.B_3$
 - $B_3 = F$
 - $T, F \neq 0$

CSP Formulation Example 3: Sudoku

- Variables
 - $X = \{ A_1, \dots, A_9, \dots, I_1, \dots, I_9 \}$
 - 81 variables
- Domains
 - $d_i = \{ 1, 2, 3, 4, 5, 6, 7, 8, 9 \}$
- Constraints
 - alldiff(...)**
 - 27 cases
 - 9 columns
 - 9 rows
 - 9 boxes

	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		

	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

Variable Domain Types & Constraint Types

- Variable domain types

- Continuous
- Discrete
- Continuous and Infinite
 - Real values
- Discrete and Infinite
 - All integers
- Discrete and finite
 - Sudoku

CS3243 focuses on
discrete, finite domains

- Constraint types

- Linear
- Nonlinear

Continuous domain and linear
constraints → linear programming

Not covered in CS3243

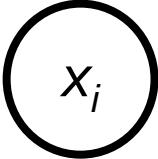

More on Constraints

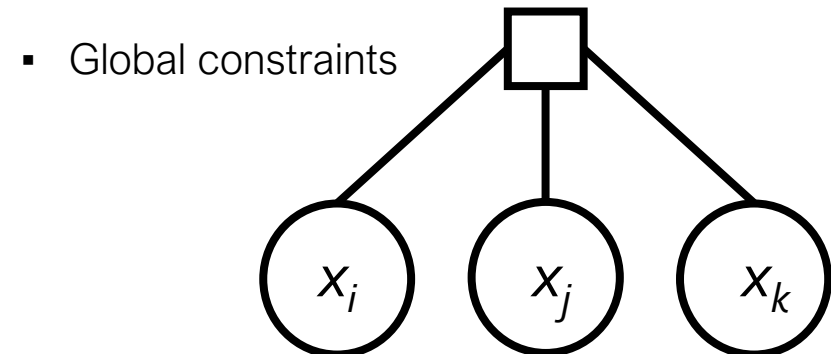
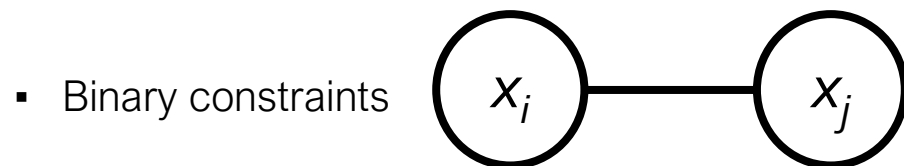
- A language is necessary to express the constraints
 - Arithmetic
 - Sets (of legal values)
 - Logic
- For example, x_1 greater than x_2 given $d = \{1, 2, 3\}$ may be written
 - $\langle (x_1, x_2), x_1 > x_2 \rangle$
 - $\langle (x_1, x_2), \{ (2, 1), (3, 1), (3, 2) \} \rangle$
- Each constraint, c_i ,
 - Describes the necessary relationship, **rel**, between a set of variables, **scope**
 - For the example above, **scope** = (x_1, x_2) . **rel** = $x_1 > x_2$
- Types of constraints
 - Unary: $|\text{scope}| = 1$
 - Binary: $|\text{scope}| = 2$
 - Global: $|\text{scope}| > 2$ (i.e., higher-order constraints)

Constraint Graphs

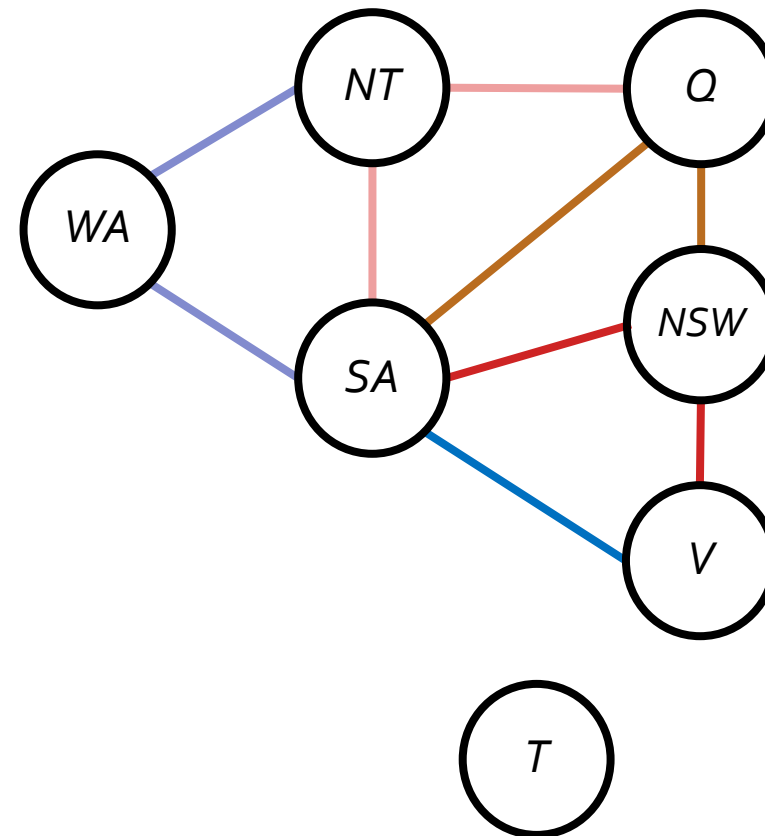
Drawing Constraint Graphs and Hypergraphs

- Constraint graphs represent the constraints in a CSP

- Simple Vertex: variable 
- Linking Vertex: for global constraints 
- Edge: links all variables in the scope of a constraint (*rel*)



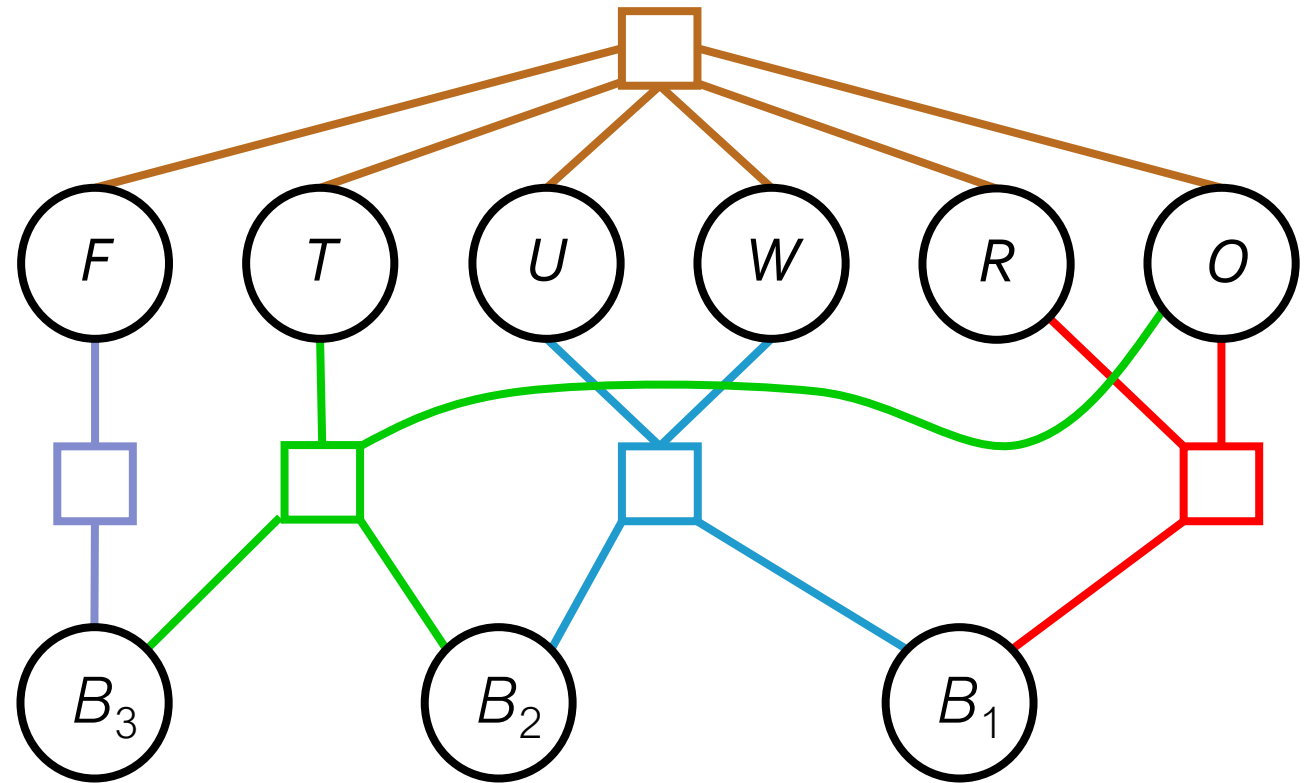
Constraint Graph for Example 1: Graph Colouring



Constraint Graph for Example 2: Cryptarithmic Puzzle

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

- Constraints
 - $\mathbf{alldiff}(T, W, O, F, U, R)$
 - $O + O = R + 10.B_1$
 - $B_1 + W + W = U + 10.B_2$
 - $B_2 + T + T = O + 10.B_3$
 - $B_3 = F$
 - $T, F \neq 0$



A First Look at an Algorithm for CSPs

General Idea for the Algorithm

```
assignments = initial state (no assignments made)
while assignments incomplete:
    if no possible assignments left return failure
    current = assign a value to non-assigned variable
    if current consistent then assignments.store(current)
return assignments
```

- Applicable to all CSPs
- Search path irrelevant
 - May use complete-state formulation
- All solutions require $|X| = n$ assignments

Which algorithm should be used?

DFS

Search Tree Size

- Example CSP

- $X = \{A, B, C, D\}$
- All domains: $d = \{1, 2, 3\}$
- No constraints

- Analysis

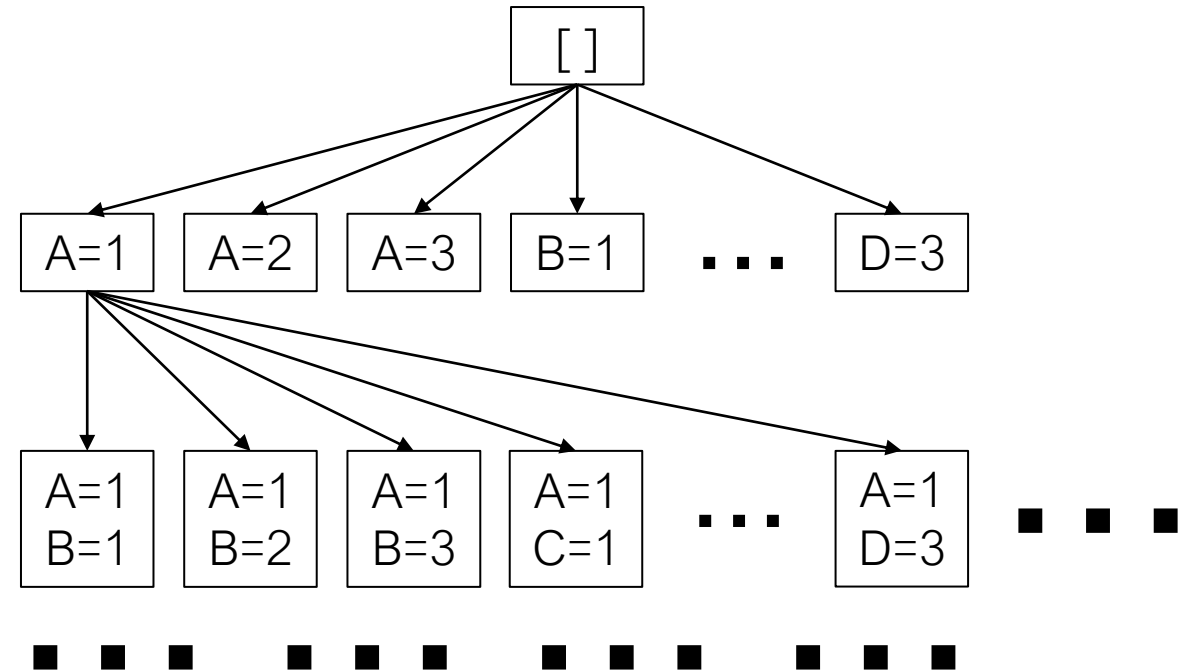
b at depth 1: 4 variables \times 3 values = 12 states
b at depth 2: 3 variables \times 3 values = 9 states
b at depth 3: 2 variables \times 3 values = 6 states
b at depth 4: 1 variables \times 3 values = 3 states

At depth ℓ : $(|X| - \ell) \cdot |d|$ states

Total number of leaf states:

$$nm \times (n-1)m \times (n-2)m \times \dots \times 2m \times m = n!m^n$$

where $n = |X|$ and $m = |d|$



Order of variable assignments not important
Just consider assignments to ONE variable per level (m^n leaves)

Basic uninformed search for CSPs: Backtracking

Backtrack when no legal assignments

Backtracking Algorithm for CSPs

function BACKTRACKING-SEARCH(*csp*) **returns** a solution or *failure*
 return BACKTRACK(*csp*, { })

function BACKTRACK(*csp*, *assignment*) **returns** a solution or *failure*

if *assignment* is complete **then return** *assignment*

var ← SELECT-UNASSIGNED-VARIABLE(*csp*, *assignment*)

Determine the variable to assign to

for each *value* **in** ORDER-DOMAIN-VALUES(*csp*, *var*, *assignment*) **do**

Determine the value to assign

if *value* is consistent with *assignment* **then**

 add {*var* = *value*} to *assignment*

inferences ← INFERENCE(*csp*, *var*, *assignment*)

Trying to determine if the chosen assignment will lead to a terminal state

if *inferences* ≠ *failure* **then**

 add *inferences* to *csp*

result ← BACKTRACK(*csp*, *assignment*)

Continues recursively as long as the *assignment* is *viable*

if *result* ≠ *failure* **then return** *result*

 remove *inferences* from *csp*

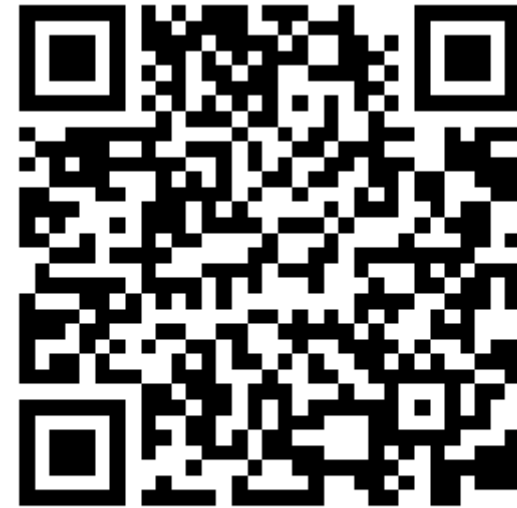
 remove {*var* = *value*} from *assignment*

return *failure*

We will look into making these choices in the next lecture

Questions on the Lecture?

- Was anything unclear?
- Do you need to clarify anything?
- Channels
 - Verbally on Zoom
 - On Archipelago
 - Via Zoom Chat



OR <https://archipelago.rocks/app/resend-invite/29794382657>