SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Constraint Satisfaction Problems I

PROF LIM KWAN HUI

50.021 Artificial Intelligence

The following notes are compiled from various sources such as textbooks, lecture materials, Web resources and are shared for academic purposes only, intended for use by students registered for a specific course. In the interest of brevity, every source is not cited. The compiler of these notes gratefully acknowledges all such sources.

Outline & Objectives

- Understand the differences between standard search problems and constraint satisfaction problems
- Able to formulate a constraint satisfaction problem
- Understand the workings behind backtracking search and the various heuristics used to enhance its efficiency
- Able to use backtracking search to solve a CSP

Recap: Standard Search Problem Formulation

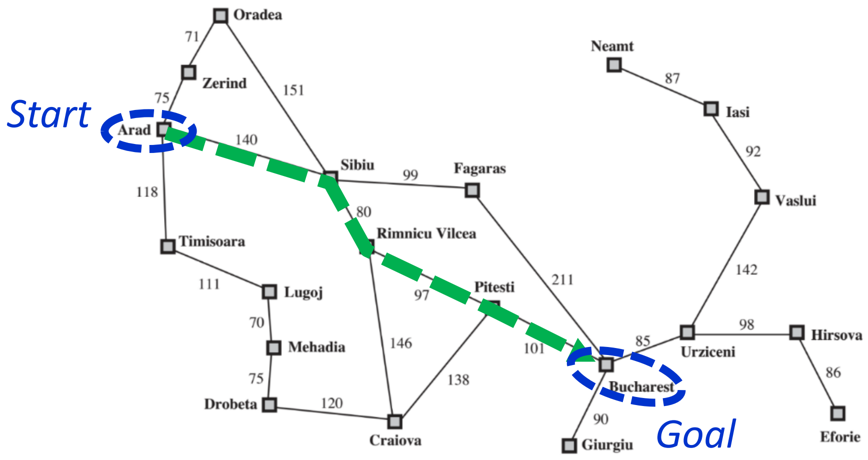
- **State space**, e.g. *At(Arad), At(Bucharest)*
- **Initial state**, e.g. *At(Arad)*
- **Actions**, set of actions given a specific state
  - **Transition model** e.g., *Result(At(Arad),Go(Zerind))* → *At(Zerind)*
  - **Path cost** (additive), e.g., sum of distances, number of actions, etc
- **Goal test**, can be
  - Explicit, e.g. *At(Bucharest)*
  - Implicit, e.g. *checkmate(x)*

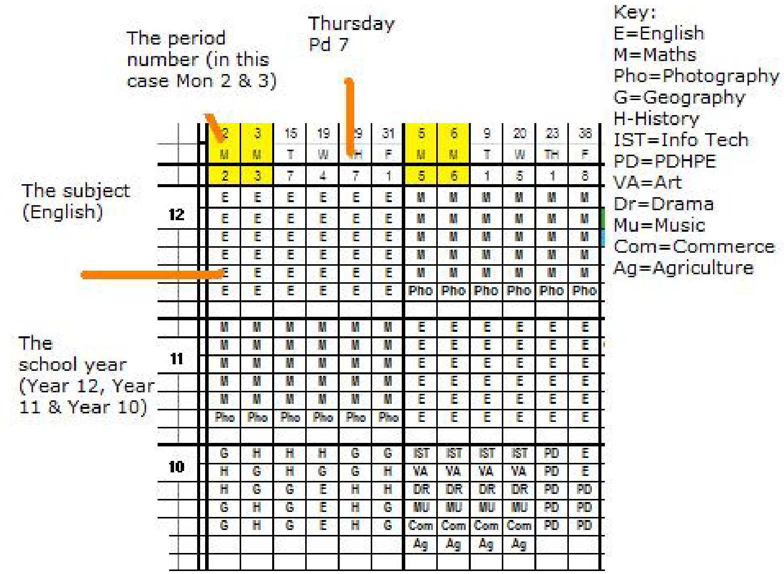
Recap: Standard Search Problem Solution

- A solution is a sequence of actions from the initial state to a goal state
  - E.g., Arad → Sibiu → Fagarus → Bucharest
- An optimal solution is a solution with the lowest path cost

Compare and Contrast

- What are the differences among these problems?





Initial State

8		6
5	4	7
2	3	1

Goal State

	1	2
3	4	5
6	7	8

Initial State

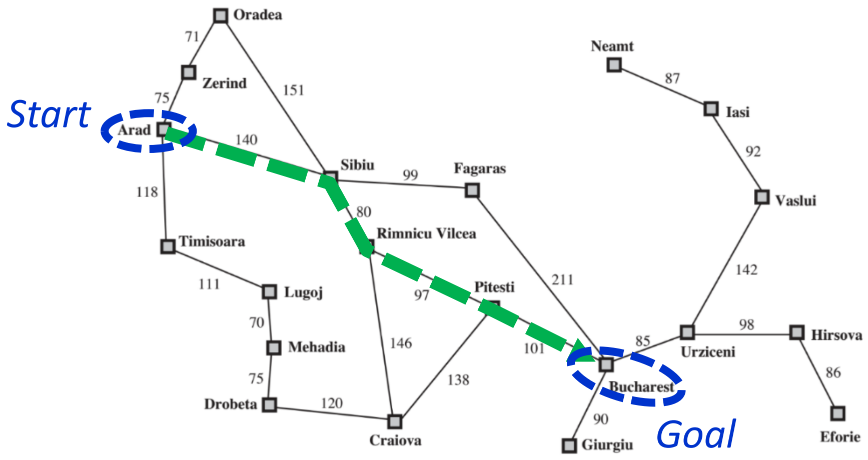
	3					9		
		6						
			2	4	1		3	
			9			7		
					2			4
	8			7			2	
8	5							
	9		7		4			
					6			1

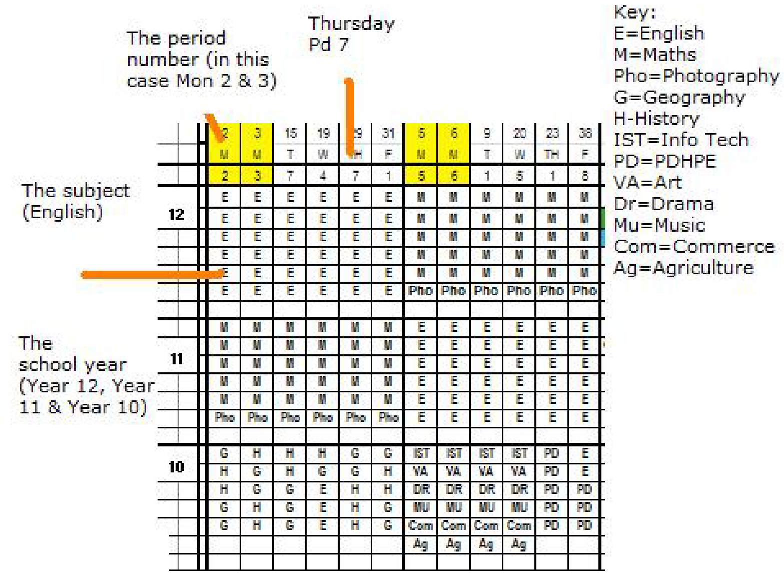
Goal State

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

Compare and Contrast

- What are the differences among these problems? Two main types





Initial State

8		6
5	4	7
2	3	1

Goal State

	1	2
3	4	5
6	7	8

Initial State

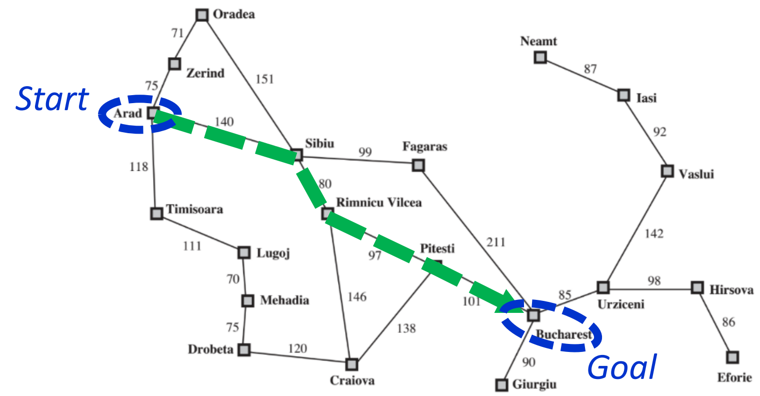
	3					9		
		6						
			2	4	1		3	
			9			7		
					2			4
	8			7			2	
8	5							
	9		7		4			
					6			1

Goal State

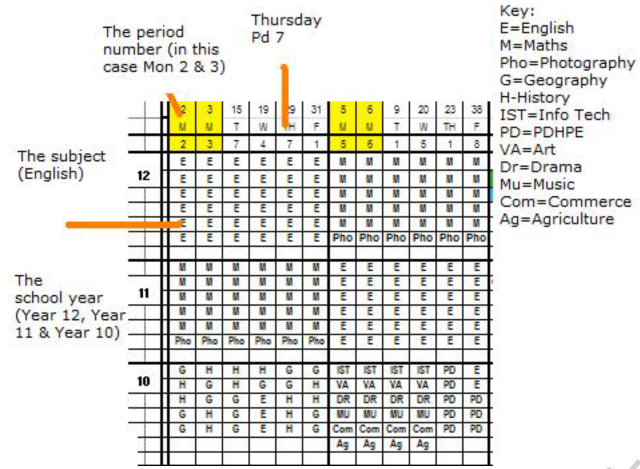
1	3	2	5	6	7	9	4	8
5	4	6	3	8	9	2	1	7
9	7	8	2	4	1	6	3	5
2	6	4	9	1	8	7	5	3
7	1	5	6	3	2	8	9	4
3	8	9	4	7	5	1	2	6
8	5	7	1	2	3	4	6	9
6	9	1	7	5	4	3	8	2
4	2	3	8	9	6	5	7	1

What is the Solution?

- Standard Search Problems
  - More interested in the sequence of actions (path) to the goal
  - Paths have various costs, depths
  - Heuristics give problem-specific guidance



- Constraint Satisfaction Problems
  - More interested in the goal itself, not the sequence of actions (path) there
  - All paths at the same depth (for some formulations)
  - CSPs are specialized for this type of task



Recap: Standard Search Problem Formulation

- **State space**, e.g. *At(Arad), At(Bucharest)*
- **Initial state**, e.g. *At(Arad)*
- **Actions**, set of actions given a specific state
  - **Transition model** e.g., *Result(At(Arad),Go(Zerind))* → *At(Zerind)*
  - **Path cost** (additive), e.g., sum of distances, number of actions, etc
- **Goal test**, can be
  - Explicit, e.g. *At(Bucharest)*
  - Implicit, e.g. *checkmate(x)*

Constraint Satisfaction Problems

- **State**
  - Defined by ***variables X_i*** that take on values from ***domain D_i***
- **Goal Test**
  - A set of ***constraints C_i*** specifying ***allowable combinations of values*** for subsets of variables
- In contrast to standard search problems
  - State is a "black box" - any old data structure that supports goal test, evaluation, successor

Constraint Satisfaction Problems

- **State**
  - Defined by ***variables X_i*** that take on values from ***domain D_i***
- **Goal Test**
  - A set of ***constraints C_i*** specifying ***allowable combinations of values*** for subsets of variables
- Simple example of a *formal representation language*
- Allows useful *general-purpose algorithms* with more power than standard search algorithms

Constraint Satisfaction Problems Formulation

- Finite set of ***variables X*** $= \{X_1, X_2, \ldots, X_n\}$
- Non-empty ***domain D*** of k possible values for each variable D_i where $D_i = \{v_1, \ldots, v_k\}$
- Finite set of ***constraints C*** $= \{C_1, C_2, \ldots, C_m\}$
  - Each constraint C_i limits the values that variables can take, e.g., $X_1 \neq X_2$

Constraint Satisfaction Problems Formulation

- Finite set of ***variables X*** $= \{X_1, X_2, \ldots, X_n\}$
- Non-empty ***domain D*** of k possible values for each variable D_i where $D_i = \{v_1, \ldots, v_k\}$
- Finite set of ***constraints C*** $= \{C_1, C_2, \ldots, C_m\}$
  - Each constraint C_i limits the values that variables can take, e.g., $V_1 \neq V_2$
- In relation to a search problem, we have:
  - **State** is defined by *variables* X_i that take on values from *domain* D_i
  - **Goal Test** is a set of *constraints* C_i specifying *allowable combinations of values* for subsets of variables

Constraint Satisfaction Problems

- A *complete* assignment is where every variable is assigned a value
- A *consistent* assignment does not violate any constraint
- A *CSP solution* is a complete and consistent assignment for all variables

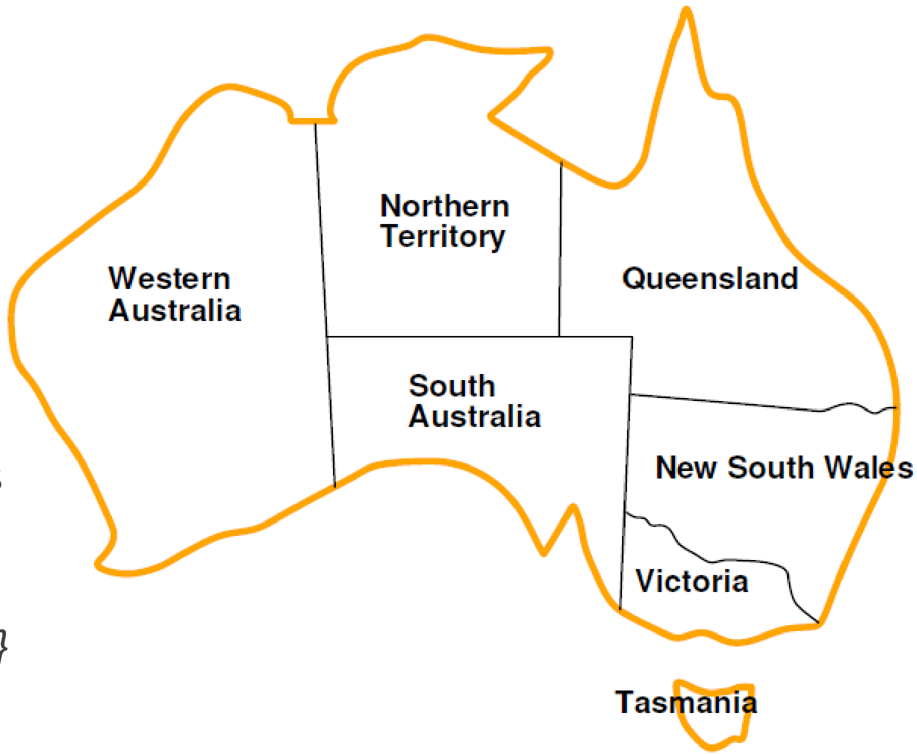
Advantages of CSPs

- *Formal representation language* that can be used to formalize many problems types
  - Represent problem as a CSP and solve with general-purpose solver

- Able to use *general-purpose solver*, which are generally more efficient than standard search
  - Constraints allow us to focus the search to valid branches
  - Branches that violate constraints are removed
    - Non-trivial to do this for standard search (need manual selection of actions)

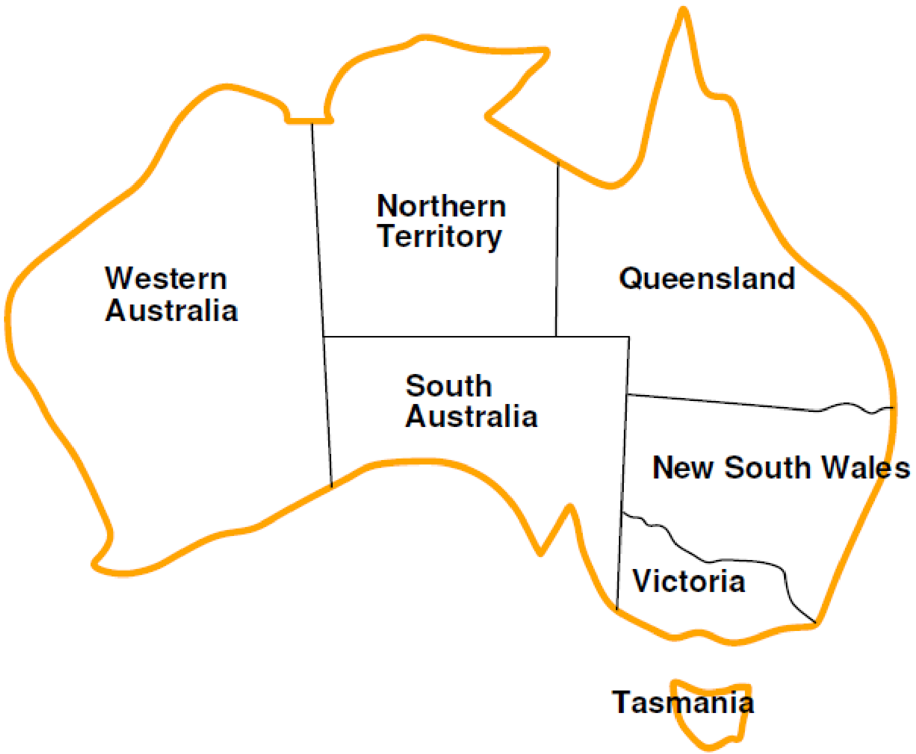
Exercise: Map Colouring

- Variables:
  - *WA, NT, Q, NSW, V , SA, T*
- Domains:
  - D_i = *{red, green, blue}*
- Constraints:
  - Adjacent regions must have different colors
  - E.g., *WA ≠ NT* (if the language allows this), or
  - *E.g., (WA,NT) ∈ {(red, green), (red, blue), (green, red), (green, blue), . . .}*



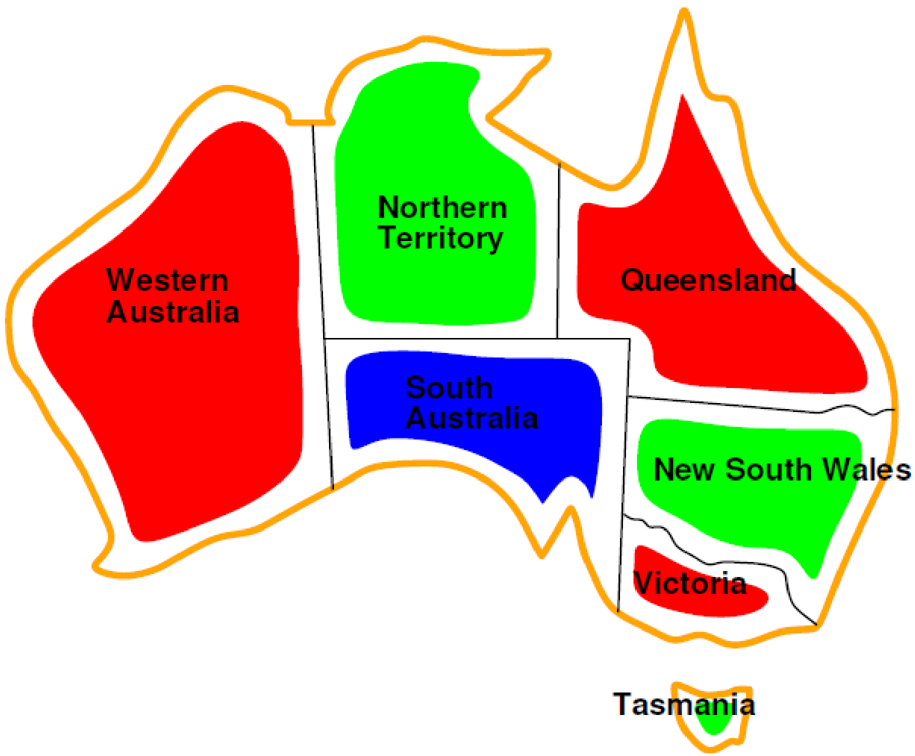
Exercise: Map Colouring

- Now, try to complete this map colouring exercise
- *What strategies did you use to:*
  - *Select the state to start?*
  - *Assign colours to states?*
  - *Select the next state to continue?*



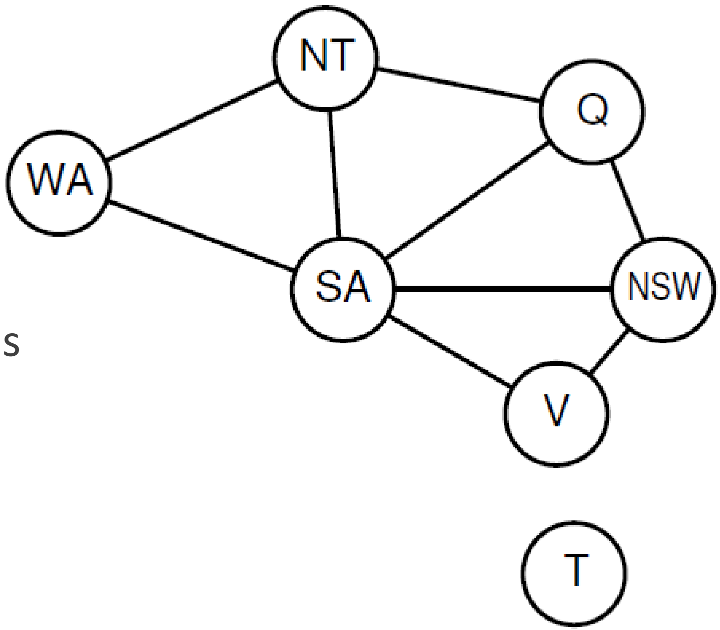
Exercise: Map Colouring

- Solutions are *complete and consistent* assignment for all variables
- That is, all variables are assigned and all constraints are satisfied
  - E.g., *{WA=red, NT =green, Q=red, NSW=green, V=red, SA=blue, T=green}*



Constraint graph

- Similar to search problems, we can represent CSPs using graphs
- Constraint graphs
  - Nodes = variables
  - Edges = constraints
- Binary CSPs
  - Each constraint related to at most two variables
- General-purpose CSP algorithms use the graph structure to speed up the search
  - E.g., Tasmania is an independent subproblem!



Varieties of CSPs

- ***Discrete variables***
  - *Finite domains*: $O(d^n)$ complete assignments for n variables, domain size d
    - e.g., map-colouring, scheduling with time limits
  - *Infinite domains*: Integers, strings, etc.
    - e.g., job scheduling, variables are start/end days for each job
    - Need a constraint language, e.g., $StartJob_1 + 5 \le StartJob_3$

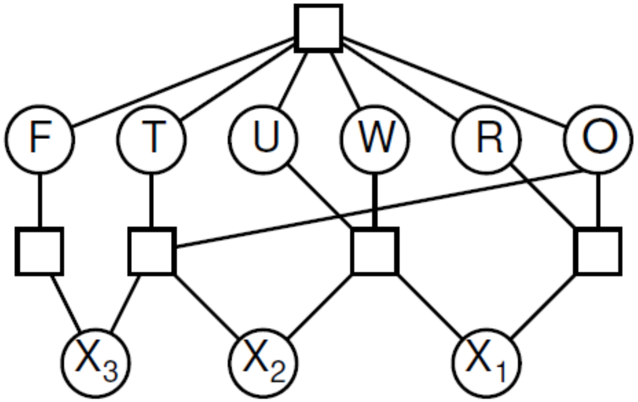
- ***Continuous variables***
  - e.g., start/end times for Hubble Space Telescope observations

Varieties of Constraints

- *Unary* constraints involve a single variable,
  - e.g., SA ≠ green
- *Binary* constraints involve pairs of variables,
  - e.g., SA ≠ WA
- *Higher-order* constraints involve 3 or more variables
  - e.g., cryptarithmetic column constraints
- *Preference* (soft constraints)
  - e.g. *red is better than green* can be represented by a cost for each variable assignment, aka constrained optimization problems.

Exercise: Cryptarithmetic

```
   T W O
 + T W O
 -------
 F O U R
```

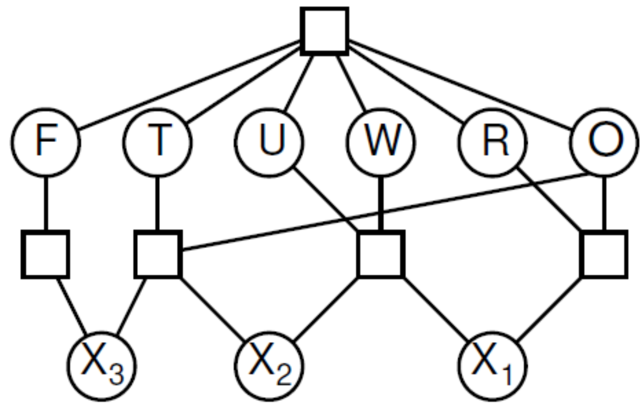


- Variables: *F T U W R O X1 X2 X3*
- Domains: *{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}*
- Constraints
  - *alldiff(F, T,U,W,R,O)*
  - $O + O = R + 10 \cdot X1$, *etc.*

Exercise: Cryptarithmetic

***Task:** Work out the remaining constraints*

```
   T W O
 + T W O
 -------
 F O U R
```

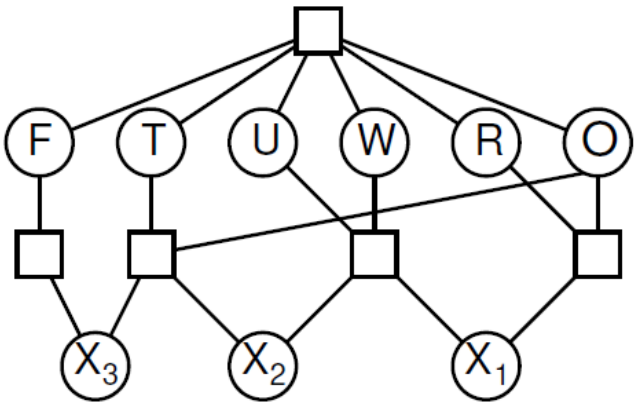


- Variables: *F T U W R O X1 X2 X3*
- Domains: *{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}*
- Constraints
  - *alldiff(F, T,U,W,R,O)*
  - *O + O = R + 10 · X1, etc.*

Exercise: Cryptarithmetic

Task: *Work out the remaining constraints*

```
    T W O
+   T W O
---------
  F O U R
```



- Variables: *F T U W R O X1 X2 X3*
- Domains: *{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}*
- Constraints
  - *alldiff(F, T,U,W,R,O)*
  - $O + O = R + 10 \cdot X1$, *etc.*
  - $W + W + X_1 = U + 10 \cdot X_2$
  - $T + T + X_2 = O + 10 \cdot X_3$
  - $X_3 = F$

Real-world CSPs

- Assignment problems
  - e.g., who teaches what class
- Timetabling problems
  - e.g., which class is offered when and where?
- Hardware configuration
- Spreadsheets
- Transportation scheduling
- Factory scheduling
- Floor planning

- ***Notice that many real-world problems involve real-valued variables***

Solving CSPs

- One idea is to solve CSPs as though it were standard search
  - Then we can apply the standard search algorithms
- First need to formulate this search problem

CSPs as Standard Search

- CSPs can be easily formulated/represented as standard search problems
  - *Initial State:* An empty assignment {}
  - *Successor function (actions):* Assign a value to an unassigned variable
  - *Path cost:* A constant cost for every step
  - *Goal test:* The current assignment is complete and consistent

CSPs as Standard Search

- For CSPs, the sequence of actions or path is not important, only the final goal state
  - E.g., for map colouring, it does not matter the sequence in which you colour the states, as long as all states are coloured with no conflicting colours

- Thus, the solution will appear at depth *n* for *n* variables
  - To find the solution, we can use depth-first search

CSPs as Standard Search

- For CSPs, the sequence of actions or path is not important, only the final goal state
  - E.g., for map colouring, it does not matter the sequence in which you colour the states, as long as all states are coloured with no conflicting colours

- Thus, the solution will appear at depth *n* for *n* variables
  - To find the solution, we can use depth-first search
  - There are potentially $n!d^n$ leaves in the search tree
    - *How did we get this?*

CSPs as Standard Search

- There are potentially $n!d^n$ leaves in the search tree
  - *How did we get this?*

- For a CSP with *n* variables with *d* domains, we have:
  - Depth 1: branching factor of *nd*
  - Depth 2: branching factor of *(n-1)d*
  - Depth 3: branching factor of *(n-2)d*
  - …
  - Depth *n*: branching factor of *d*

CSPs as Standard Search

- There are potentially $n!d^n$ leaves in the search tree
  - *How did we get this?*

- $n!d^n$ leaves is too many to search through, can we reduce it?
  - Next: Backtracking search with various enhancements