

CPSC 322: Introduction to Artificial Intelligence

CSPs: Search and Consistency

Textbook reference: [[4.2](#),[4.3](#),[4.4](#)]


Instructor: Varada Kolhatkar
University of British Columbia

Credit: These slides are adapted from the slides of the previous offerings of the course. Thanks to all instructors for creating and improving the teaching material and making it available!

Announcements

- Midterm time and location
Time: Friday, Oct 25th, from 6pm to 7pm
Location: Woodward 2
(Instructional Resources Centre-IRC) (WOOD) - 2
- Midterm practice questions are available on Piazza.
- My office hours: Fridays from 11am to noon at ICCS 185

Lecture outline

- Recap CSPs (~5 mins) 
- Solving CSPs
 - Generate and Test (~10 mins)
 - Search (~30 mins)
 - Constraint network (~10 mins)
- Class activity (~15 mins)
- Summary and wrap up

Today: Learning outcomes

From this lecture, students are expected to be able to:

- Implement the **Generate-and-Test** Algorithm. Explain its disadvantages.
- Solve a **CSP by search** (specify neighbours, states, start state, goal state). Compare strategies for CSP search. Implement pruning for DFS search in a CSP.
- Build a **constraint network** for a set of constraints.

A rough CPSC 322 overview

Representation
and reasoning

We'll now
focus on

Environment

Problem

Deterministic

Stochastic

Constraint
satisfaction

Arc consistency

Variables +
constraints

Search

Static

Logics

Search

Belief networks

Variable elimination

Query

Decision networks

Variable elimination

Sequential

Planning

STRIPS

Search

Markov decision
processes

Value iteration

CSP: Motivation



Hardware verification



Software verification

“Using our constraint solver, IBM's hardware verification tools have reduced design and verification costs by more than \$100M. External companies that licensed the test generator tools noted the powerful CSP engine capabilities as a key factor in their decision.”

https://www.research.ibm.com/haifa/dept/vst/csp_verification.shtml

Constraint satisfaction problems (CSPs)

A **constraint satisfaction problem** (CSP) consists of

- a set of **variables** V
- a **domain** $dom(V_i)$ for each variable $V_i \in V$
- a set of **constraints** C

Simple example:

$$V = \{V_1\}$$

$$dom(V_1) = \{2, 4, 8, 16\}$$

$$C = \{c_1, c_2\}$$

$$c_1 : V_1 \neq 8$$

$$c_2 : V_1 > 2$$

Possible worlds and models

A possible world of a CSP is an assignment of values to **all** of its variables.

Simple example:

$$V = \{V_1\}$$

$$\text{dom}(V_1) = \{2, 4, 8, 16\}$$

$$C = \{c_1, c_2\}$$

$$c_1 : V_1 \neq 8$$

$$c_2 : V_1 > 2$$

Possible worlds

$$\{V_1 = 2\}$$

$$\{V_1 = 4\}$$

$$\{V_1 = 8\}$$

$$\{V_1 = 16\}$$

Possible worlds and models

A possible world of a CSP is an assignment of values to **all** of its variables.

A model/solution of a CSP is an assignment of values to all of its variables that **satisfies** all of its constraints.

Simple example:

$$V = \{V_1\}$$

$$\text{dom}(V_1) = \{2, 4, 8, 16\}$$

$$C = \{c_1, c_2\}$$

$$c_1 : V_1 \neq 8$$

$$c_2 : V_1 > 2$$

Possible worlds and models

$$\{V_1 = 2\}$$

$$\{V_1 = 4\} \text{ (a model)}$$

$$\{V_1 = 8\}$$

$$\{V_1 = 16\} \text{ (a model)}$$


Finite Constraint Satisfaction Problem

A **finite constraint satisfaction problem** (FCSP) consists of

- a **finite** set of **variables** V
- a **finite domain** $dom(V_i)$ for each variable $V_i \in V$
- a set of **constraints** C

In this course when we say CSPs, we actually mean FCSPs.

Lecture outline

- Recap CSPs (~5 mins)
- Solving CSPs 
 - Generate and Test (~10 mins)
 - Search (~30 mins)
 - Constraint network (~10 mins)
- Class activity (~15 mins)
- Summary and wrap up

Solving CSPs

In this lecture, we will look at two methods to solve CSPs

- Generate-and-Test
- Graph search

Generate-and-Test

Systematically check all possible worlds

Possible worlds: cross product of domains

$$\text{dom}(V_1) \times \text{dom}(V_2) \times \dots \times \text{dom}(V_n)$$

Example: Suppose you have three variables, say A, B, and C.

$\text{dom}(A) = \{1,2,3,4,5\}$, $\text{dom}(B) = \{2,4\}$, $\text{dom}(C) = \{1,2,3,4,5\}$

```
for a in dom(A)
```

```
    for b in dom(B)
```

```
        for c in dom(C)
```

```
            if (abc) satisfies all the constraints
```

```
                return abc
```

```
return NULL
```

Generate-and-Test

Generate possible worlds one at a time and test them to see if they violate any constraints

Example: Suppose you have three variables, say A, B, and C.
 $\text{dom}(A) = \{1,2,3,4,5\}$, $\text{dom}(B) = \{2,4\}$, $\text{dom}(C) = \{1,2,3,4,5\}$

```
for a in dom(A)
  for b in dom(B)
    for c in dom(C)
      if (abc) satisfies all the constraints
        return abc
return NULL
```

Would this method be able to solve any given CSP?
What's wrong with this approach?

Generate-and-Test complexity



If there are k variables, each with domain size d , and there are c constraints, the complexity of generate-and-test is

A. $O(ckd)$

B. $O(ck^d)$

C. $O(cd^k)$


D. $O(d^{ck})$

Generate-and-Test algorithm

If there are k variables, each with domain size d , and there are c constraints, the complexity of generate-and-test is

A. $O(ckd)$


B. $O(ck^d)$

C. $O(cd^k)$ 

D. $O(d^{ck})$

There are d^k possible worlds.
For each one need to check c constraints.

Lecture outline

- Recap CSPs (~5 mins)
- Solving CSPs
 - Generate and Test (~10 mins)
 - Search (~30 mins) 
 - Constraint network (~10 mins)
- Class activity (~15 mins)
- Summary and wrap up

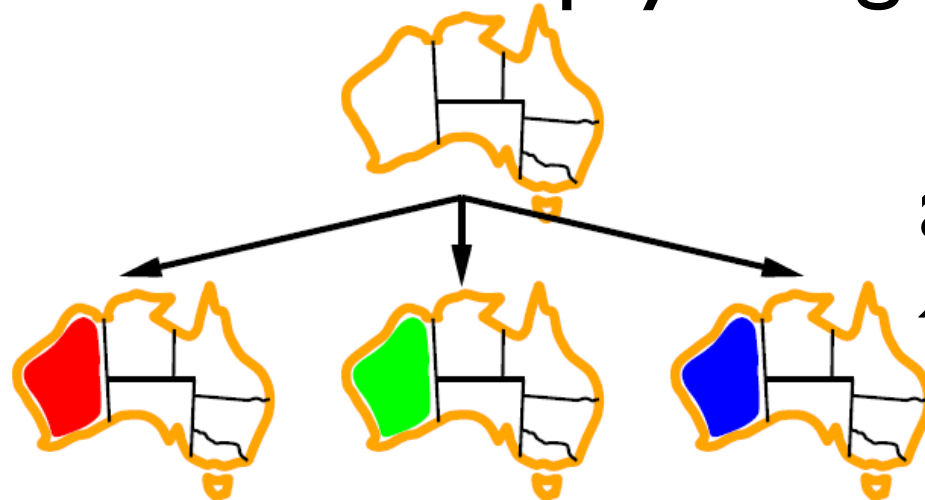
Solving CSPs using graph search

- **States:** assignments of values to a subset of the variables (partial assignments)
- **Start state:** the empty assignment (no variables assigned values)
- **Neighbours of a state:** nodes in which values are assigned to one additional variable
- **Goal state:** a state which assigns a value to each variable, and satisfies all of the constraints

The path to the goal node is not important.

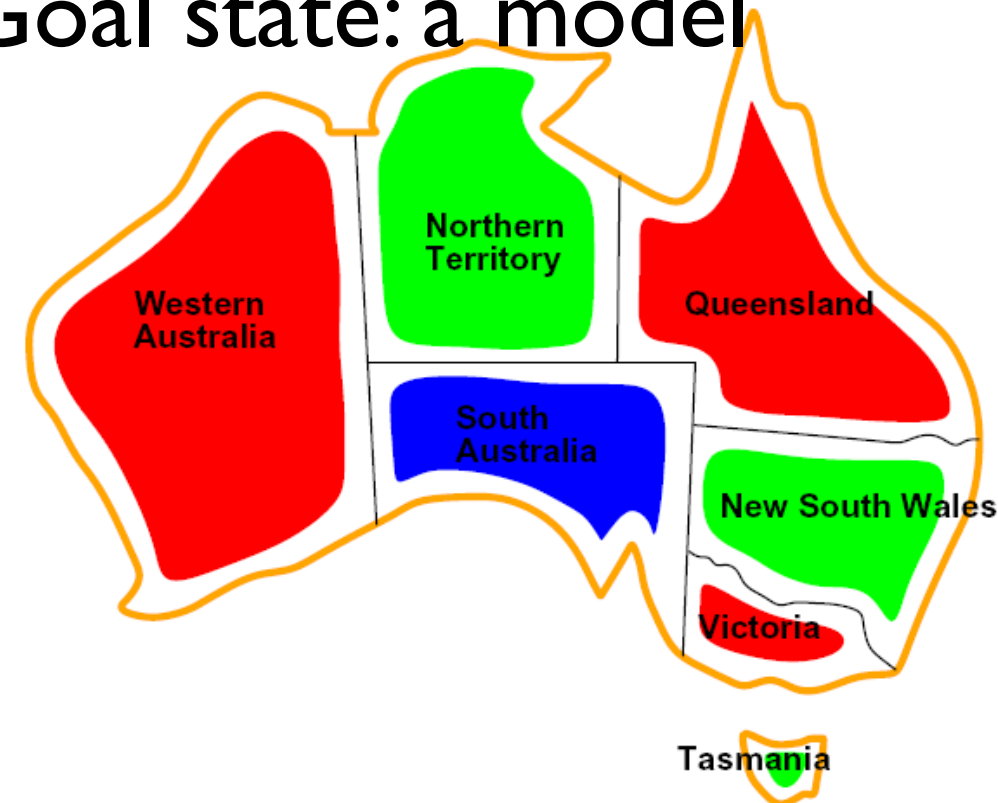
Solving CSPs using graph search

Start state: empty assignment



a neighbour: a value is assigned to an additional variable

Goal state: a model




States have partial assignments

Search strategy for a CSP



If there are n variables in a CSP, every solution is at depth n . Is there a role for heuristic function?

A. Yes


B. No 

C. It depends

Search strategy for a CSP



The search space in CSPs is always

- A. Finite with cycles
- B. Infinite without cycles
- C. Finite without cycles 
- D. Infinite with cycles

Search strategy for a CSP



So which search strategy is better for CSPs?

A. BFS

B. A*

C. IDS

D. DFS 

Solving CSPs using graph search: Backtracking

Explore search space via DFS but evaluate each constraint as soon as all its variables are bound.

Any partial assignment that doesn't satisfy the constraint can be pruned.

Example:

3 variables A, B, C each with domain {1,2,3,4}

{A = 1, B = 1} is inconsistent with constraint $A \neq B$

regardless of the value of the other variables

⇒ Fail! Prune!

Solving CSPs using graph search: Formulations

How do you find neighbours?

Two formulations of using graph searching to solve CSPs based on the successor function

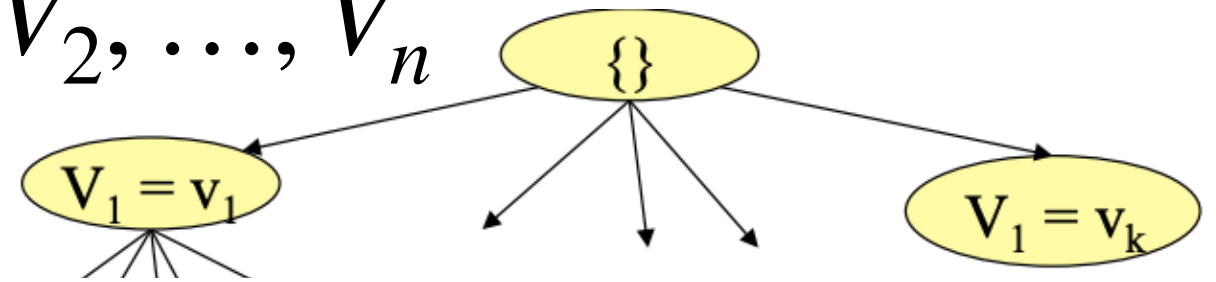
1. Decide an ordering of the variables beforehand V_1, V_2, \dots, V_n and follow the total order during search
2. Assign each previously unassigned variable not necessarily in a particular order.

Solving CSPs using graph search: Formulation I

Follow the total ordering V_1, V_2, \dots, V_n 
during search

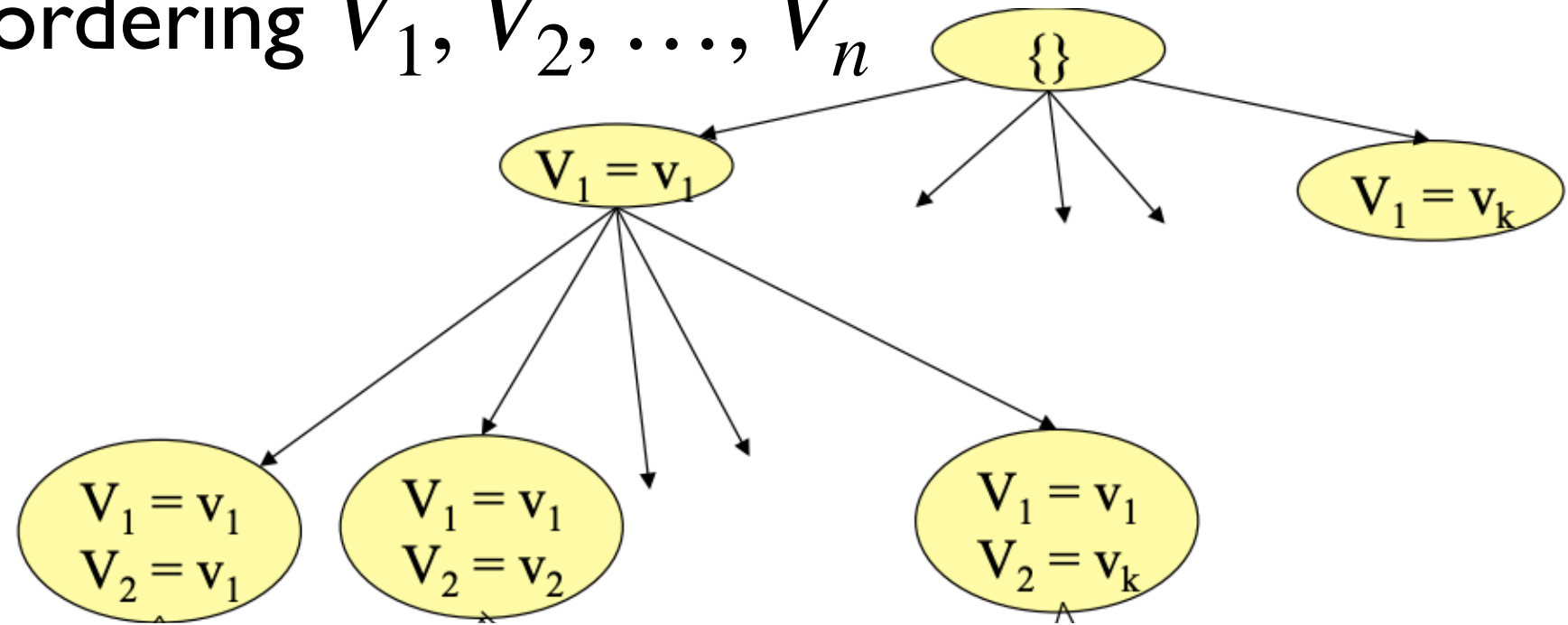
Solving CSPs using graph search: Formulation I

Follow the total ordering V_1, V_2, \dots, V_n during search



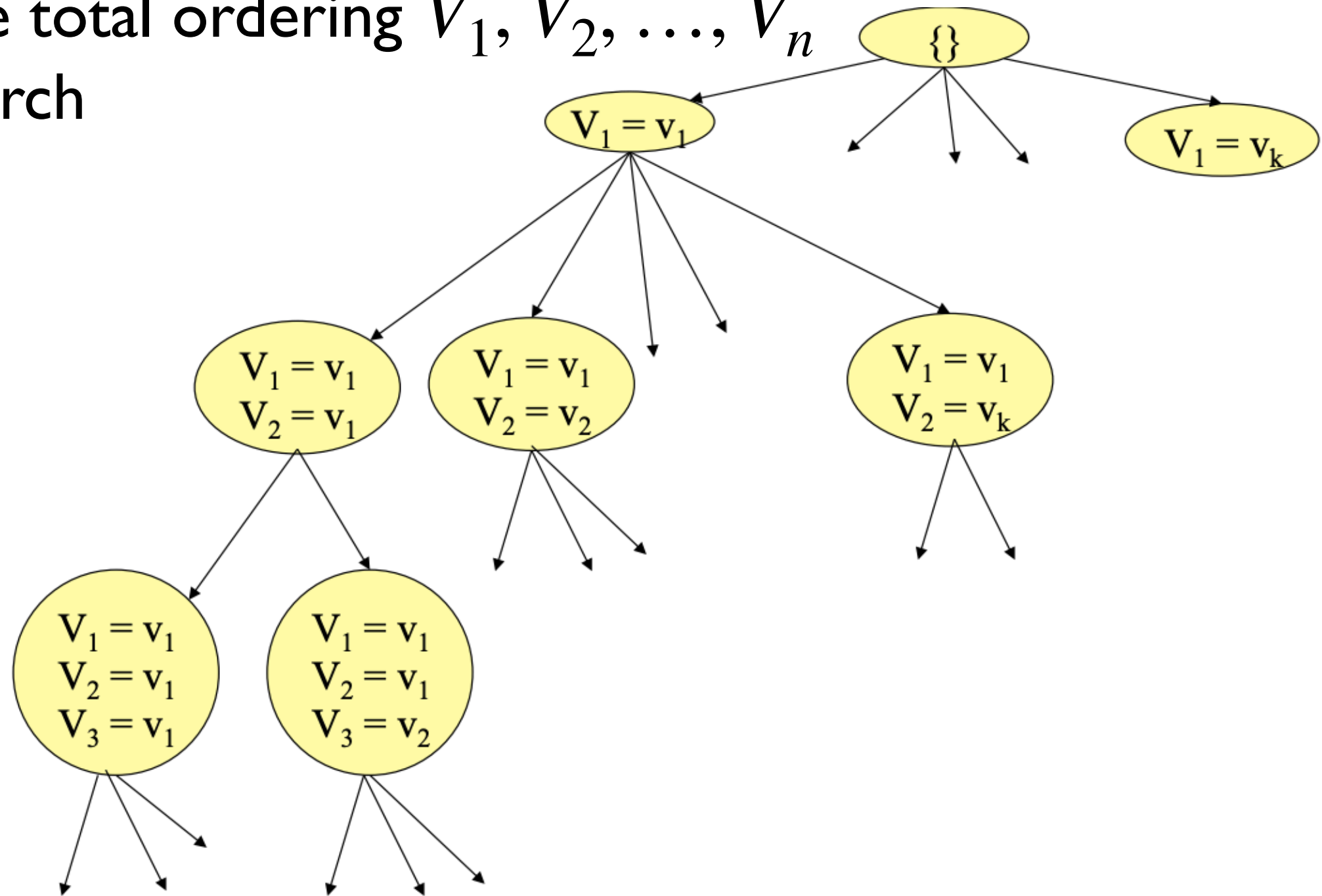
Solving CSPs using graph search: Formulation I

Follow the total ordering V_1, V_2, \dots, V_n during search



Solving CSPs using graph search: Formulation I

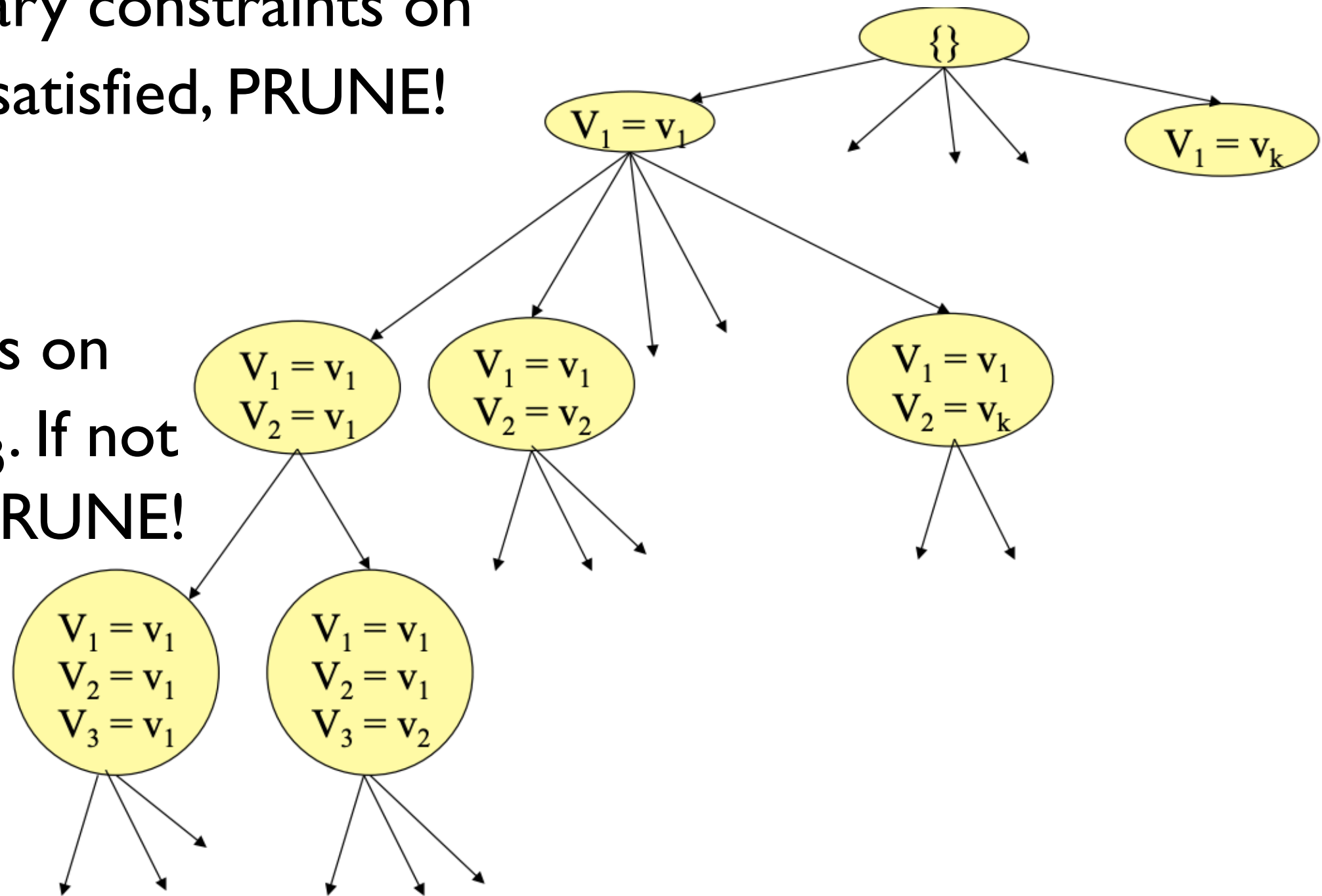
Follow the total ordering V_1, V_2, \dots, V_n during search



Solving CSPs using graph search: Formulation I

Check unary constraints on V_1 . If not satisfied, PRUNE!

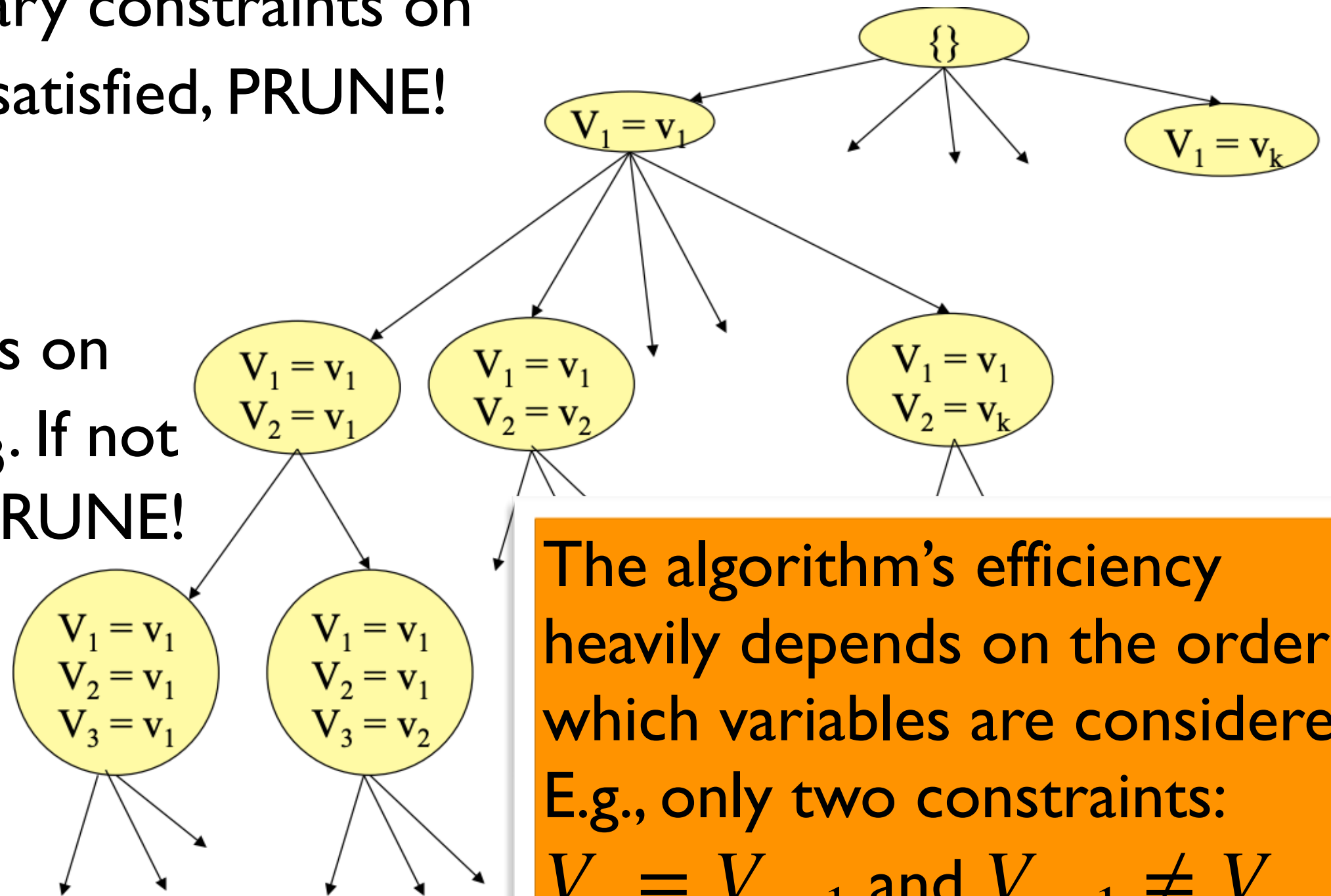
Check constraints on V_1, V_2, V_3 . If not satisfied, PRUNE!



Solving CSPs using graph search: Formulation I

Check unary constraints on V_1 . If not satisfied, PRUNE!

Check constraints on V_1, V_2, V_3 . If not satisfied, PRUNE!



The algorithm's efficiency heavily depends on the order in which variables are considered. E.g., only two constraints:
 $V_n = V_{n-1}$ and $V_{n-1} \neq V_n$

Solving CSPs using graph search: Formulation 2

- A state assigns values to some subset of variables
- Successor function assign each previously unassigned variable (not necessarily in a particular order.)

Example:

current node n : $\{ V_4 = v_1, V_7 = v_1, V_{30} = v_1 \}$

Neighbours of n :

$\{ V_4 = v_1, V_7 = v_1, V_{30} = v_1, V_x = y \}, V_x \notin \{ V_4, V_7, V_{30} \}, y \in \text{dom}(V_x)$

Selecting variables in a smart way

Relies on one or more **heuristics** to select which variables to consider next.

- E.g., variable involved in the largest number of constraints:
“If you are going to fail on this branch, fail early!”
- Can also be smart about which values to consider first

Note that we are using the word “heuristics” in a different sense here. It doesn’t mean estimate of the distance to the goal in this context.

Solving CSPs using graph search: Formulation 2

Toy problem

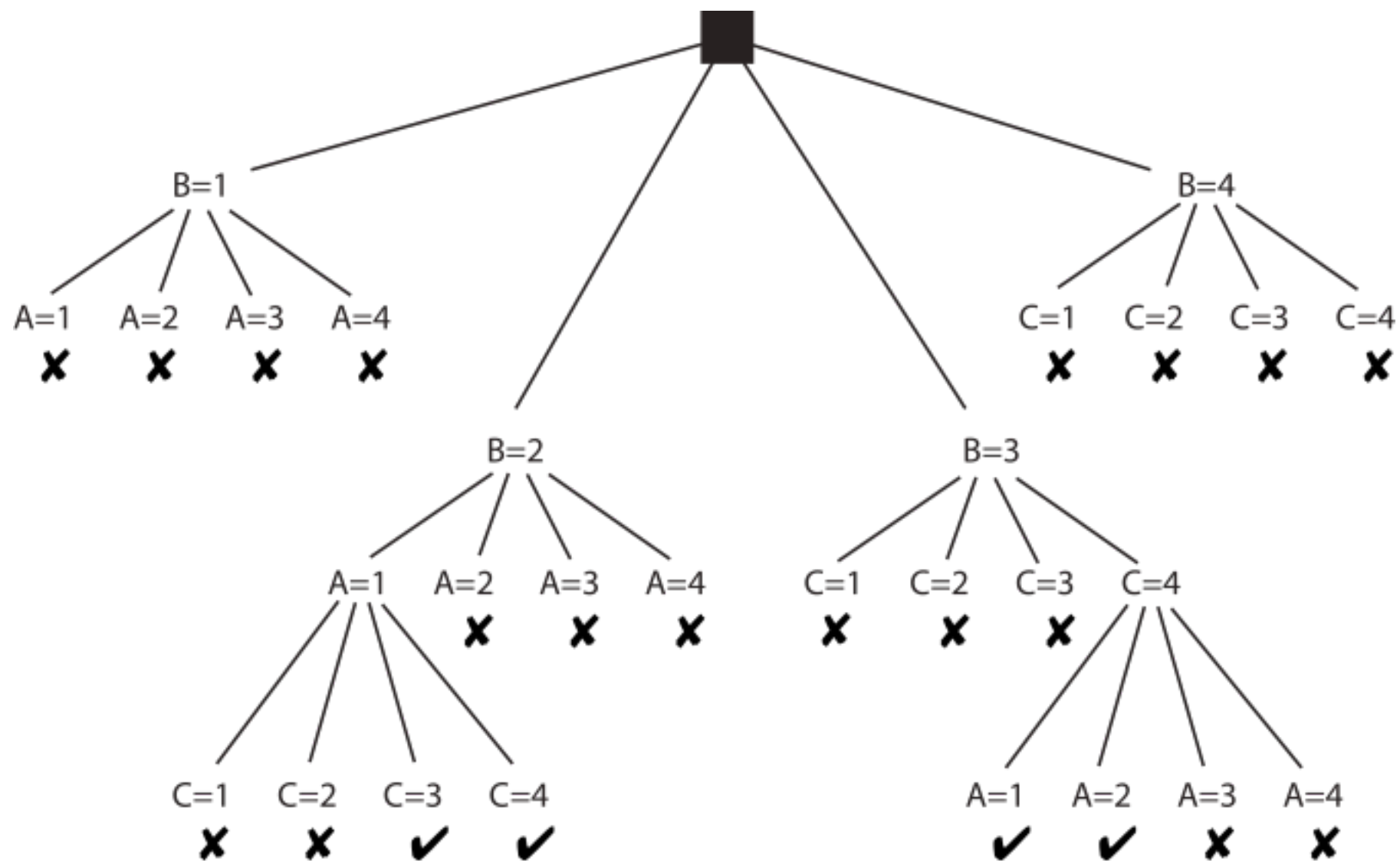
Variables: A, B, C, Domains: {1,2,3,4}, Constraints: $A < B$, $B < C$



Solving CSPs using graph search: Formulation 2

Toy problem

Variables: A, B, C, Domains: {1,2,3,4}, Constraints: $A < B$, $B < C$



Standard search vs. CSPs as graph search

State: assignment of values to a subset of variables

Successor function: assign values to a free variable

Goal test: all variables assigned a value and all constraints satisfied

Solution: a possible world that satisfies the constraints: a model

Heuristic function: None (all solutions are at a same distance from the start)

CSP solvers vs. state-space searchers

CSP solvers

- They can identify whether a partial assignment is not a solution
- They are faster because they can quickly eliminate large portions of the search space
- Only the goal is important and not the path to the goal

State-space searchers


- They can only identify whether a specific state is goal or not
- They are slower
- The path to the goal is important

Can we do better than search?

Key idea: **Prune the domains** as much as possible **before searching** for a solution

Simple when using constraints involving single variables

Lecture outline

- Recap CSPs (~5 mins)
- Solving CSPs
 - Generate and Test (~10 mins)
 - Search (~30 mins)
 - Constraint network (~10 mins) 
- Class activity (~15 mins)
- Summary and wrap up

Can we do better than search?

Definition: A variable is **domain consistent** if no value of its domain is ruled impossible by any unary constraint.

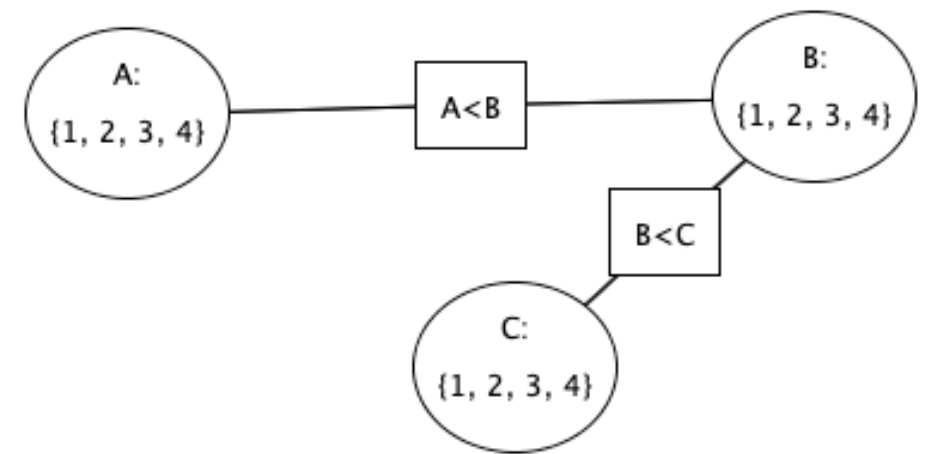
Example:

If we have the unary constraint $B \neq 3$, then $D_B = \{1, 2, 3, 4\}$ is NOT domain consistent.

Constraints involving multiple variables

Definition: A **constraint network** is defined by a graph, with

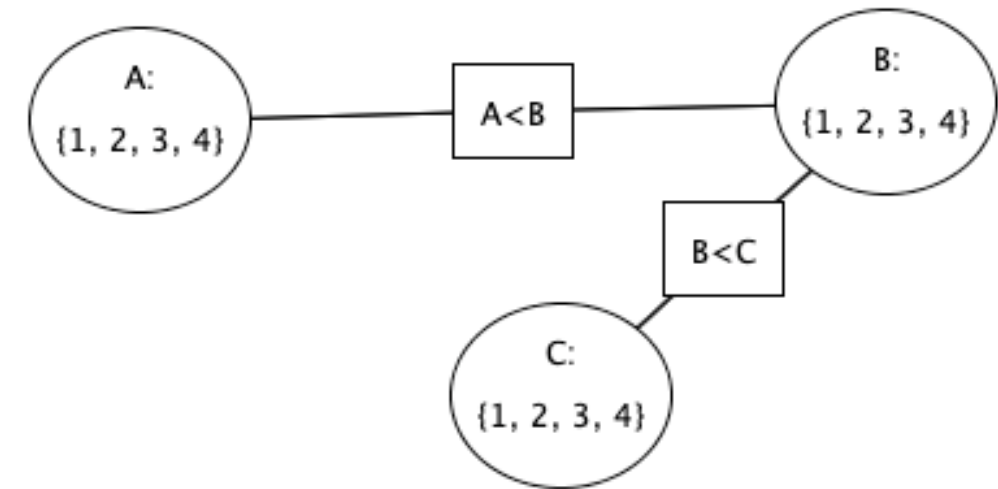
- one node for every variable (ovals)
- one node for every constraint (rectangles)
- undirected edges running between variable nodes and constraint nodes whenever a given variable is involved in a given constraint.



Constraints involving multiple variables

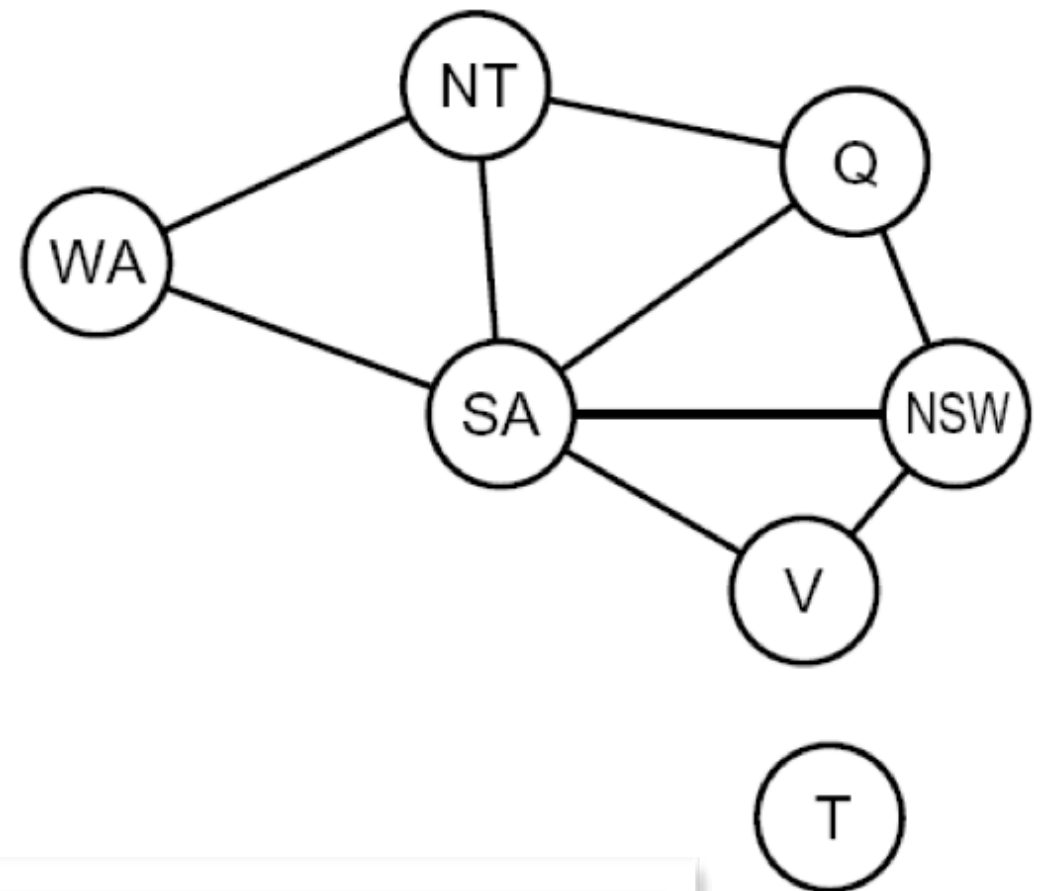
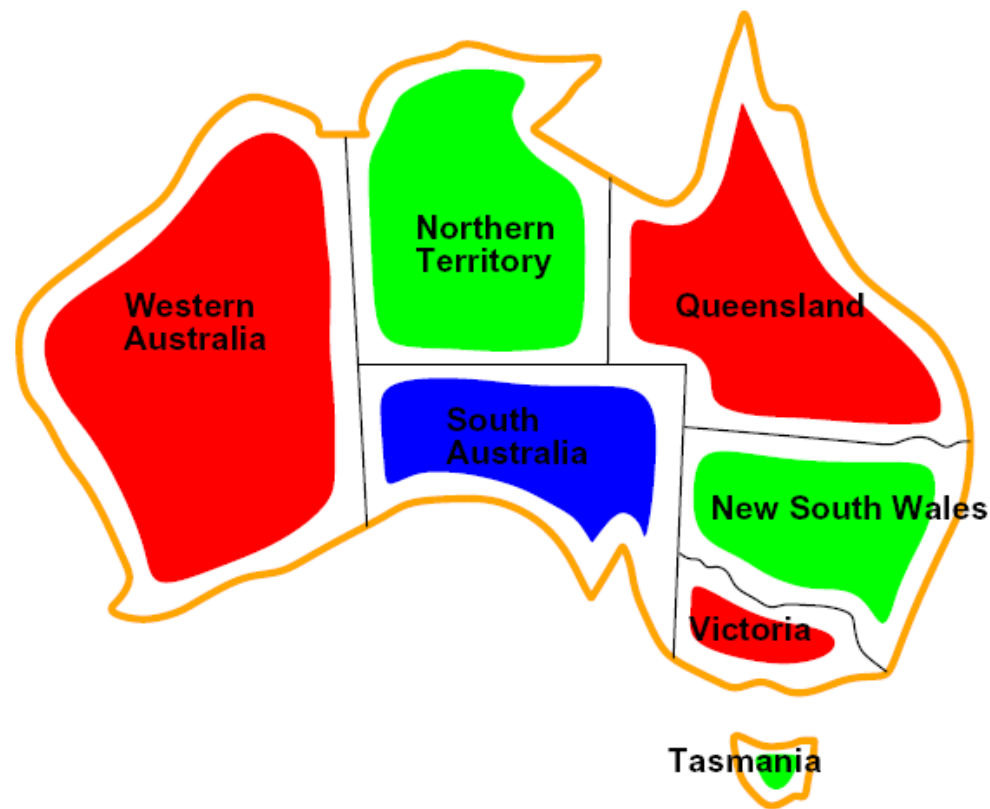
Definition: A **constraint network** is defined by a graph, with

- one node for every variable (ovals)
- one node for every constraint (rectangles)
- undirected edges running between variable nodes and constraint nodes whenever a given variable is involved in a given constraint



Note: strictly speaking, when all of the constraints are binary, constraint nodes are not necessary: we can drop constraint nodes and use edges to indicate that a constraint holds between a pair of variables. However, in this course we will always show constraint nodes.

Constraint network for map colouring



The constraint nodes are not shown because all the constraints are binary

Class activity (~15 mins)

Companies	Times they can make	Constraints
Google (G)	{1}	$G \neq F, G \neq I, G \neq M, G \neq O$ $F \neq I, F \neq M, F \neq O$ $I \neq O$ $O \neq A$
Facebook (F)	{1, 2, 3, 4}	
IBM (I)	{1, 2, 3, 4}	
Microsoft (M)	{1, 2, 3, 4}	
OpenAI (O)	{1,2,3}	
Apple (A)	{2}	

- Draw a constraint network for this problem
- Errata: left-most branch = **assign values to at least 4 variables**

Revisit: Learning outcomes

From this lecture, students are expected to be able to:

- Implement the **Generate-and-Test** Algorithm. Explain its disadvantages.
- Solve a **CSP by search** (specify neighbours, states, start state, goal state). Compare strategies for CSP search. Implement pruning for DFS search in a CSP.
- Build a **constraint network** for a set of constraints.

Coming up

Readings for next class

- 4.4 Consistency Algorithms
- 4.5 Domain Splitting

