COMS W4701: Artificial Intelligence

Lecture 5: Constraint Satisfaction Problems

Tony Dear, Ph.D.
Department of Computer Science
School of Engineering and Applied Sciences

Today

- Constraint satisfaction problems
- Backtracking search
- Constraint propagation and local consistency
- Heuristics on problem structure and constraints

States with Structure

- So far, we have reasoned explicitly in terms of states as "black boxes"
- Oftentimes, we can also describe states using a common set of *features*

- Features can be denoted using **variables** and assigned **values**
- We may have hard **constraints** or **preferences** over assignments

- The planning problem (search for goal) is now an *assignment* problem
- We can also apply more *general*, rather than problem-specific, heuristics

Constraint Satisfaction Problems

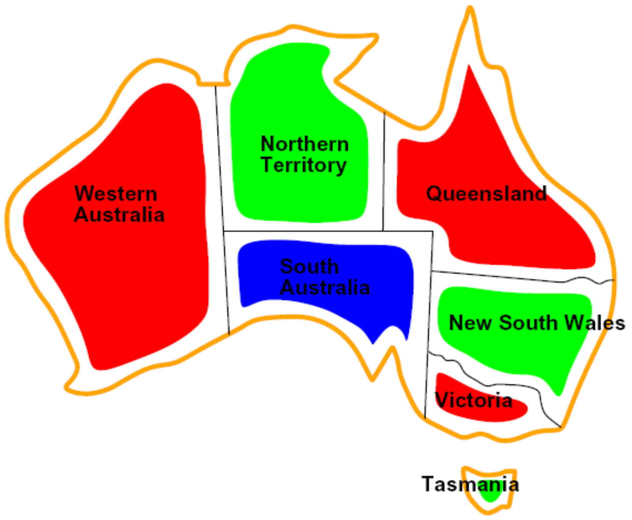
- Special structured search problems with 3 components
  - Variables (e.g., discrete, binary, Boolean, continuous, etc.): $X = \{X_1, \ldots, X_n\}$
  - Domain for each variable (may be the same for some/all): $D = \{D_1, \ldots, D_n\}$
  - Constraints over variables (e.g., unary, binary, trinary, etc.): $C = \{C_1, \ldots, C_m\}$

- Constraints may be defined *implicitly*, e.g., using formulas, logic, or relations
- May also be defined *explicitly*, i.e., as a set of all allowed assignments

- Goal test: A **complete, consistent** assignment of values to each variable X_i from respective domain D_i s.t. all constraints C are satisfied

Example: Map Coloring

- Goal: Color a map so that no adjacent territories have the same color
- Variables: $X = \{\text{WA, NT, Q, NSW, V, SA, T}\}$
- Domains: $D_i = \{\text{red}, \text{green}, \text{blue}\}$
- Constraints: Implicit vs explicit representation
  - $C = \{\text{WA} \neq \text{NT}, \text{WA} \neq \text{SA}, \text{NT} \neq \text{SA}, \text{NT} \neq \text{Q}, \text{SA} \neq \text{Q}, \text{Q} \neq \text{NSW}, \text{NSW} \neq \text{V}\}$
  - $C = \{(\text{WA}, \text{NT}) \in \{(\text{red}, \text{green}), (\text{red}, \text{blue}), (\text{green}, \text{red}), (\text{green}, \text{blue}), \dots\}, \dots\}$

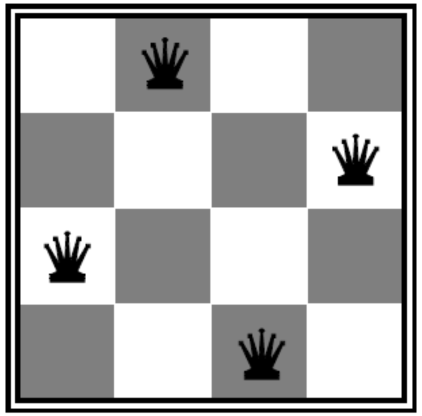


Example: n-Queens

- Place n queens on $n \times n$ board s.t. none share a row, column, or diagonal
- Can come up with more than one CSP representation

- CSP 1: A Boolean variable for each cell (n^2 total)
- Constraints: Every set of row, column, and diagonal cells contains at most one queen

- CSP 2: A variable X_i for row i with domain $\{0,1,2,3\}$ (4 total)
- Constraints: $\forall i, j, X_i \neq X_j$ and $X_j - X_i \neq |j - i|$



More Examples

- **Cryptarithmetic**
- Variables: $\{T, W, O, F, U, R, C_1, C_2\}$
- Domains: $\{0, \dots, 9\}$
- Constraints: All variables different, and each column satisfies addition relation accounting for carryover variables C_1 and C_2

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

- **Sudoku**
- Variables: One for each empty cell with domain $\{1, \dots, 9\}$
- Constraints: Different values in each row, col, 3x3 square

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

Backtracking Search

- Naïve idea: Simply test all complete assignments for consistency
- For n variables with domain size d, total number of assignments is d^n
- We don’t have to check all of them, since some will violate constraints

- Better idea: *Incrementally* assign one variable at a time
- Only assign consistent values so that a constraint is never violated

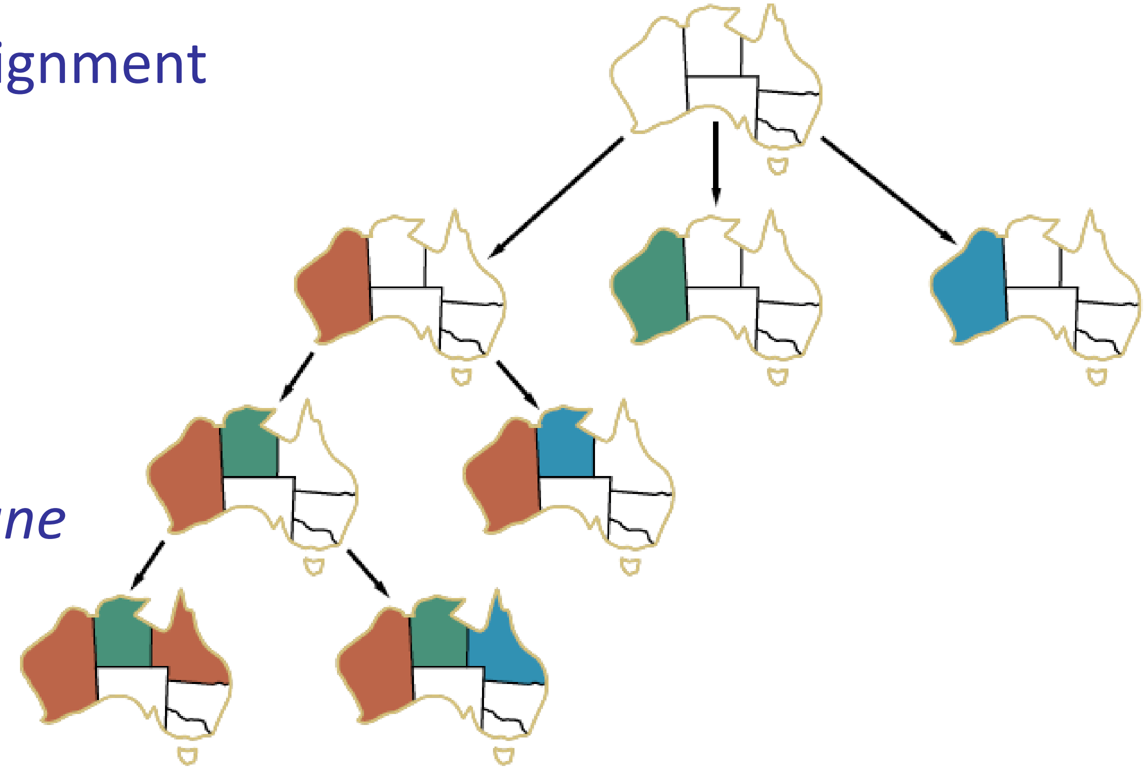
- If we cannot progress further from a partial assignment, *backtrack* and change some current assignments to try alternatives

Backtracking Search

- Backtracking search is just a specific case of DFS!
- States are partial assignments, goal is a complete assignment

- Each search iteration tries to make a new assignment
- If we cannot do so, DFS *backtracks* and considers the next closest partial assignment

- Same time and space complexities as DFS
- May be more efficient as constraints help *prune* invalid assignments from the search tree



Constraint Propagation

- Before or during search, we can also *reduce the domains* of variables according to the constraints so that they satisfy **local consistency**

- **Node (domain) consistency**: Remove all domain values violating unary constraints
  - Should always be done prior to starting any search process

- **Arc consistency**: Remove all domain values violating binary constraints
  - X is consistent wrt Y if $\forall x \in D_1, \exists y \in D_2$ s.t. $(X = x, Y = y)$ is consistent

- To make X consistent with Y, check all values of and potentially remove values from D_1
- To make Y consistent with X, check all values of and potentially remove values from D_2

Example: Arc Consistency

- Variables A, B, C, D with domains $\{1,2,3,4\}$
- Constraints $c_1: A < B$ and $c_2: B < C$

- Make A consistent with B: $A = 4$ does not satisfy c_1, so reduce A domain to $\{1,2,3\}$
- Make B consistent with A: $B = 1$ does not satisfy c_1, so reduce B domain to $\{2,3,4\}$

- Make B consistent with C: $B = 4$ does not satisfy c_2, so reduce B domain to $\{\mathbf{2}, \mathbf{3}\}$
- Make C consistent with B: $C = 1,2$ both cannot satisfy c_2, so reduce C domain to $\{\mathbf{3}, \mathbf{4}\}$

- Since B's domain changed, $A = 3$ no longer satisfies c_1, so reduce A domain to $\{\mathbf{1}, \mathbf{2}\}$

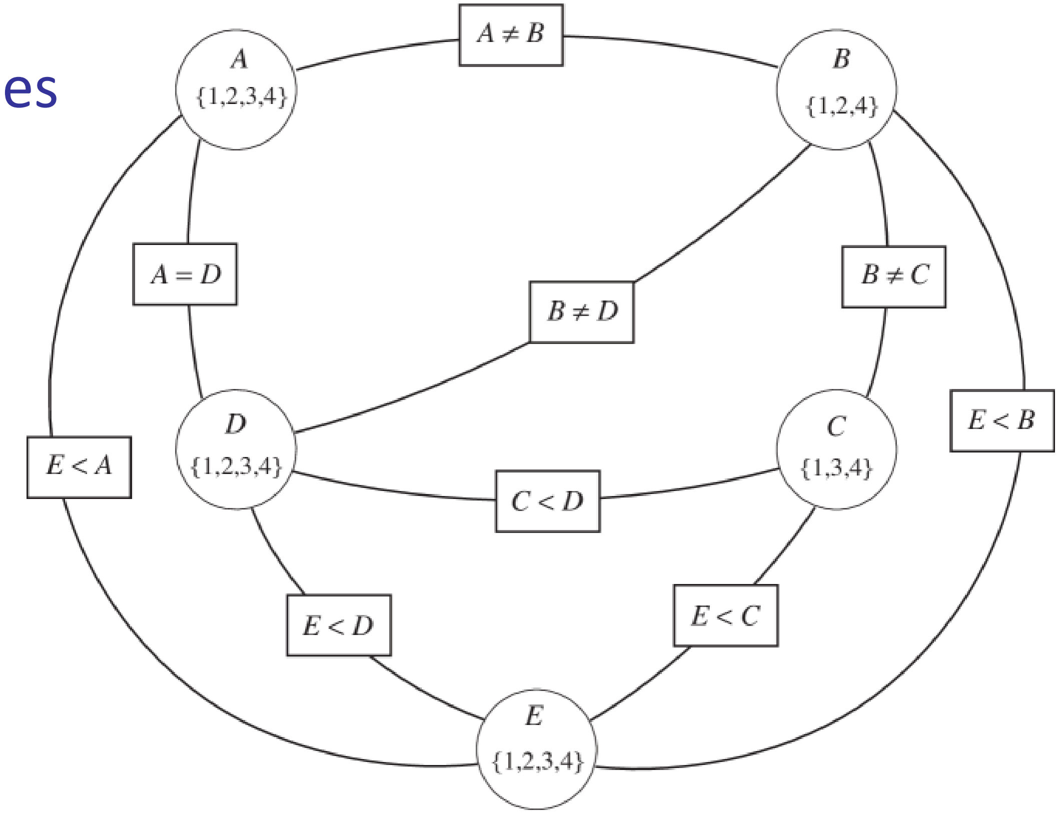
AC-3 Algorithm

- CSP representation as a **constraint graph** where variable vertices are adjacent to constraint vertices

- Algorithm: Initialize a queue containing all constraints to check for consistency

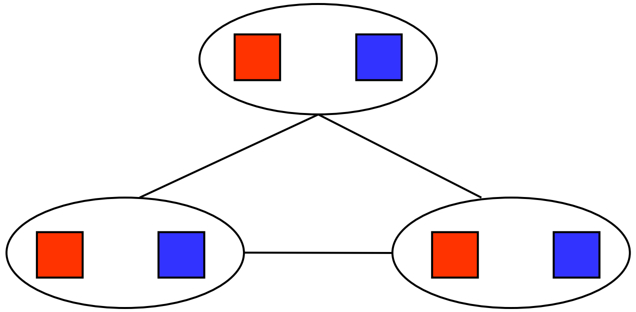
- While queue not empty, pop a constraint and modify neighbor domains for consistency

- If domain of X changes, all constraint neighbors must be *re-added* to the queue to recheck



AC-3 with Search

- AC-3 complexity: Each constraint check takes $O(d^2)$ time
- For each value in D_i, check each value in D_j (and vice versa)
- There are m constraints, each may be checked up to d times
- Overall time complexity is thus $O(md^3)$—not too bad!



- AC-3 ends when entire CSP is arc-consistent or no assignment is possible
- A arc-consistent CSP may have no solution discovered upon further search

- Can look at higher orders of consistency, but the tradeoff is more computation
- **Path consistency**: $\forall(x_1, x_2) \in D_1 \times D_2, \exists x_3 \in D_3$ s.t. (x_1, x_2, x_3) is consistent

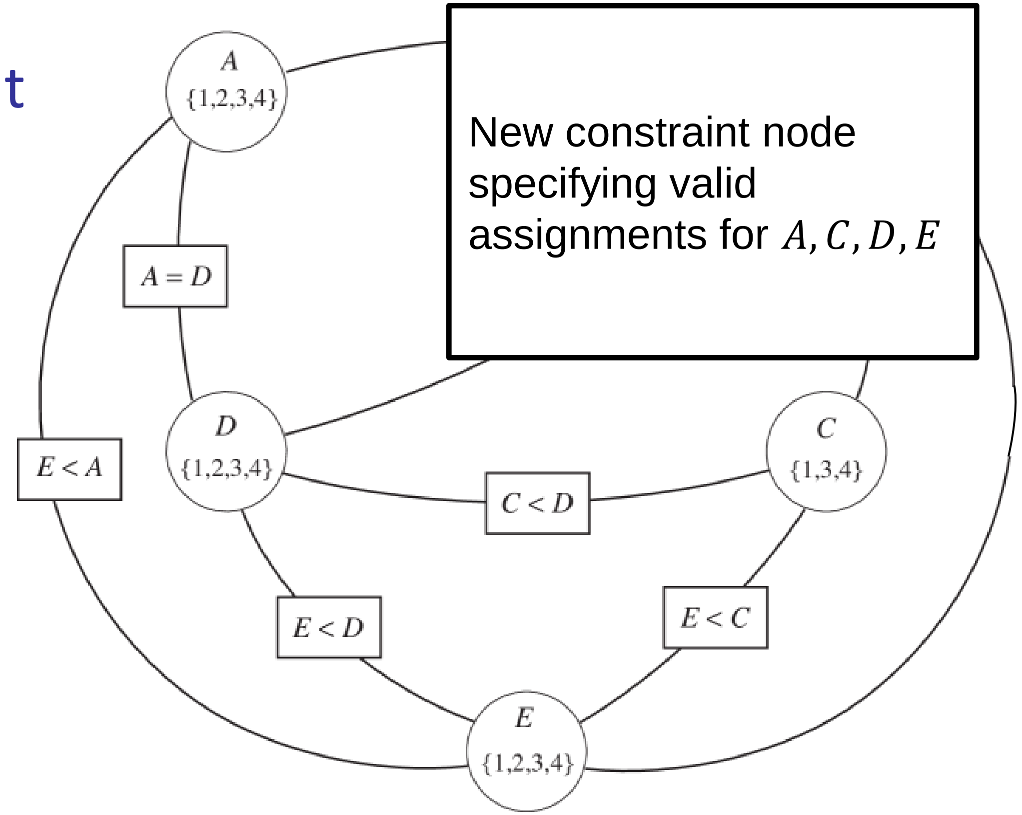
Variable Elimination

- Another class of strategies exploits and modifies the constraint graph *structure*
- We can *shrink* the graph by eliminating one or more variables

- We can replace a variable X with a new constraint specifying valid assignments for its neighbors

- Alternatively, we can assign X to different values in its domain, ensure arc consistency with neighbors, and then remove it from the graph

- In both cases, the resultant graph will have simpler structure than the original



Heuristics on Constraints

- **Bounds propagation**: If constraints are equality or inequality (e.g., *resource constraints*), we can use them to tighten finite domain bounds
  - Ex: X_1, X_2, X_3, all with domains $\{1,2,3,4,5\}$
  - Constraint $\sum X_i = 13$: Reduce domains to $\{3,4,5\}$
  - Constraint $\sum X_i \leq 5$: Reduce domains to $\{1,2,3\}$

- If a CSP has multiple solutions due to *value symmetry* or assignment permutations, we can *introduce* **symmetry-breaking constraints** to help reduce domain sets
  - Ex: $X_1 \neq X_2$ can be replaced with $X_1 < X_2$ if it does not affect other variables/constraints

- **Constraint learning**: When we see inconsistent assignments in backtracking or constraint propagation, record it as a new constraint

Summary

- CSPs are structured search problems with variables, domains, constraints
- Solution consists of a complete, consistent assignment (goal, not path)

- Backtracking search performs incremental assignments while satisfying constraints, behaves like DFS
- Constraint propagation ensures local consistency of variable domains

- Other heuristics may involve modifying or adding new constraints