COMP3702/COMP7702
Artificial Intelligence

Module 2: Reasoning and planning with certainty — Part 1

Dr Archie Chapman

Semester 2, 2020

The University of Queensland
School of Information Technology and Electrical Engineering

Thanks to Dr Alina Bialkowski and Dr Hanna Kurniawati

Module 2: Reasoning and planning with certainty

Using logic to represent a problem.

Two types of problems:

- Satisfiability (today)
- Validity (later)

Table of contents

Constraint Satisfaction Problems

Constraint Satisfaction Problems

Constraint satisfaction problems (CSPs) are a subset of search problems. They have the same assumptions about the world:

- a single agent,
- deterministic actions,
- fully observed state,
- (typically) discrete state space

CSP are specialised to **identification** problems, or to provide assignments to variables:

- The goal itself is important, not the path
- All paths at the same depth (for most formulations)

At the end of this part you should be able to:

- recognise and represent constraint satisfaction problems
- show how constraint satisfaction problems can be solved with search
- implement and trace arc-consistency of a constraint graph

Constraint Satisfaction Problems: Definition

A Constraint Satisfaction Problem (CSP) is given by:

- A set of variables, $V_1, V_2, \ldots, V_n$.
- Each variable V_i has an associated domain, dom_{V_i}, of possible values.
- A set of constraints on various subsets of the variables which are logical predicates specifying legal combinations of values for these variables.
- A **model** of a CSP is an assignment of values variables that satisfies all of the constraints

---

There are also constraint **optimisation** problems, in which there is a function that gives a cost for each assignment of a value to each variable:

- A **solution** is an assignment of values to the variables that minimises the cost function.
- We will skip constraint optimisation problems (despite your lecturer's keen interest in them).

Example: scheduling activities

- **Variables:** $X = \{A, B, C, D, E\}$ that represent the starting times of various activities.
- **Domains:** Four start times for the activities

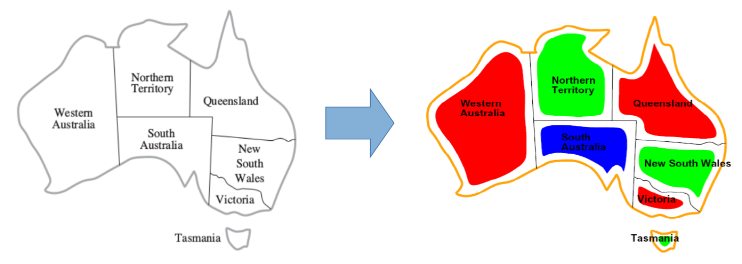
$$dom_A = \{1, 2, 3, 4\}, \ dom_B = \{1, 2, 3, 4\}$$
$$dom_C = \{1, 2, 3, 4\}, \ dom_D = \{1, 2, 3, 4\}$$
$$dom_E = \{1, 2, 3, 4\}$$

- **Constraints: represent illegal conflicts between variables**

$$(B \neq 3) \wedge (C \neq 2) \wedge (A \neq B) \wedge (B \neq C) \wedge$$
$$(C < D) \wedge (A = D) \wedge (E < A) \wedge (E < B) \wedge$$
$$(E < C) \wedge (E < D) \wedge (B \neq D).$$

Example: Graph colouring

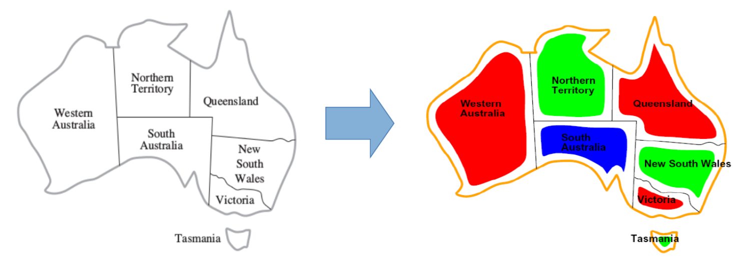
Problem: assign each state (and territory) a colour such that no two adjacent states have the same colour.



- **Variables:** ?
- **Domains:** ?
- **Constraints:** ?

Example: Graph colouring

Problem: assign each state (and territory) a colour such that no two adjacent states have the same colour.



- **Variables:** $X = \{\text{NSW}, \text{VIC}, \text{QLD}, \text{WA}, \text{SA}, \text{TAS}, \text{NT}\}$
- **Domains:** $dom_x = \{r, g, b\}$ for each $x \in X$.
- **Constraints:** $(\text{WA} \neq \text{SA}) \wedge (\text{WA} \neq \text{NT}) \wedge (\text{NT} \neq \text{QLD}) \wedge \ldots$

Example: Soduko

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

Variables:

Domains:

Constraints:

Generate-and-Test Algorithm

- Generate the assignment space $dom = dom_{V_1} \times dom_{V_2} \times \ldots \times dom_{V_n}$ (i.e. Cartesian product).
- Test each assignment with the constraints.
- How many assignments need to be tested for n variables each with domain size d?
- Example:

How many leaf nodes are expanded in the worst case? $3^7 = 2187$.

Naively apply DFS to a CSP

- States defined by the values assigned so far (partial assignments)
- Initial state: the empty assignment, {}
- Successor function: assign a value to an unassigned variable
- Goal test: the current assignment is complete and satisfies all constraints

What can go wrong?

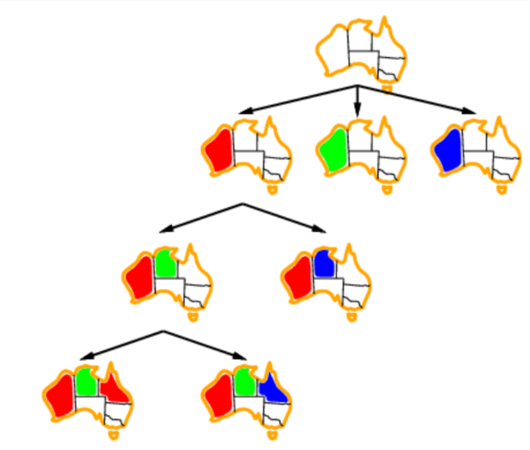
Backtracking algorithms

Backtracking Algorithms

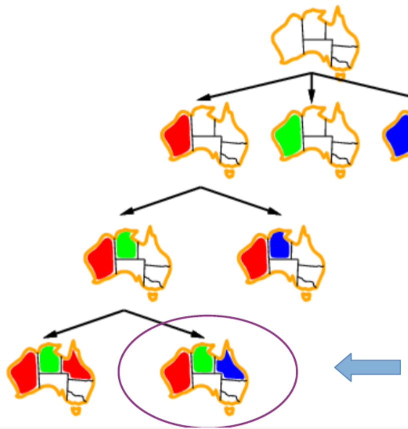
- Systematically explore *dom* by instantiating the variables one at a time
- evaluate each constraint predicate as soon as all its variables are bound
- any partial assignment that doesn't satisfy the constraint can be pruned.

Scheduling example: Assignment $A = 1 \wedge B = 1$ is inconsistent with constraint $A \neq B$ regardless of the value of the other variables.

Backtracking Algorithms: Graph colouring example



Backtracking Algorithms: Graph colouring example



Does this state have any valid successors?

CSP as Graph Searching

A CSP can be solved by graph-searching with variable-ordering and fail-on-violation

- A node is an assignment values to some of the variables.
- Suppose node N is the assignment $X_1 = v_1, \ldots, X_k = v_k$. Select a variable Y that isn't assigned in N (using your favourite graph search algorithm).
  For each value $y_i \in dom(Y)$
  $X_1 = v_1, \ldots, X_k = v_k, Y = y_i$ is a neighbour if it is consistent with the constraints.
- The start node is the empty assignment.
- A goal node is a total assignment that satisfies the constraints.

Recursive implementation of Backtracking Search

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment given CONSTRAINTS[csp] then
            add {var = value} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

Consistency algorithms

Consistency Algorithms: Prune the search space

Idea: prune the domains as much as possible before selecting values from them.

A variable is domain consistent (or 1-consistent) if no value of the domain of the node is ruled impossible by any of the constraints.

Example: Is the scheduling example domain consistent?
Variables: $X = \{A, B, C, D, E\}$ that represent the starting times of various activities. Domains: $dom_1 = \{1, 2, 3, 4\}, \forall i \in X$ Constraints: $(B \neq 3) \wedge (C \neq 2) \wedge (A \neq B) \wedge (B \neq C) \wedge (C < D) \wedge (A = D) \wedge (E < A) \wedge (E < B) \wedge (E < C) \wedge (E < D) \wedge (B \neq D)$.

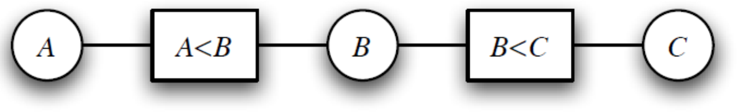
No, $dom_B = \{1, 2, 3, 4\}$ isn't domain consistent as $B = 3$ violates the constraint $B \neq 3$.

Even better, we can propagate this information to other unassigned variables.

Constraint Network: a bipartite graph

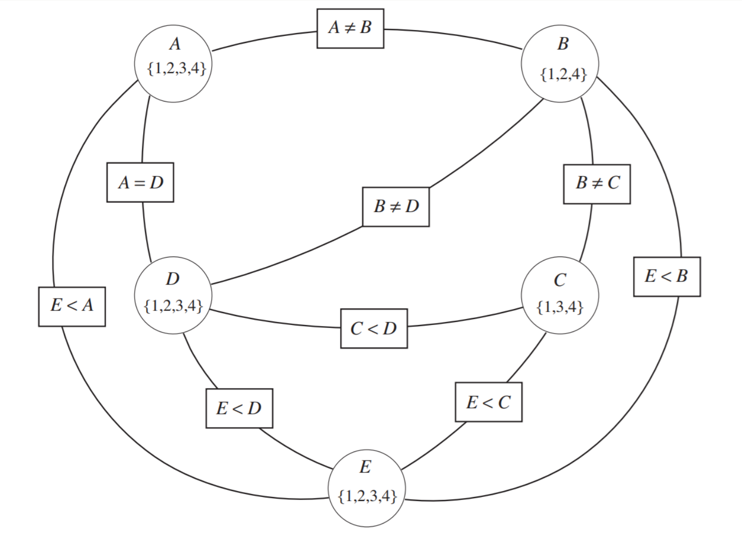
- There is a circle or oval-shaped node for each variable.
- There is a rectangular node for each constraint.
- There is a domain of values associated with each variable node.
- There is an arc from variable X to each constraint that involves X.

Example *binary* constraint network:



Variables: $\{A, B, C\}$, Domains: (not specified)
Constraints: $r_1(A, B) = (A < B)$, $r_2(B, C) = (B < C)$
Arcs: $\langle A, r_1(A, B)\rangle$, $\langle B, r_1(A, B)\rangle, \ldots$

Constraint Network: Scheduling example



Arc Consistency

Arc consistency is a form of constraint propagation that repeatedly enforces constraints locally.

Arc consistency

An arc $\langle X, r(X, \overline{Y})\rangle$ is **arc consistent** if, for each value $x \in dom(X)$, there is some value $\overline{y} \in dom(\overline{Y})$ such that $r(x, \overline{y})$ is satisfied. A network is arc consistent if all its arcs are arc consistent.

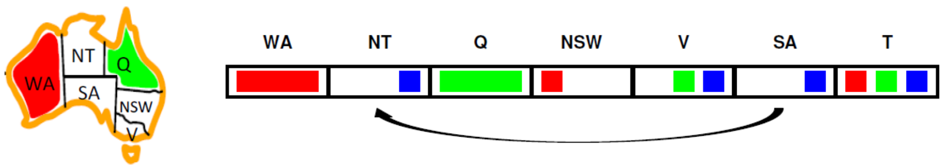
- What if arc $\langle X, r(X, \overline{Y})\rangle$ is *not* arc consistent?
- All values of X in $dom(X)$ for which there is no corresponding value in $dom(\overline{Y})$ can be deleted from $dom(X)$ to make the arc $\langle X, r(X, \overline{Y})\rangle$ consistent.
- This removal can never rule out any models (do you see why?)

Arc Consistency Algorithm

- The arcs can be considered in turn making each arc consistent.
- When an arc has been made arc consistent, does it ever need to be checked again? An arc $\langle X, r(X, \overline{Y}) \rangle$ needs to be revisited if the domain of one of the Y's is reduced.
- Regardless of the order in which arcs are considered, we will terminate with the same result: an arc consistent network.
- Three possible outcomes when all arcs are made arc consistent: (Is there a solution?)
  - One domain is empty $\Rightarrow$ no solution
  - Each domain has a single value $\Rightarrow$ unique solution
  - Some domains have more than one value $\Rightarrow$ there may or may not be a solution
    Need to solve this new (usually simpler) CSP: same constraints, domains have been reduced

Arc consistency: Graph colouring example

$\langle X, r(X, \overline{Y}) \rangle$ is consistent iff for every value x of X there is some allowed y



Algorithm:

1. Delete values from tail in order to make each arc consistent
2. If X loses a value, neighbours of X need to be rechecked!
3. Arc consistency detects failure earlier than forward checking
4. Can be run as a preprocessor or after each assignment (e.g. within backtracking search)

Complexity of the arc consistency algorithm

Worst-case complexity of this procedure:

- let the max size of a variable domain be d
- let the number of constraints be e
- complexity is $O(ed^3)$

Some special cases are faster:

- e.g. if the constraint graph is a tree, arc consistency is $O(ed)$

Finding solutions when AC finishes

If some variable domains have more than one element $\Rightarrow$ search

- Advanced alternatives (beyond this course):
- Variable and arc ordering heuristics speed up arc consistency and search.
- Split a domain, then recursively solve each part.
- Use *conditioning* (fix a variable and prune its neighbours' domains) or *cutset conditioning* (instantiate, in all ways, a set of variables such that the remaining constraint graph is a tree).
- Many other heuristics for exploiting graph structure.

Final note on hard and soft constraints

- Given a set of variables, assign a value to each variable that either
  - satisfies some set of constraints: **satisfiability problems** — "hard constraints"
  - minimises some cost function, where each assignment of values to variables has some cost: **optimisation problems** — "soft constraints"
- Many problems are a mix of hard and soft constraints (called constrained optimisation problems).

Attributions and References

Thanks to Dr Alina Bialkowski and Dr Hanna Kurniawati for their materials.

Many of the slides in this course are adapted from David Poole and Alan Mackworth, *Artificial Intelligence: foundations of computational agents*, 2E, CUP, 2017 `http://https://artint.info/`. These materials are copyright © Poole and Mackworth, 2017, licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Other materials derived from Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, 3E, Prentice Hall, 2009.

Some materials are derived from Chris Amato (Northeastern University).

All remaining errors are Archie's — please email if you find any: archie.chapman@uq.edu.au