Artificial Intelligence

CS3AI18/ CSMAI19

Lecture - 3/10: Problem Solving (Constraint Satisfaction Problems)

DR VARUN OJHA

Department of Computer Science

University of Reading

Learning Objectives

- On completion of this week, you will be able to
    - Understand constraint satisfaction problems and its representation
    - Apply methods to solve constraint satisfaction problems such as:
        - Search tree (depth first search)
        - Backtracking search
        - Heuristics search

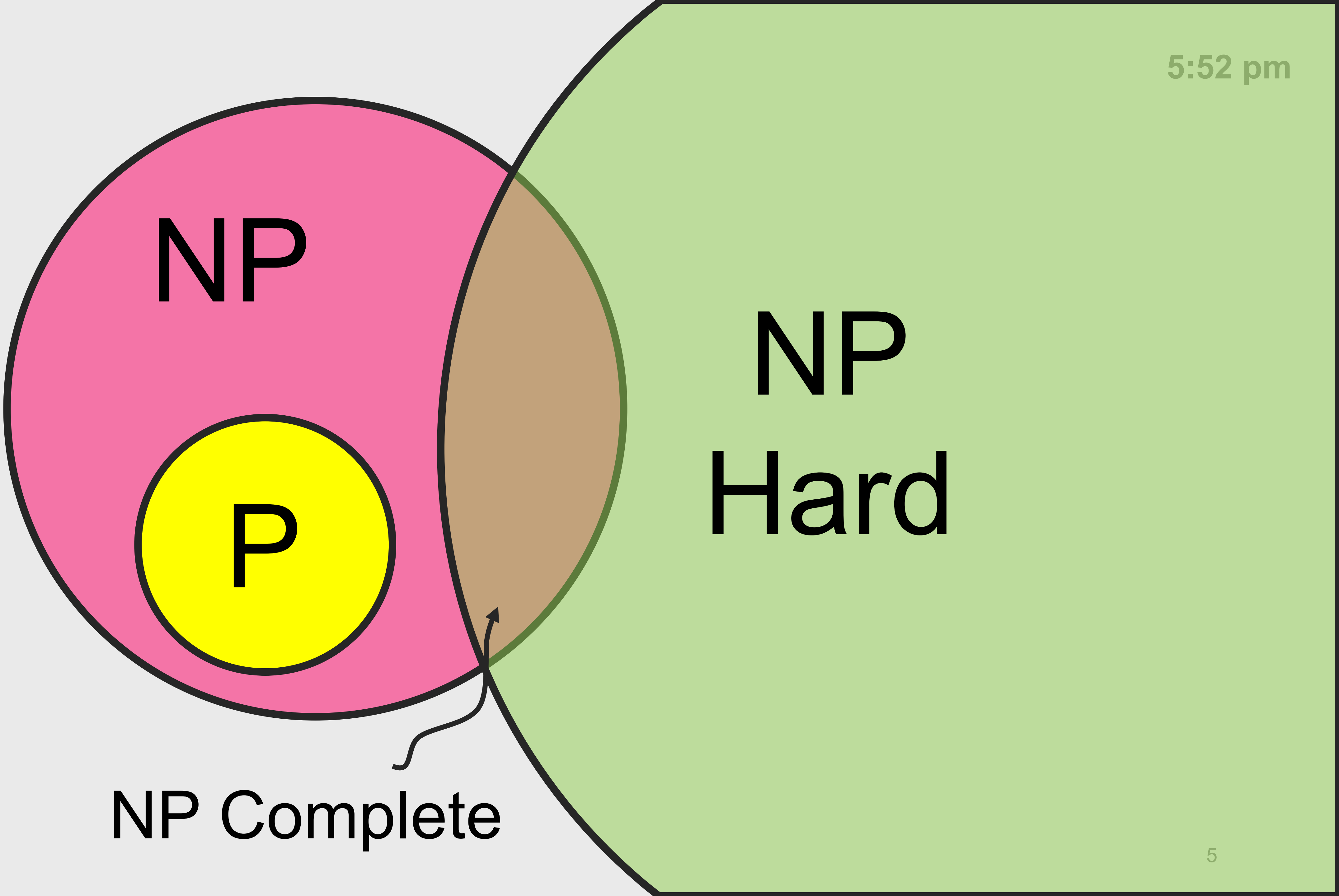
Content of this Lecture

Introduction

- Part – I : Constrain Satisfaction Problems and Solution Approach
- Part – II : Mathematical Formulation
- Part – III : Backtracking Search
- Part – IV : Heuristic Search
- Quiz

Part 1

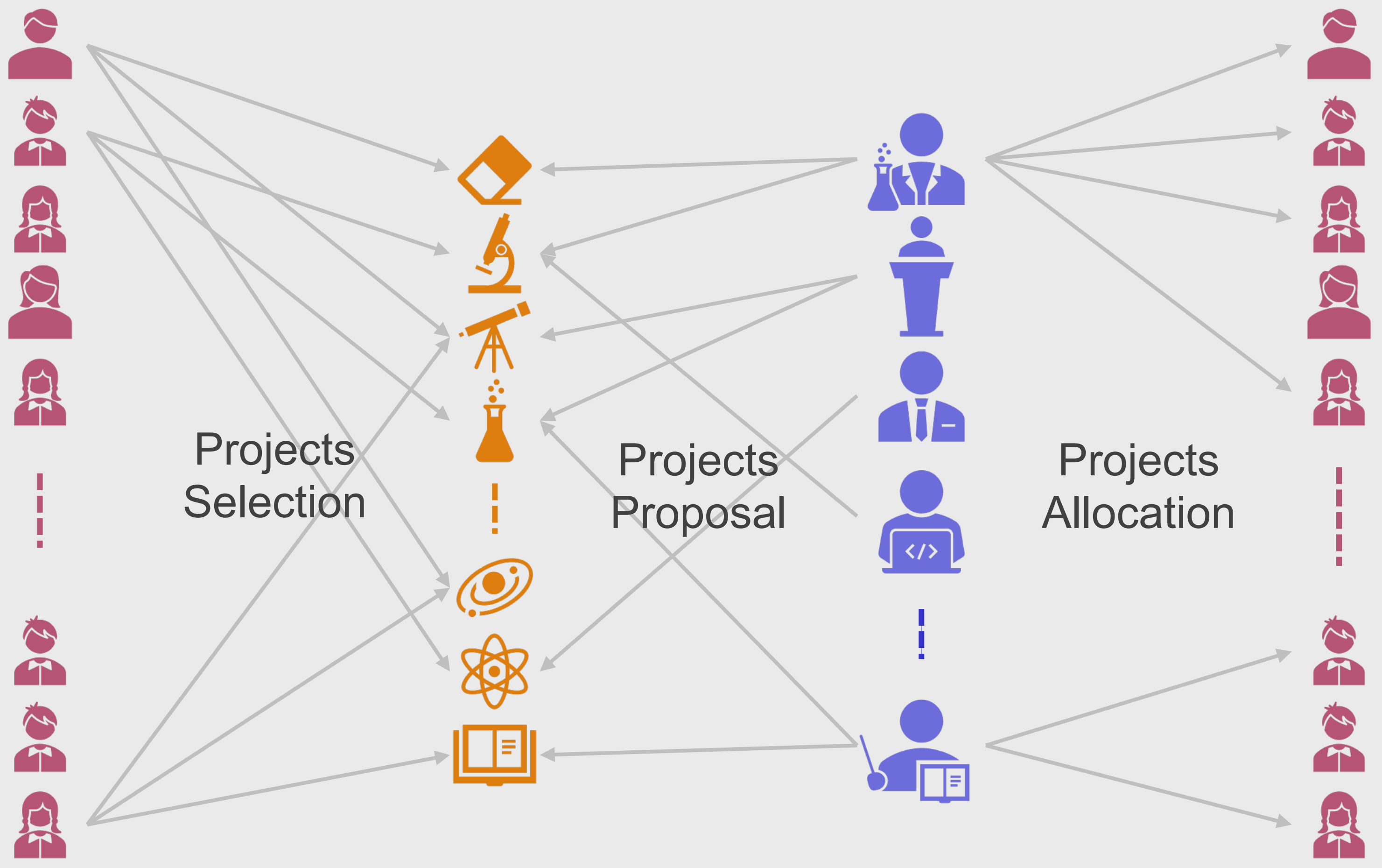
Constraints Satisfaction Problem



NP Complete

Final Year Project Supervisor Allocation

5:52 pm



Projects Selection

Projects Proposal

Projects Allocation

Students

Projects

Supervisors

Students

Constrains?

How many available projects

How many choices of projects

How many supervisors

How may students one supervisor can supervisor

What projects a supervisor will supervisor

Is there an unavailability

University Timetable

Constrains?

How many lecture rooms

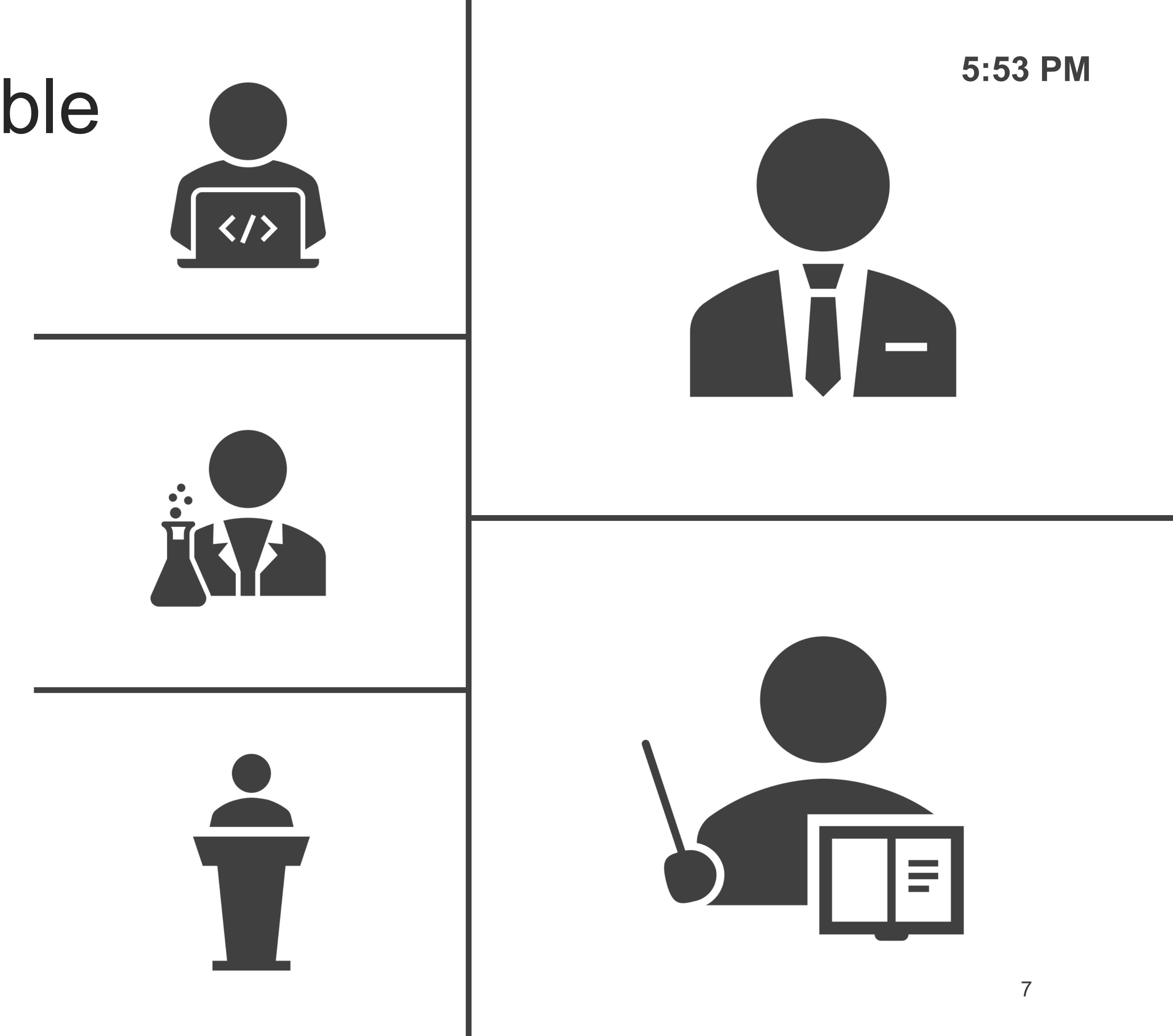
What are the sizes of rooms

How many students in a lecture

How many lectures

How many lecturers

What are the type of lectures



5:53 PM

Satisfiability Problems

Scheduling, Planning, Software Verification problems and many more can be represented as Satisfiability Problems

Satisfiability Problems

variables take certain values for which there is a solution

Boolean Satisfiability (SAT) Problem

NP- Complete (solvable in polynomial time)

Boolean Satisfiability or simply **SAT** is the problem of determining if a Boolean formula is satisfiable or unsatisfiable.

Satisfiable : If the Boolean variables can be assigned values such that the formula turns out to be **TRUE**, then we say that the formula is satisfiable.

$F = A \wedge \neg B$ is **Satisfiable**

Unsatisfiable : If it is not possible to assign such values, then we say that the formula is unsatisfiable.

$F = A \wedge \neg A$ is **Unsatisfiable**

Satisfiability (SAT) Problem

We use conjunctive normal form (CNF) formulas for satisfiability problems.

An example may be:

$$(A \vee B \vee \neg C) \ \wedge (B \vee \ D)$$

Where

- $(A \vee B \vee \neg C)$ is a **Clause**, which is a disjunction a of literals
- $A, \ B,$ and $\neg C$ are **literals**, each of which is a variable or the negation of a variable
- Each clause is a requirement which must be satisfied
- Number of literals in a clause determine type of Satisfiability problems
- A **k-SAT problem** is the one where a clause has at most **k** literals.

Conjunctive Normal Form

$$(A \vee B) \;\rightarrow\; (C \;\rightarrow\; D)$$

- Eliminate arrows ($A \;\rightarrow\; B$ can be written as $\neg A\,VB$)

$$\neg(\,A \vee\; B) \;\vee\; (\neg C \;\vee\; D)$$

- Drive in negations

$$(\neg A \;\wedge\; \neg B) \;\vee\; (\neg C \;\vee\; D)$$

- Distribute

$$(\neg A \;\vee\; \neg C \;\vee\; D) \;\wedge\; (\neg B \;\vee\; \neg C \;\vee\; D)$$

Source: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/
6.825 Techniques in Artificial Intelligence

Solving a CNF

$$(P \lor Q) \land (P \lor \neg Q \lor R) \land (T \lor \neg R) \land (\neg P \lor \neg T) \land (P \lor S) \land (T \lor R \lor S) \land (\neg S \lor T)$$

If we assign $P = False$ (or say 0), we get simpler set of constraints

- $(P \lor Q)$ simplifies to (Q)
- (P ∨¬QvR) simplifies to $(\neg Q \lor R)$
- $(\neg P \lor \neg T)$ simplifies to $True$ (or say 1), i.e., is satisfied and can be removed
- $(P \lor S)$ simplifies to (S)

Result is

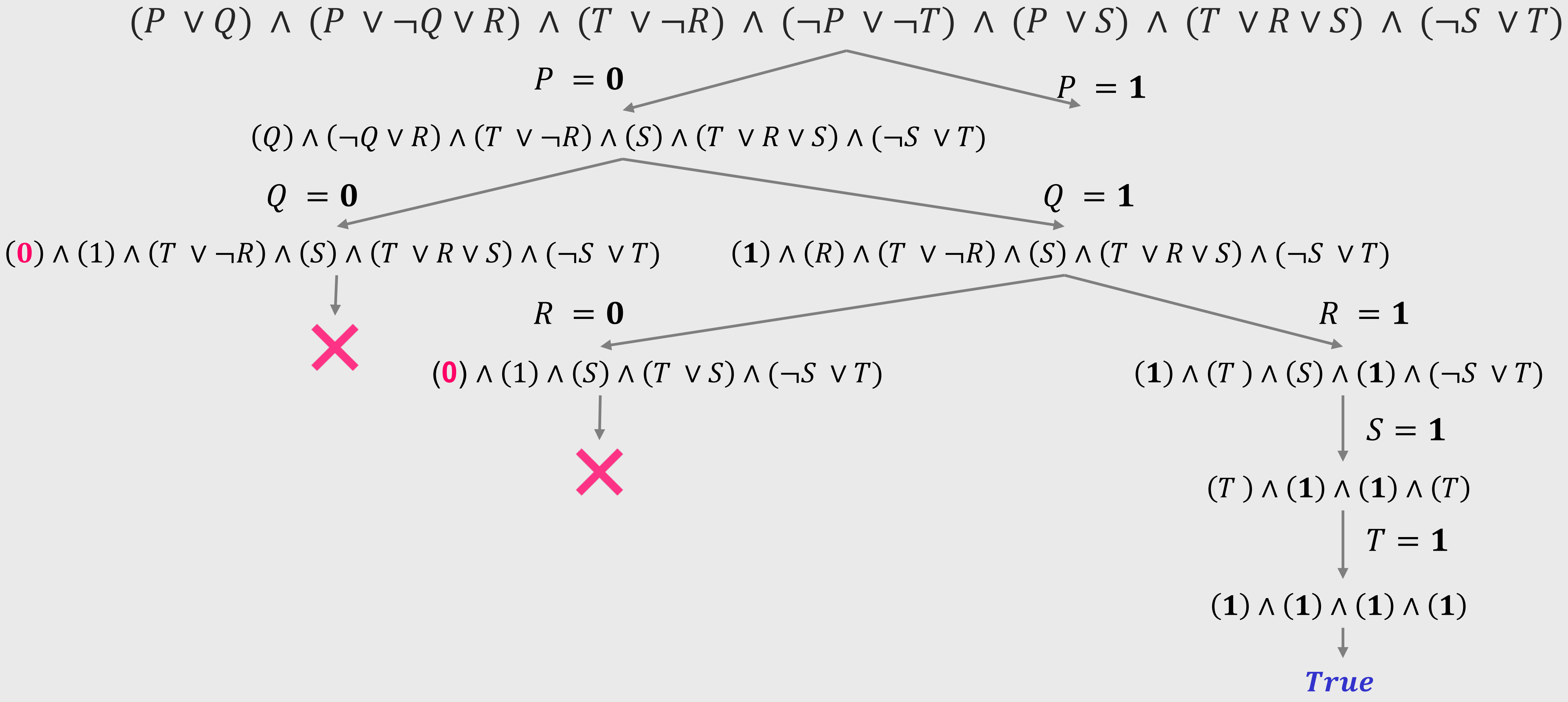
$$(Q) \land (\neg Q \lor R) \land (T \lor \neg R) \land (S) \land (T \lor R \lor S) \land (\neg S \lor T)$$

Source: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/
6.825 Techniques in Artificial Intelligence

Solving a CNF

5:53 pm

Source: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6.825 Techniques in Artificial Intelligence



$$(P \lor Q) \land (P \lor \neg Q \lor R) \land (T \lor \neg R) \land (\neg P \lor \neg T) \land (P \lor S) \land (T \lor R \lor S) \land (\neg S \lor T)$$

$P = \mathbf{0}$

$(Q) \land (\neg Q \lor R) \land (T \lor \neg R) \land (S) \land (T \lor R \lor S) \land (\neg S \lor T)$

$Q = \mathbf{0}$

$(\mathbf{0}) \land (1) \land (T \lor \neg R) \land (S) \land (T \lor R \lor S) \land (\neg S \lor T)$

✗

$Q = \mathbf{1}$

$(\mathbf{1}) \land (R) \land (T \lor \neg R) \land (S) \land (T \lor R \lor S) \land (\neg S \lor T)$

$R = \mathbf{0}$

$(\mathbf{0}) \land (1) \land (S) \land (T \lor S) \land (\neg S \lor T)$

✗

$R = \mathbf{1}$

$(\mathbf{1}) \land (T) \land (S) \land (\mathbf{1}) \land (\neg S \lor T)$

$S = \mathbf{1}$

$(T) \land (\mathbf{1}) \land (\mathbf{1}) \land (T)$

$T = \mathbf{1}$

$(\mathbf{1}) \land (\mathbf{1}) \land (\mathbf{1}) \land (\mathbf{1})$

True

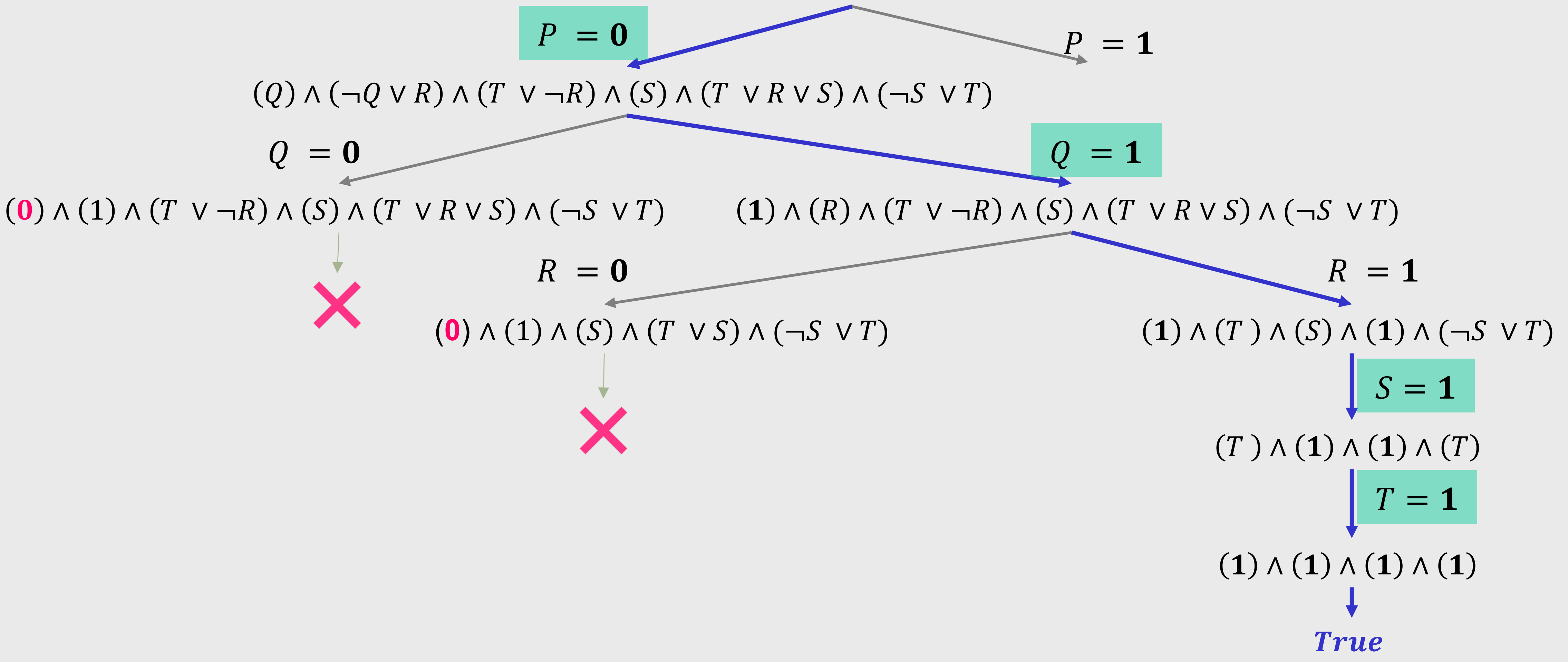
$P = \mathbf{1}$

Solving a CNF

5:53 pm

Source: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6.825 Techniques in Artificial Intelligence

$$(P \vee Q) \wedge (P \vee \neg Q \vee R) \wedge (T \vee \neg R) \wedge (\neg P \vee \neg T) \wedge (P \vee S) \wedge (T \vee R \vee S) \wedge (\neg S \vee T)$$



$P = \mathbf{0}$

$(Q) \wedge (\neg Q \vee R) \wedge (T \vee \neg R) \wedge (S) \wedge (T \vee R \vee S) \wedge (\neg S \vee T)$

$P = \mathbf{1}$

$Q = \mathbf{0}$

$(\mathbf{0}) \wedge (1) \wedge (T \vee \neg R) \wedge (S) \wedge (T \vee R \vee S) \wedge (\neg S \vee T)$

✗

$Q = \mathbf{1}$

$(\mathbf{1}) \wedge (R) \wedge (T \vee \neg R) \wedge (S) \wedge (T \vee R \vee S) \wedge (\neg S \vee T)$

$R = \mathbf{0}$

$(\mathbf{0}) \wedge (1) \wedge (S) \wedge (T \vee S) \wedge (\neg S \vee T)$

✗

$R = \mathbf{1}$

$(\mathbf{1}) \wedge (T) \wedge (S) \wedge (\mathbf{1}) \wedge (\neg S \vee T)$

$S = \mathbf{1}$

$(T) \wedge (\mathbf{1}) \wedge (\mathbf{1}) \wedge (T)$

$T = \mathbf{1}$

$(\mathbf{1}) \wedge (\mathbf{1}) \wedge (\mathbf{1}) \wedge (\mathbf{1})$

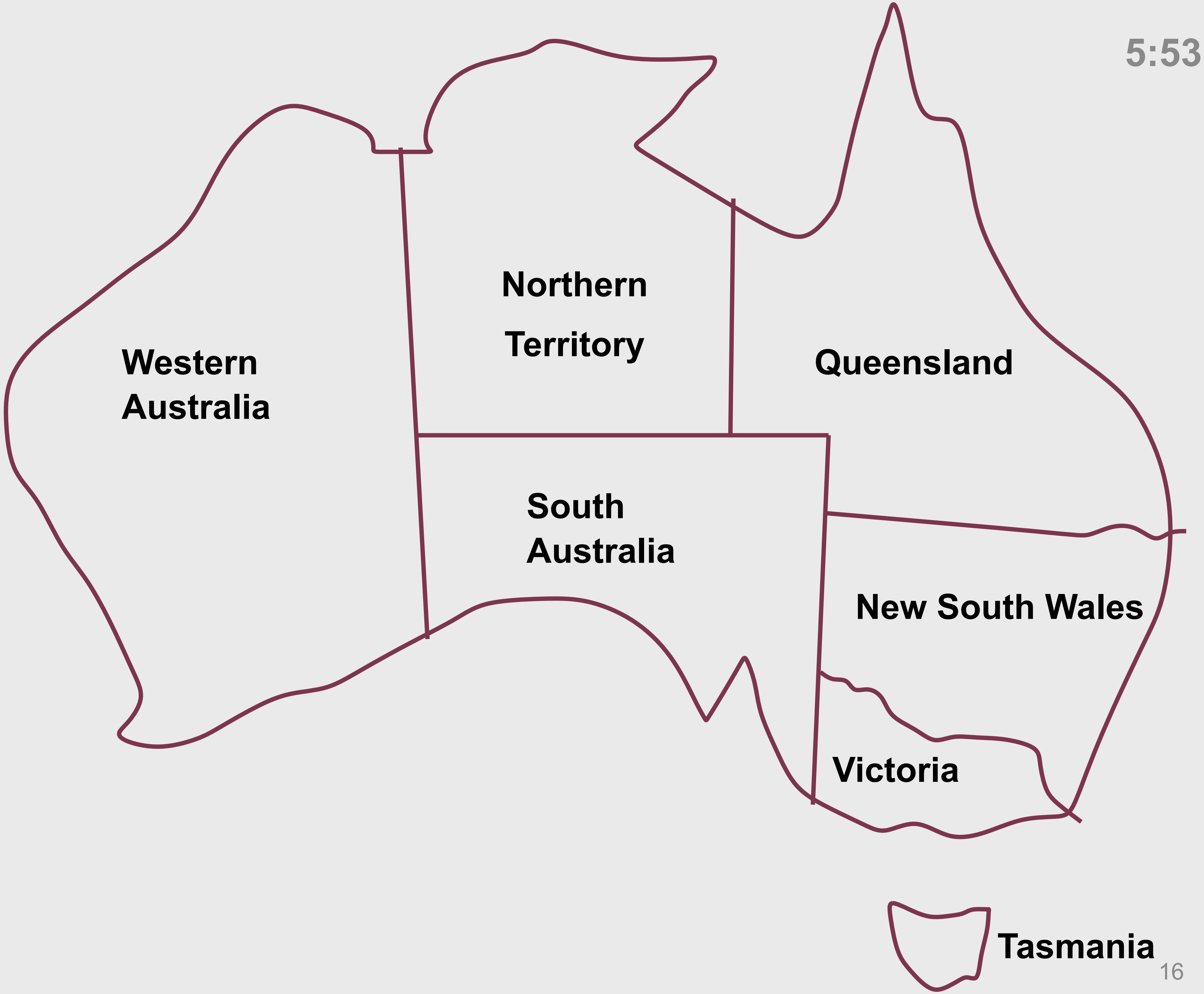
True

Map Colouring

Constraints?

Adjacent regions must have different colours

🟥 🟩 🟦



Some other problems

- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Floor planning

Part 2
Definition

Constraint satisfaction problems (CSPs)

CSPs are search problems

State: Variable X_i with values from domain D_i

Goal: **goal test** is a set of constraints specifying allowable combinations of values for subsets of variables

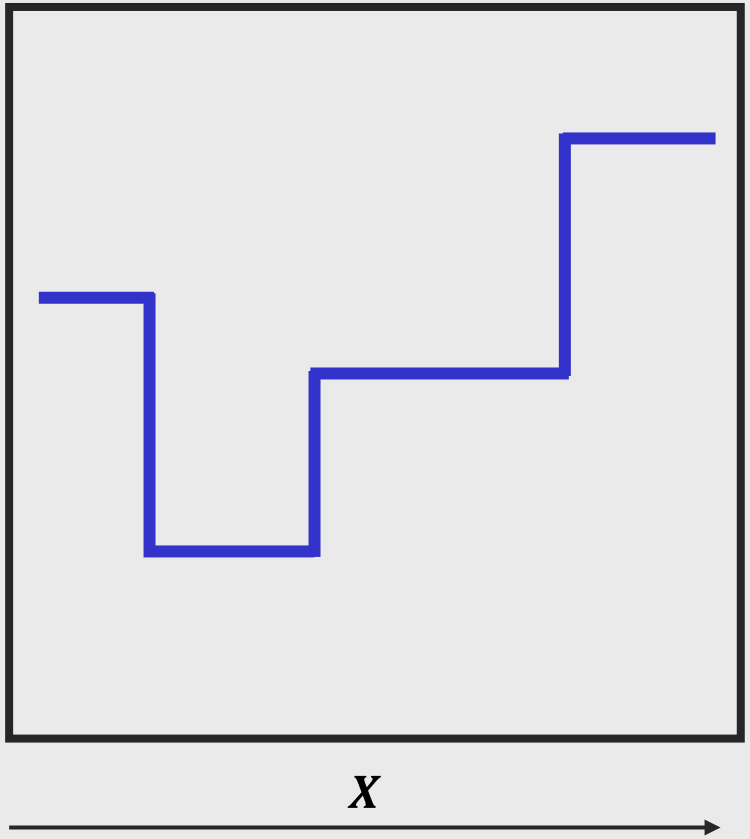
CSPs Components

1 $\boldsymbol{X}$: is a set of variable $X = \{x_1,\ x_2, \dots, x_n\}$

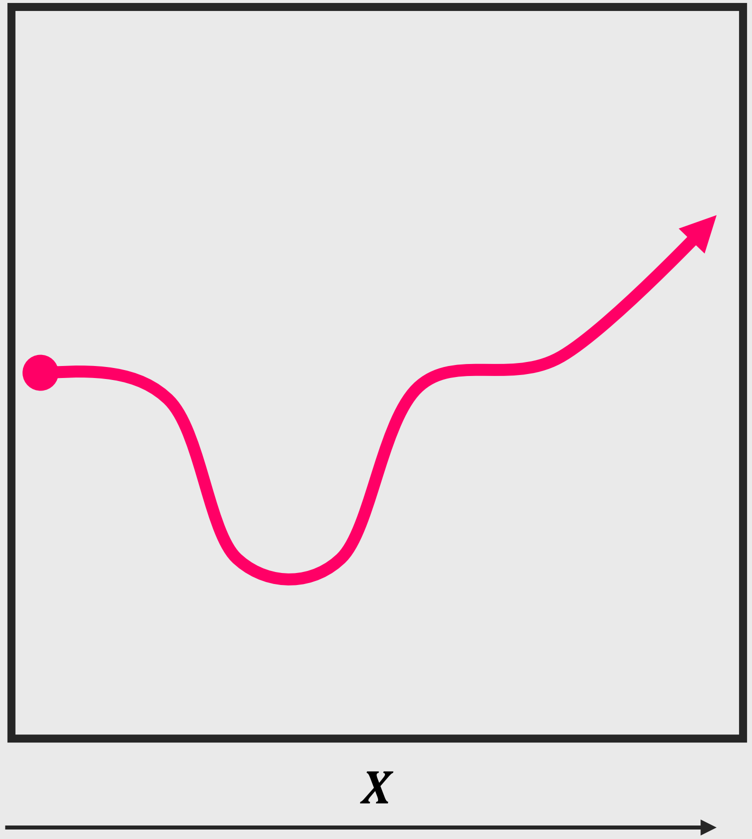
2 $\boldsymbol{D}$: is a set of domain $D = \{d_1,\ d_2, \dots, d_m\}$ for each variable

3 $\boldsymbol{C}$: is a set of constraints

Discrete variables



Continuous variables



Varieties of CSPs

Discrete variables

- Finite domains (simplest CSPs):
  - n variables, domain size $d \Rightarrow O(d^n)$ complete assignments
  - 5 states and 2 colours: 2^5
  - e.g., Binary satisfiability (**NP-complete**)
- Infinite domains: integers, strings, etc.
  - e.g., **job scheduling**, variables are start / end days for each job
  - need a **constraint language**, e.g., **Start Job1 + 5 ≤ Start Job3**
  - linear constraints solvable, nonlinear undecidable

Continuous variables

- linear constraints solvable in **polynomial time** by linear programming

Map Colouring

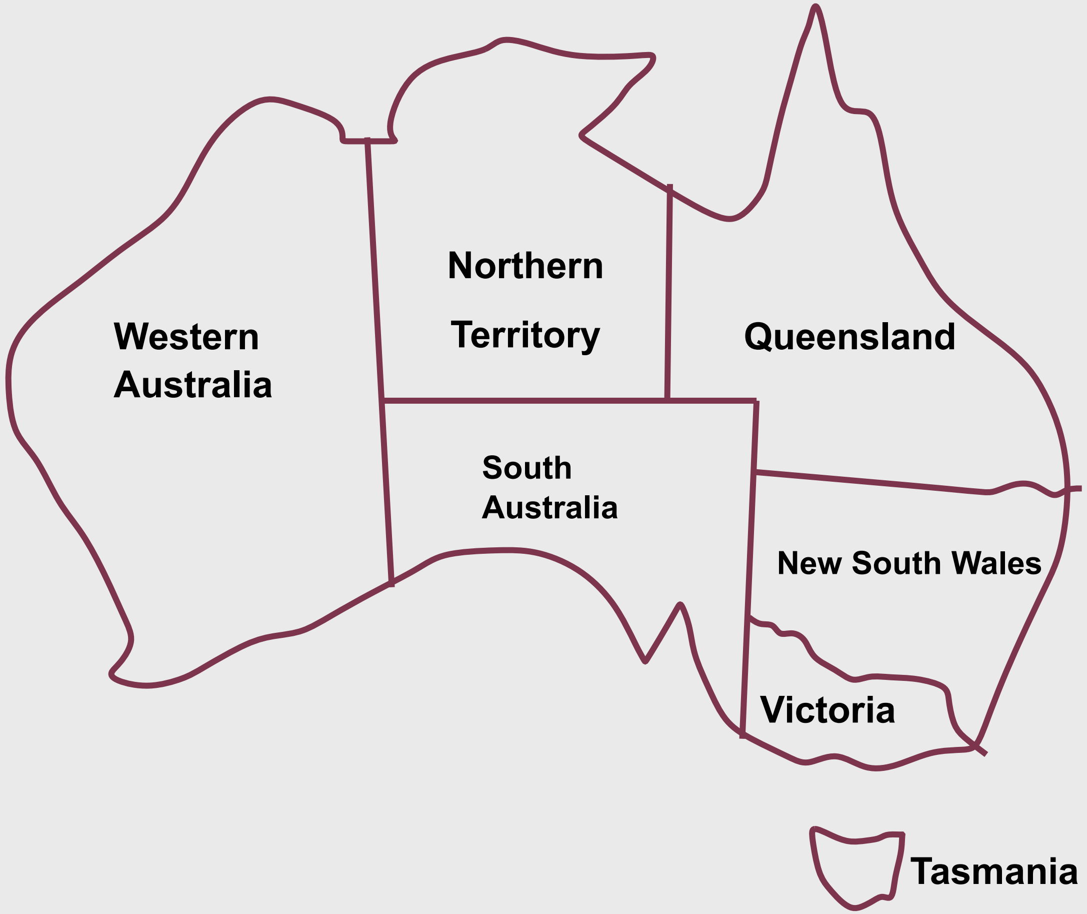
1 $\boldsymbol{X}$: is a set of variable are state name

$X = \{WA, NT, Q, NS, V, SA, T\}$

2 $\boldsymbol{D}$: is a set of colour

$D = \{Red, Green\ Blue\}$ for each Variable state X_1

3 $\boldsymbol{C}$: adjacent sates must have different colours (e.g., $\boldsymbol{WA \neq NT}$)

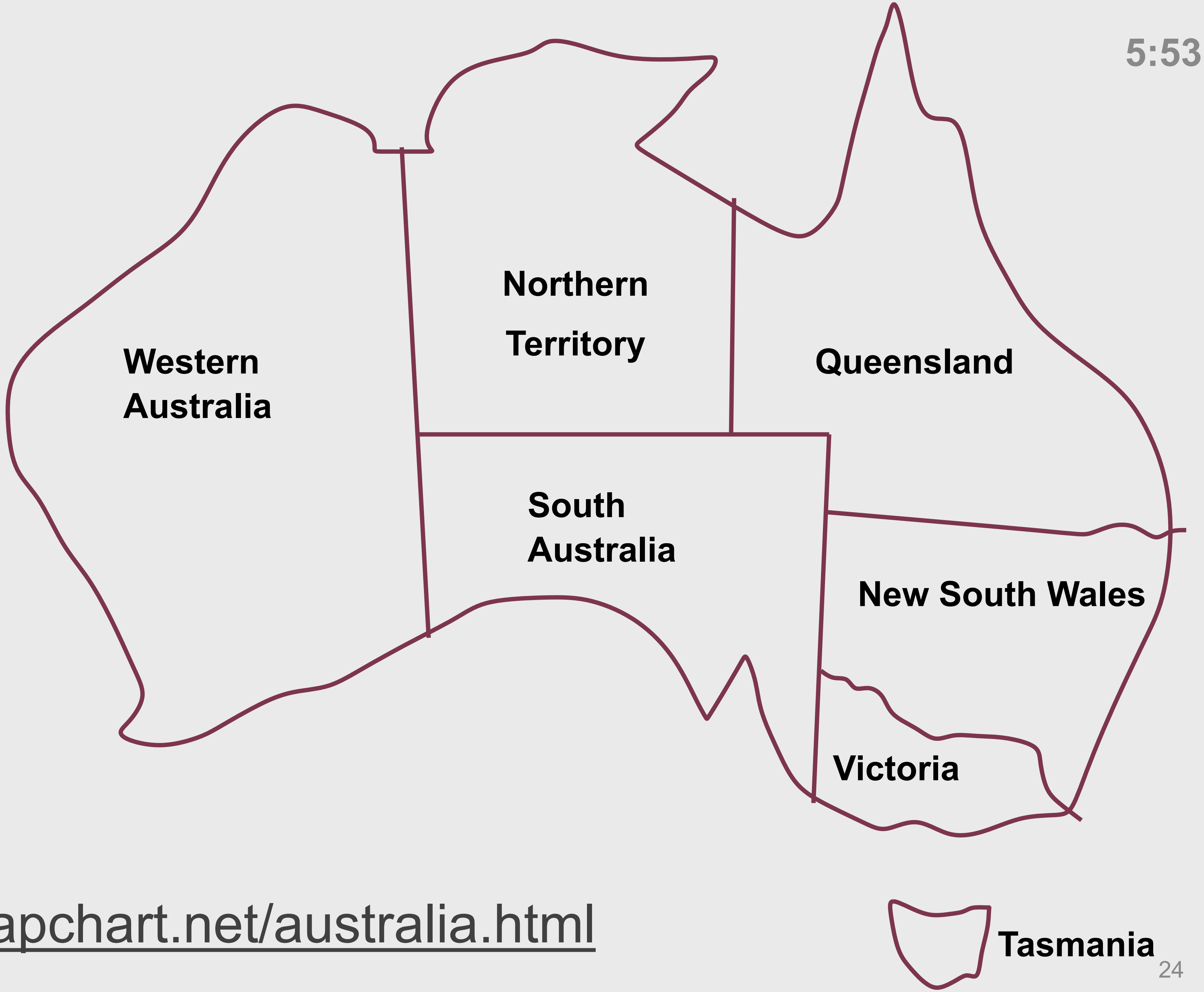


Map Colouring

Constraints?

Adjacent regions must have different colours

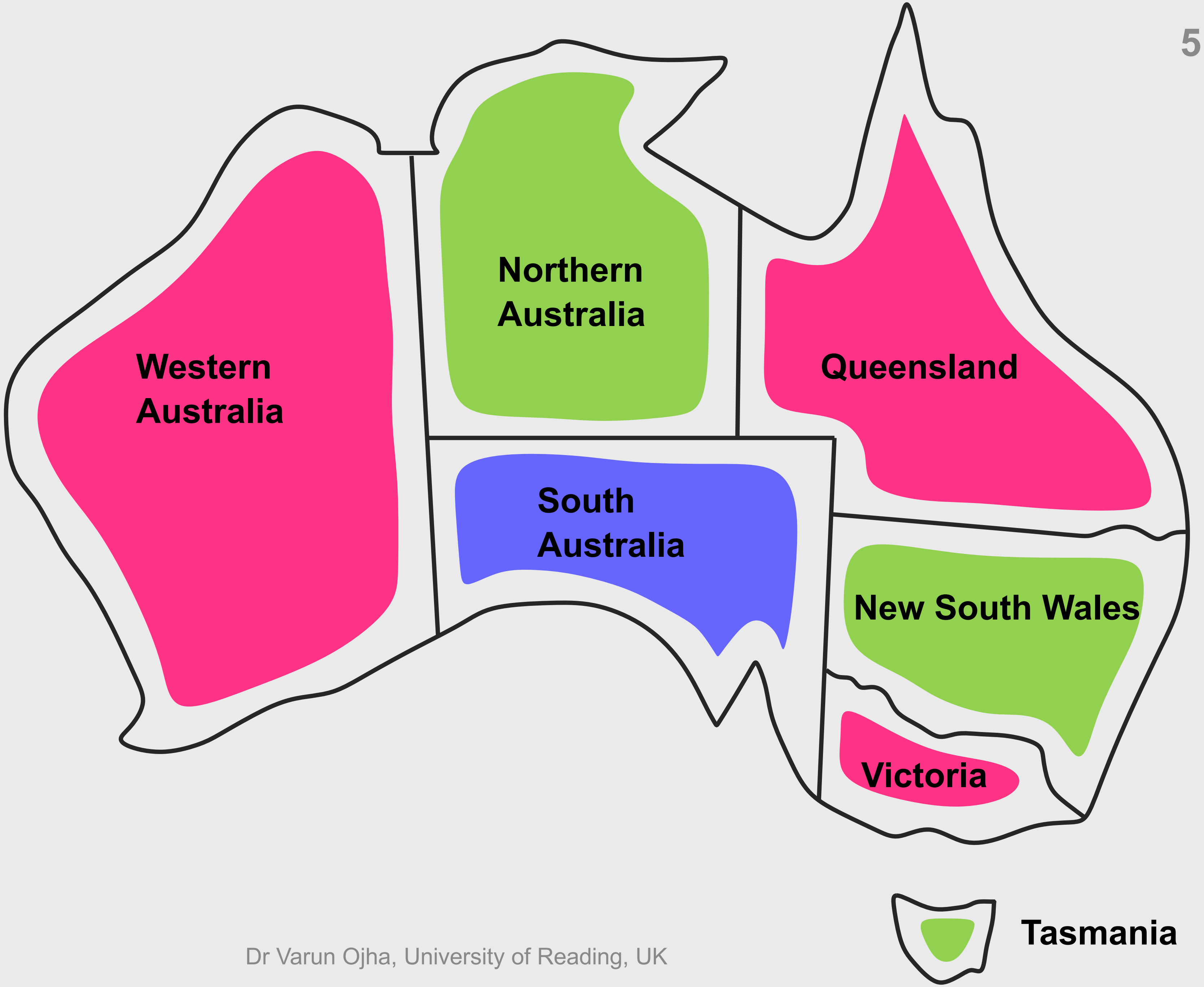
🟥 🟩 🟦



Fun App: https://mapchart.net/australia.html

Map Colouring

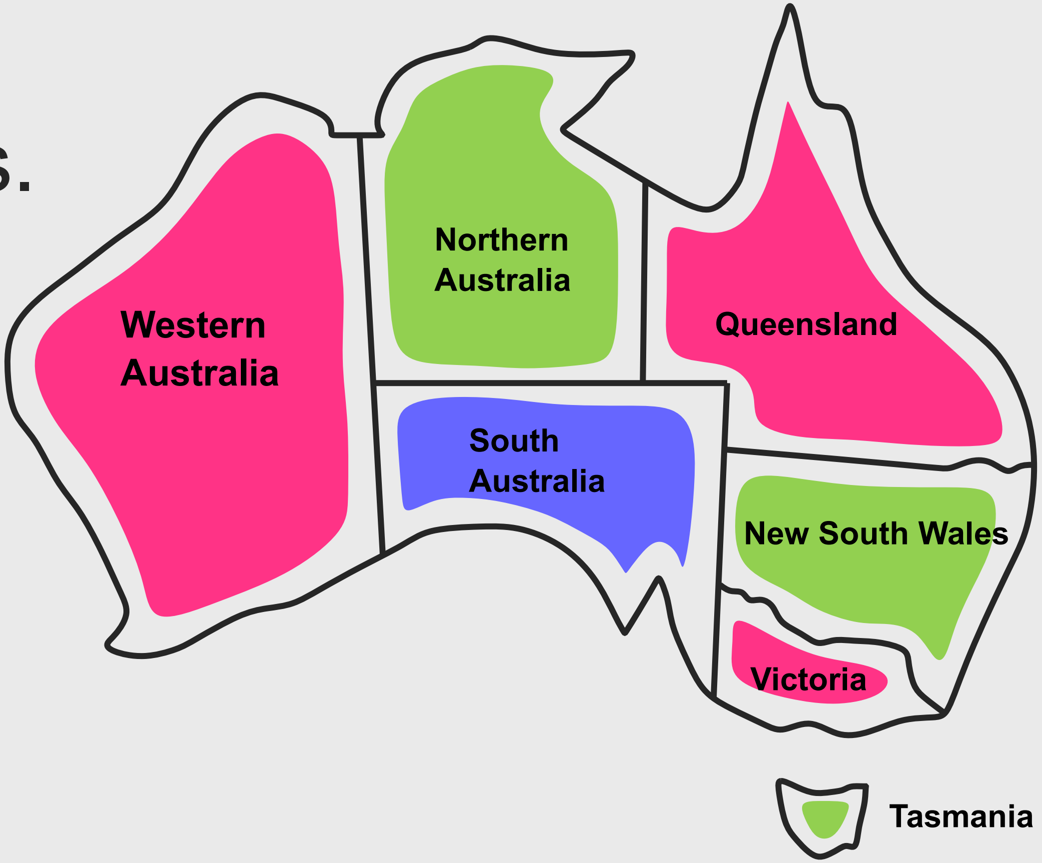
5:53 pm



Map Colouring

Solutions are **complete** and has **consistent** assignments.

WA = red,
NT = green,
Q = red,
NSW = green,
V = red,
SA = blue,
T = green



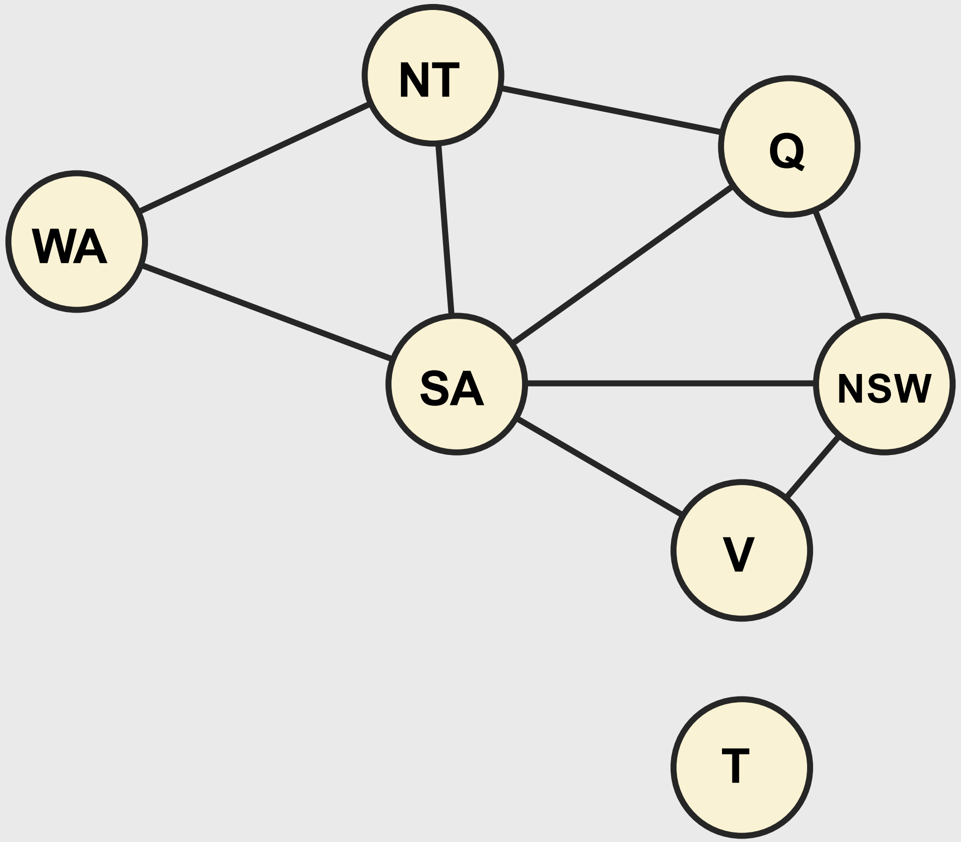
Constraints Graph

Constraint graph

Binary CSPs: each constraint relates at most two variables

Constraint graph: nodes are variables, arcs (links) show constraints

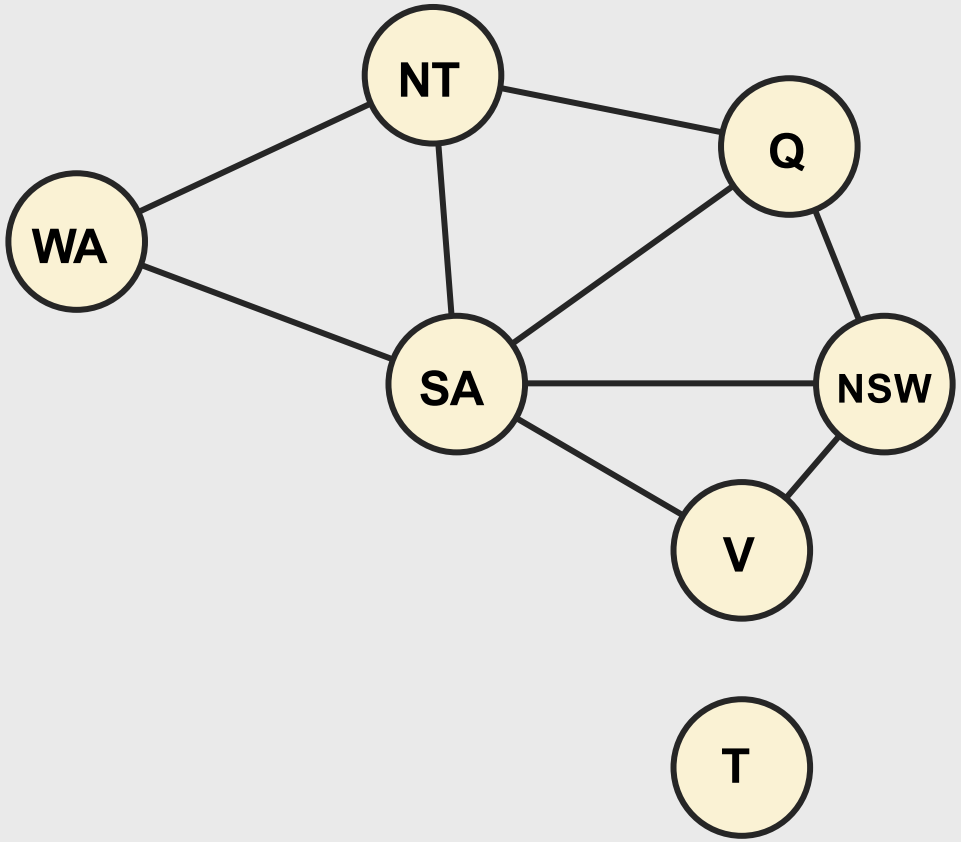
Constraint graph



Constraint graph

General-purpose CSP algorithms use the **graph structure** to speed up search.

E.g., Tasmania is an independent subproblem



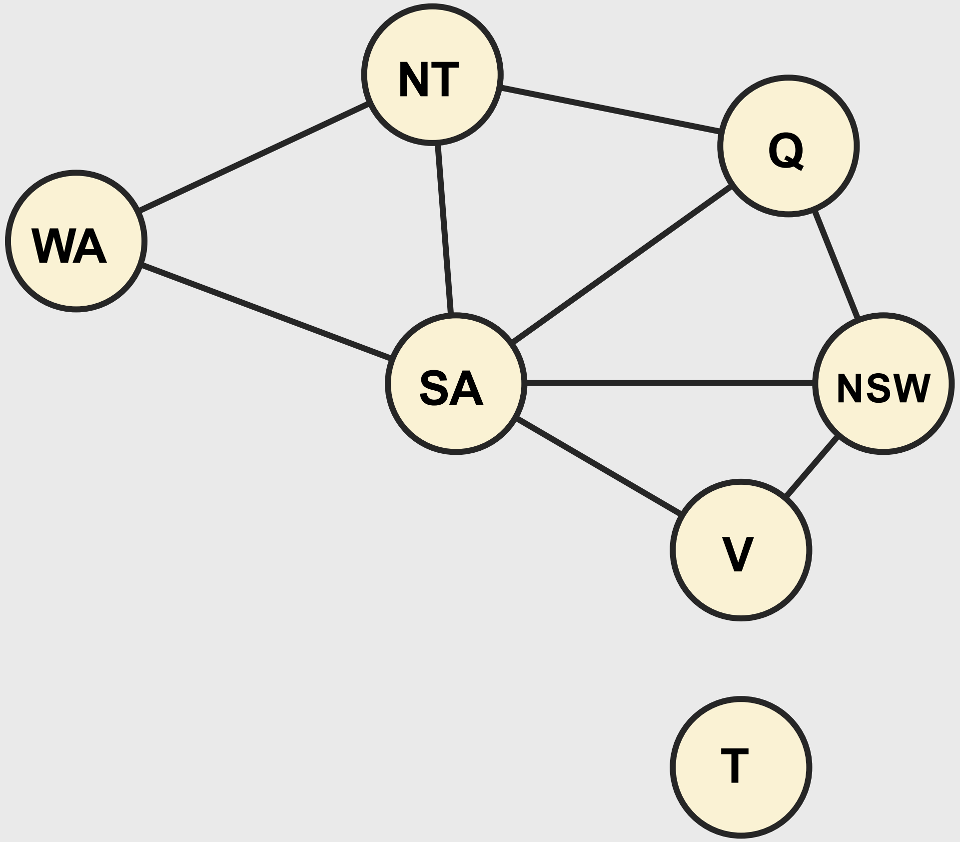
Constraint graph

CSPs are faster ways to solve problems

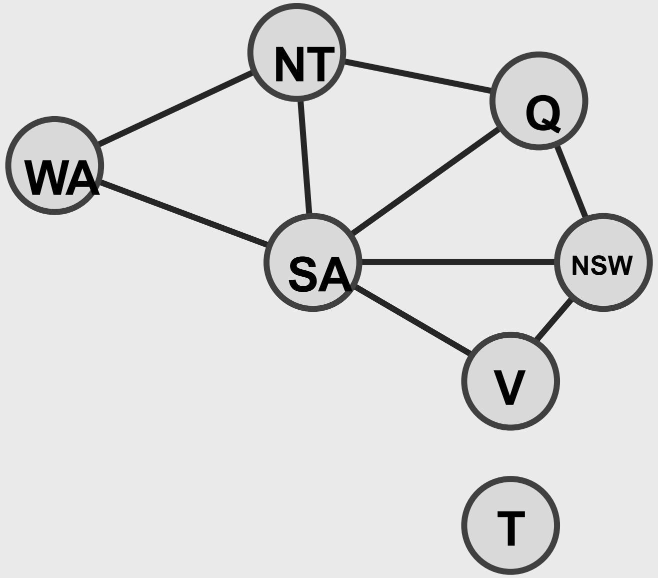
E.g., If SA = **blue then** the other five linked states will not take blue that is **$2^5 = 32$** possible assignments.

Else any other search algorithm will search **$3^5 = 243$** assignments.

87% reduction



Variations 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., SA ≠ WA ≠ NT

Part 3

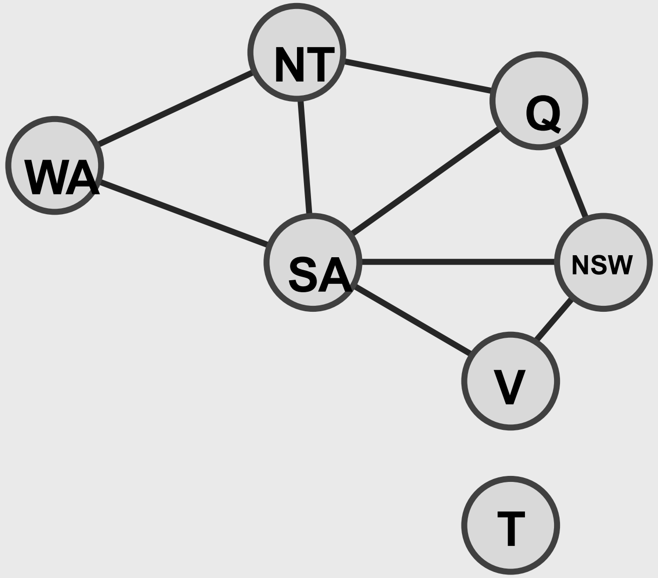
Backtracking Search

Standard Search Formulation

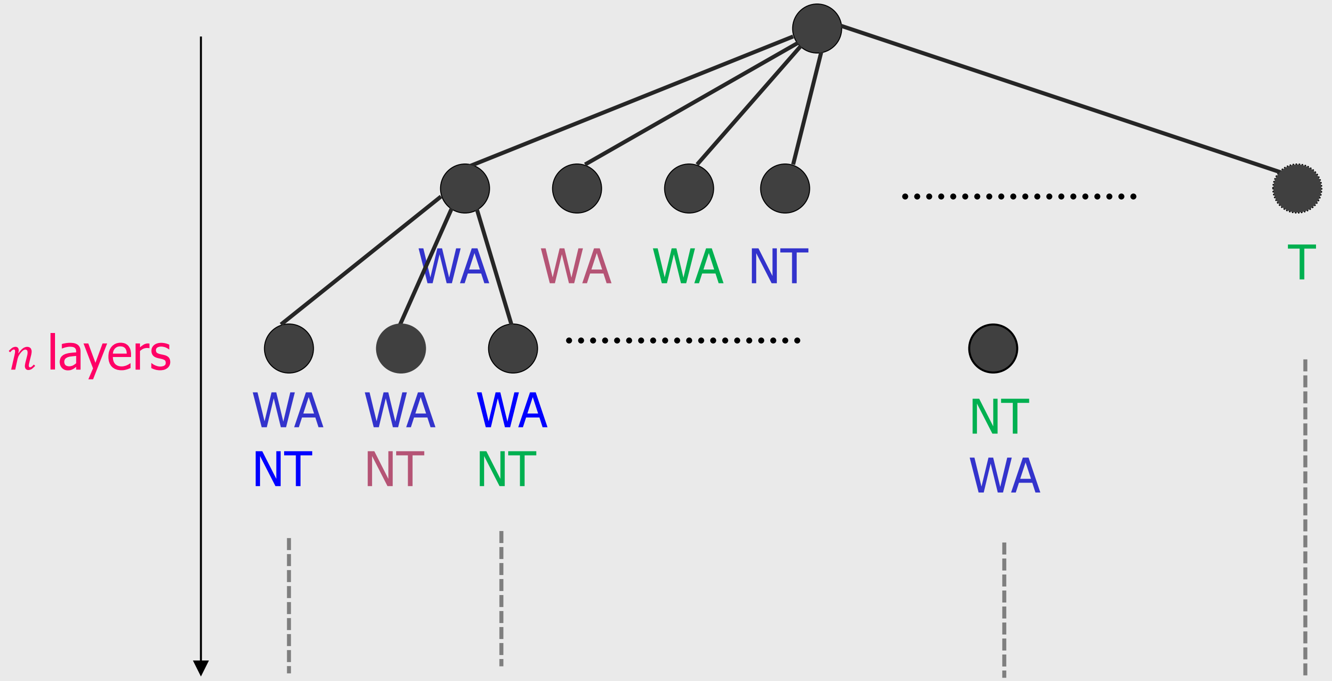
1 **Initial state:** none of the variables has a value (colour), the empty assignment, { }

2 **Successor state:** one of the variables without a value will get some value that does not conflict with constraints.

3 **Goal state:** all variables have a value and none of the constraints is violated.



Standard Search Formulation (Contd..)

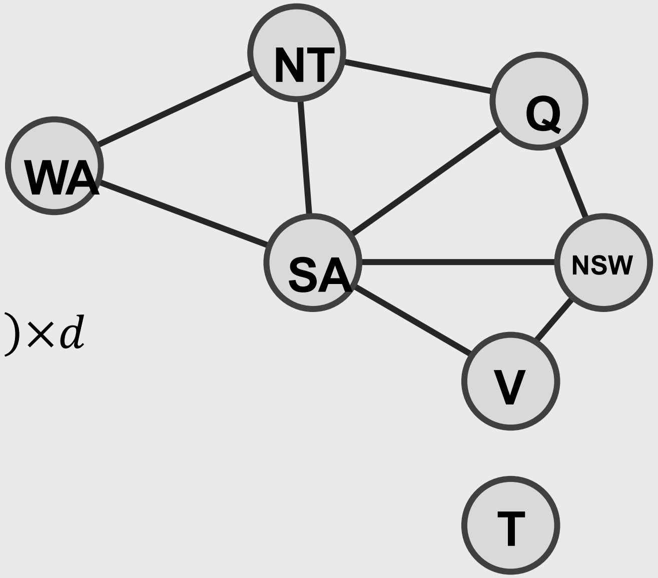


$n\times d$

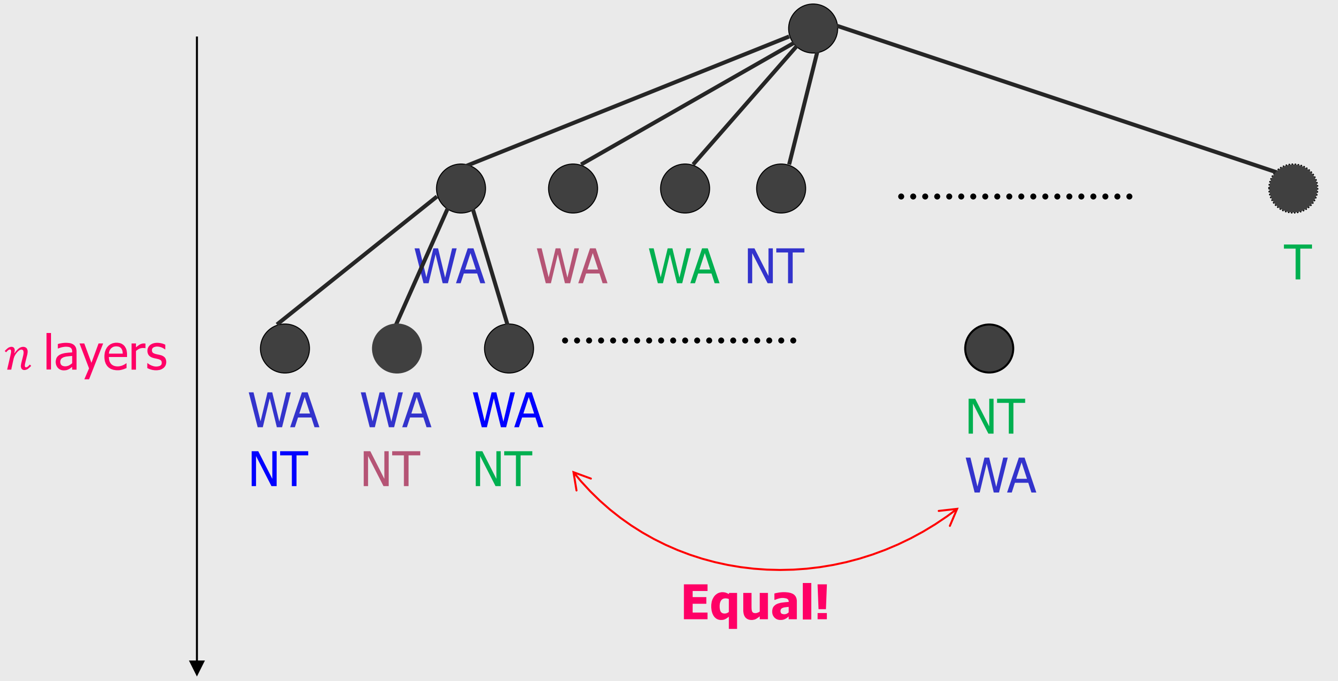
$(n\times d)\times(n-1)\times d$

$n!\times d^n$ **leaves**

TERRIBLE !



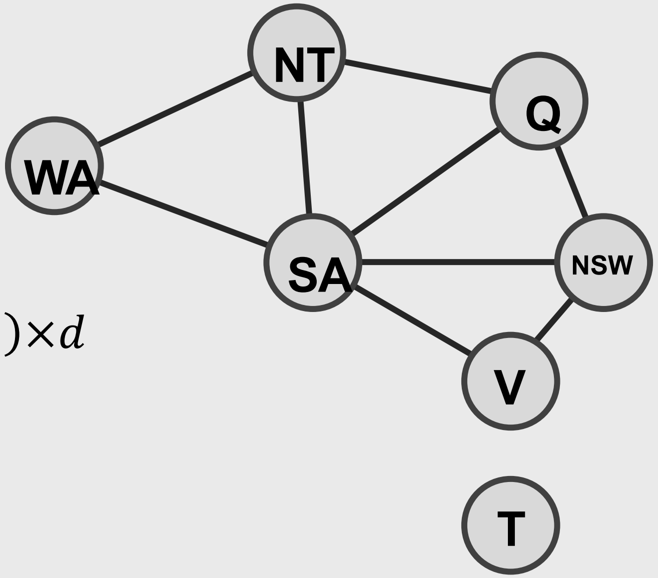
Special property of CSPs: Commutativity



$n \times d$

$(n \times d) \times (n-1) \times d$

$n! \times d^n$ **leaves**

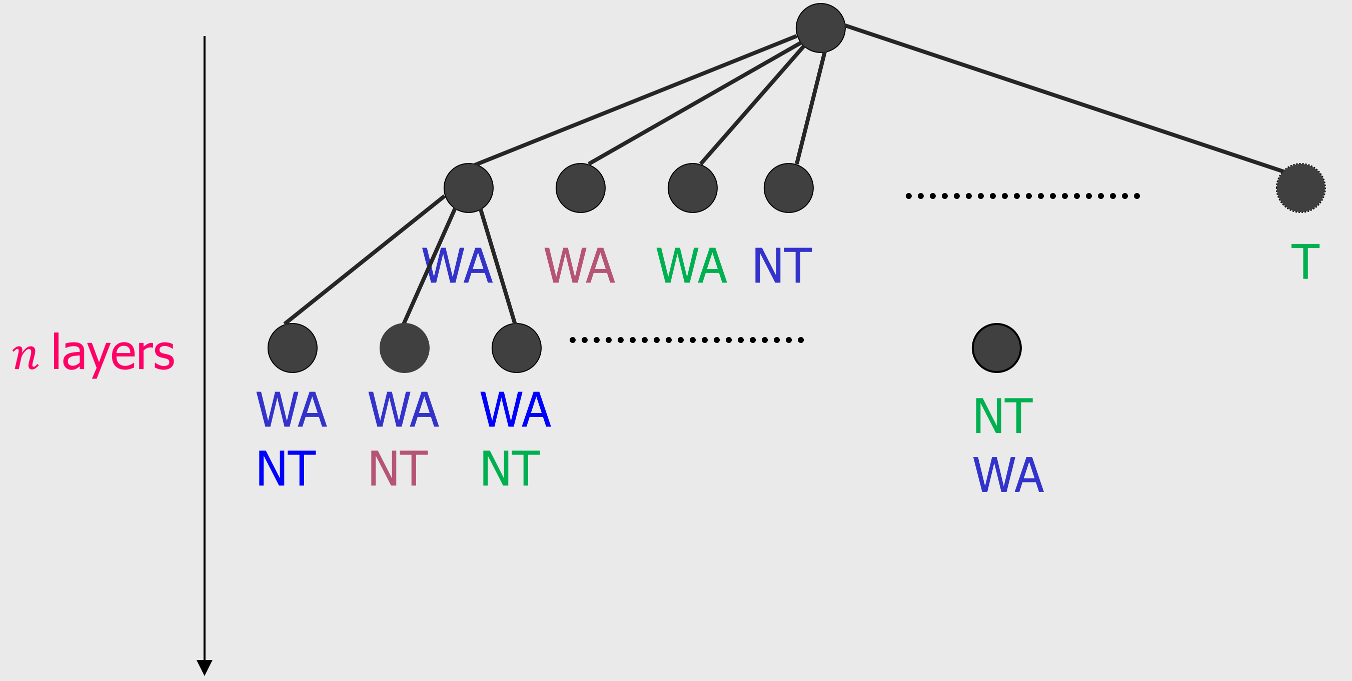


NT WA same as WA NT

That's Good!

Backtracking (Depth-First) search

Backtracking search uses **depth first search** that chooses **value** for one **variable** at time and **backtracks** when no legal value left.

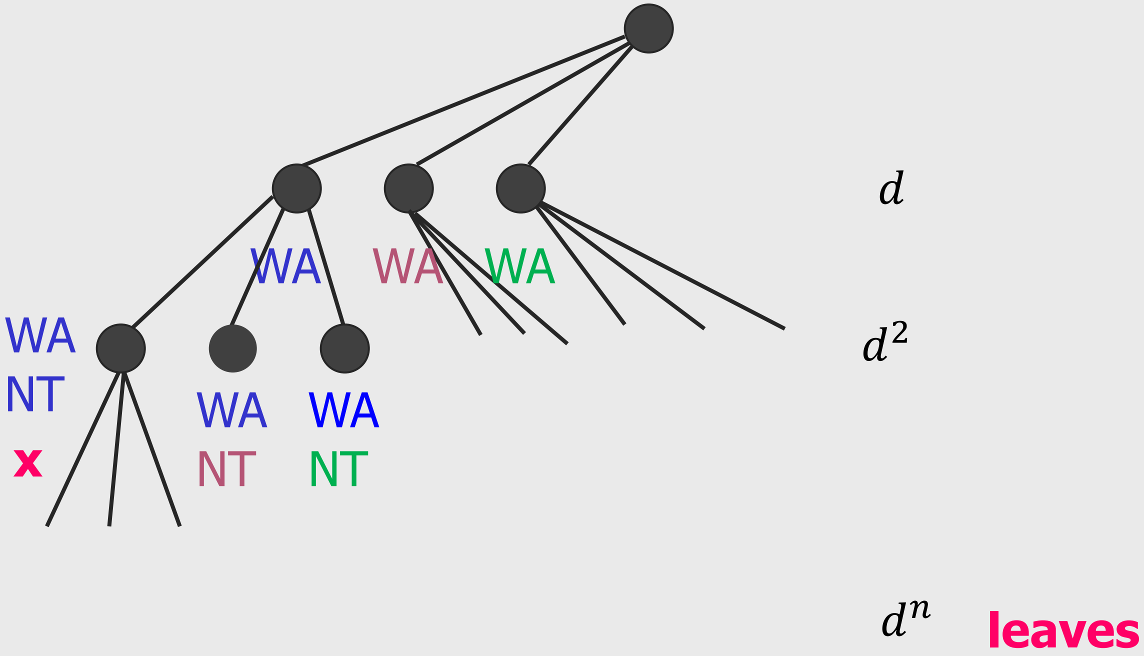


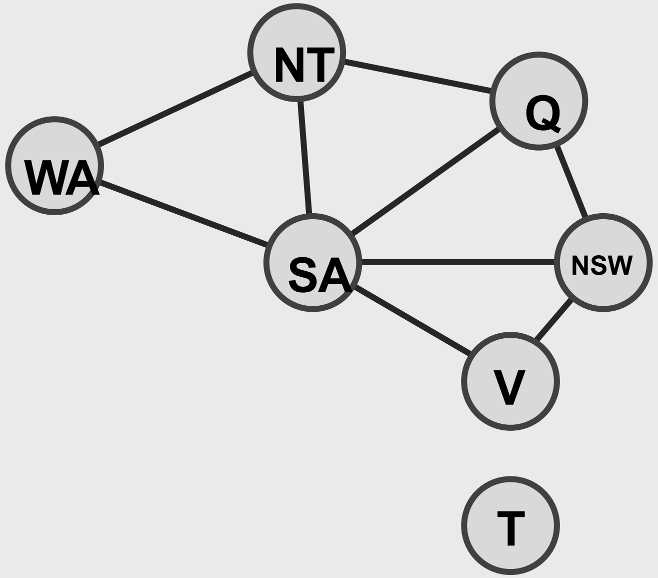
$n \times d$

$(n \times d) \times (n-1) \times d$

$n! \times d^n$ **leaves**

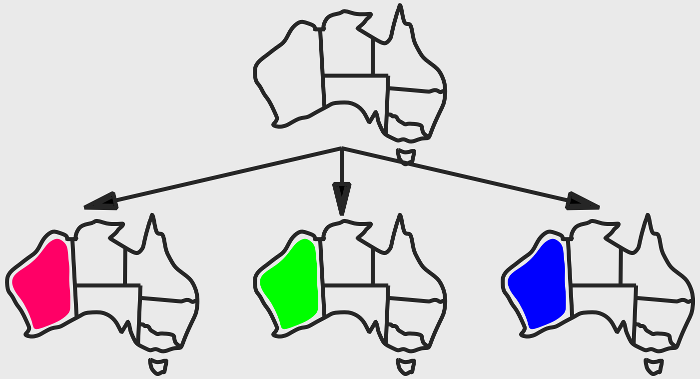
Backtracking search



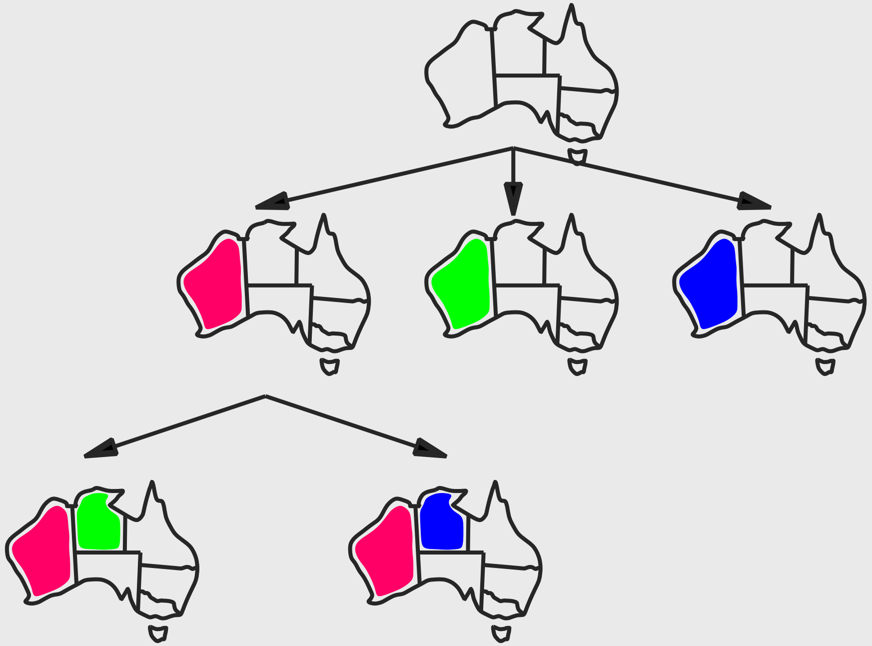


Backtracking search

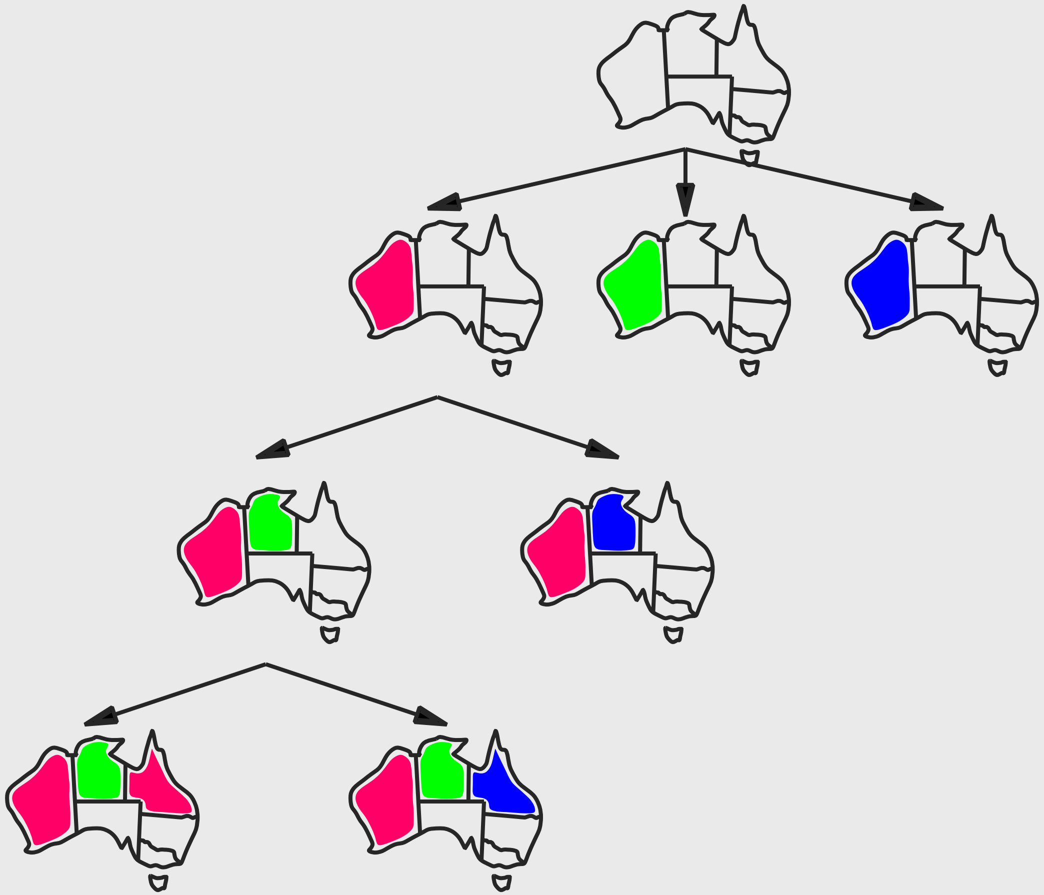
Backtracking search



Backtracking search



Backtracking search



Backtracking Search (efficiency Improvement)

General-purpose methods can give huge gains in speed:

1. Which variable should be assigned next?
2. In what order should its values be tried?
3. Can we detect inevitable failure early?
4. Can we take advantage of problem structure?

Variable and Value Ordering

1 **2**

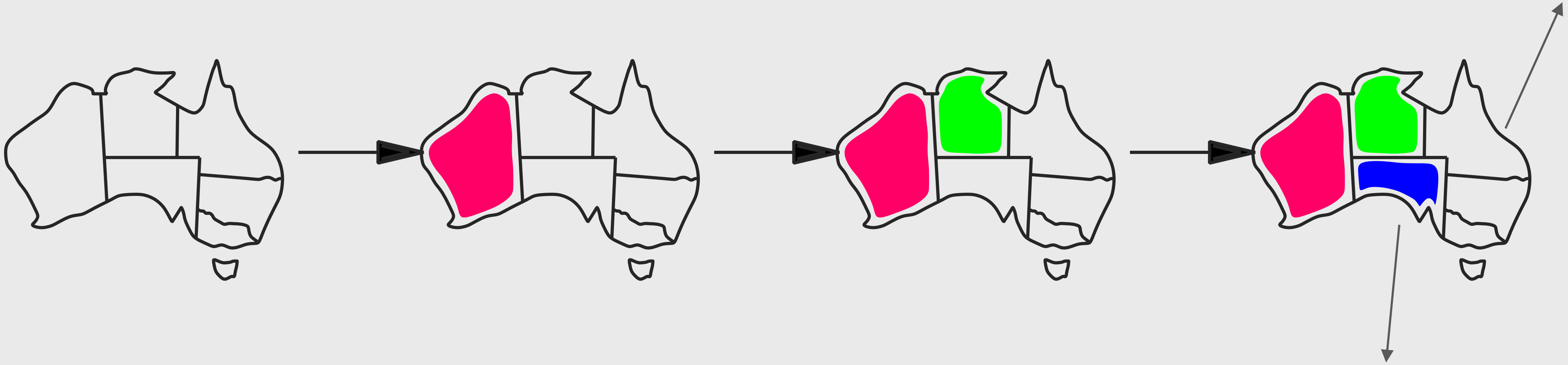
Which variable should be assigned next?

And

In what order should its values be tried?

Minimum remaining values heuristic

MRV: choose the variable with the fewest legal values

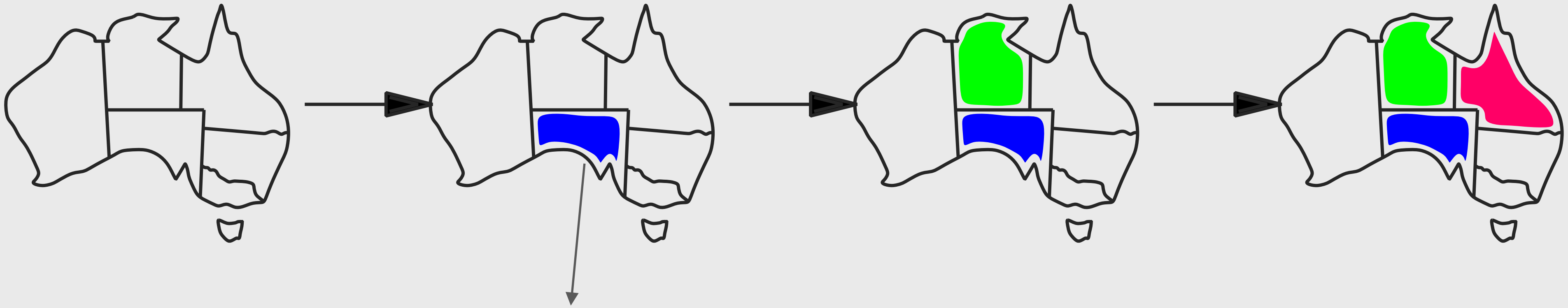


Q has two values left: blue and red

SA has only blue value left

Degree heuristic

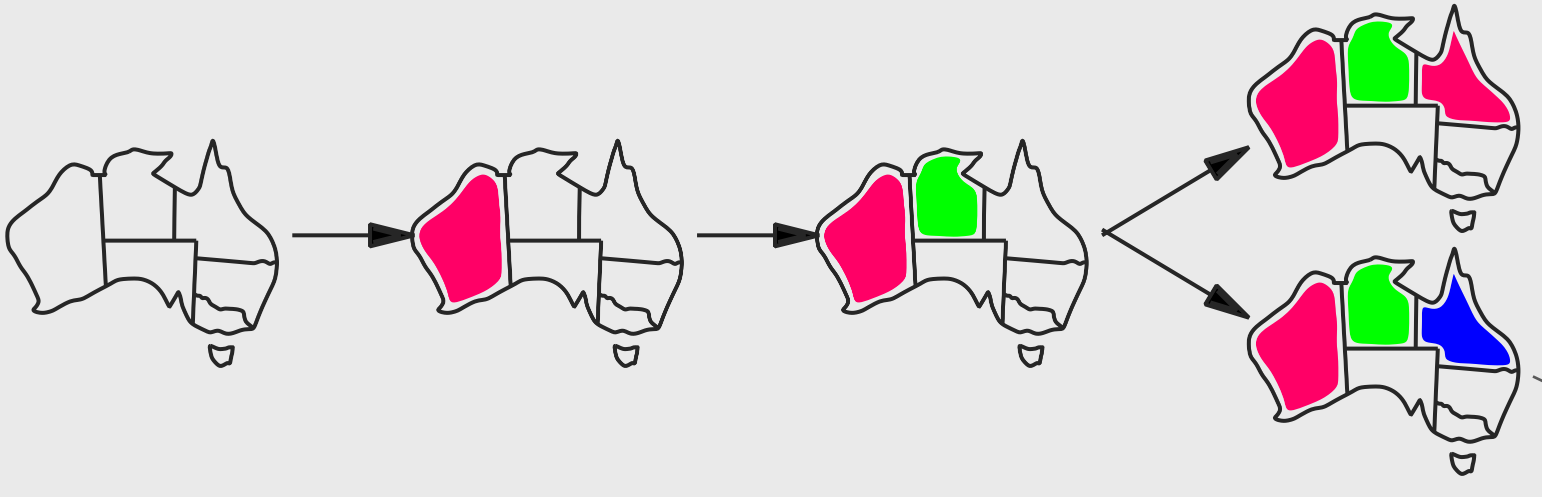
Picks a variable which will cause failure as soon as possible, allowing the tree to be pruned. (i.e., choose the variable with the most constraints on remaining variables)



If SA the most neighbour so
it will restrict others

Least constraining value heuristic

- Given a variable, choose the least constraining value:
- the one that rules out the fewest values in the remaining variables



Allows 1 value for SA

Allows 0 values for SA

Blue would be bad choice since it will constrain its neighbour most

Rationale for MRV, DH, LCV

- In all cases we want to enter the most promising branch, but we also want to **detect inevitable failure as soon as possible**.

- **MRV + DH:** the variable that is most likely to cause failure in a branch is assigned first. The variable must be assigned at some point, so **if it is doomed to fail, we would better found out soon**.

- **LCV:** tries to **avoid failure by assigning values that leave maximal flexibility** for the remaining variables. We want our search to succeed as soon as possible, so given some ordering, we want to find the successful branch.

③

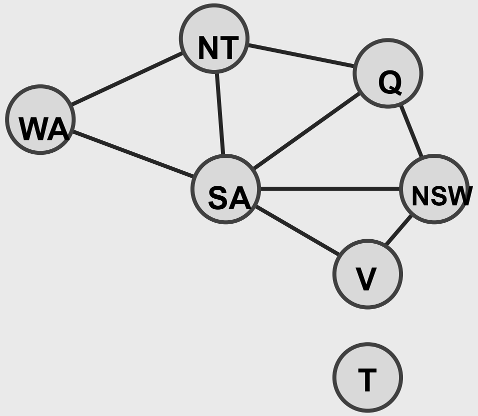
Can we detect inevitable failure early?

5:53 pm

Forward Checking

Idea: Keep track of remaining legal values for unassigned variables that are connected to current variable.

Terminate search when any variable has no legal values

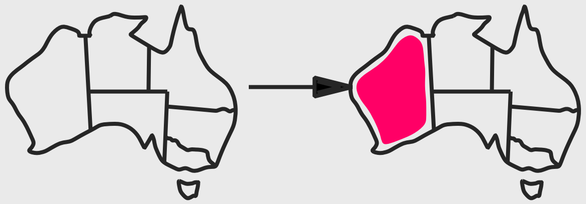


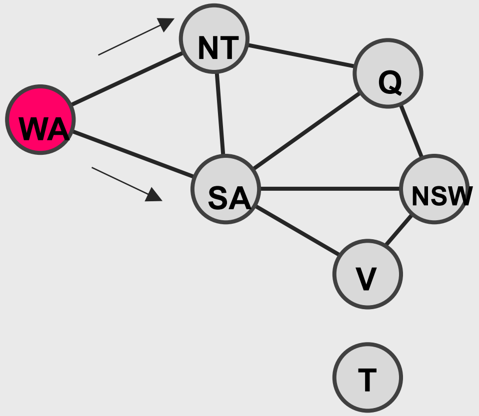
WA	NT	Q	NWS	V	SA	T
🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦

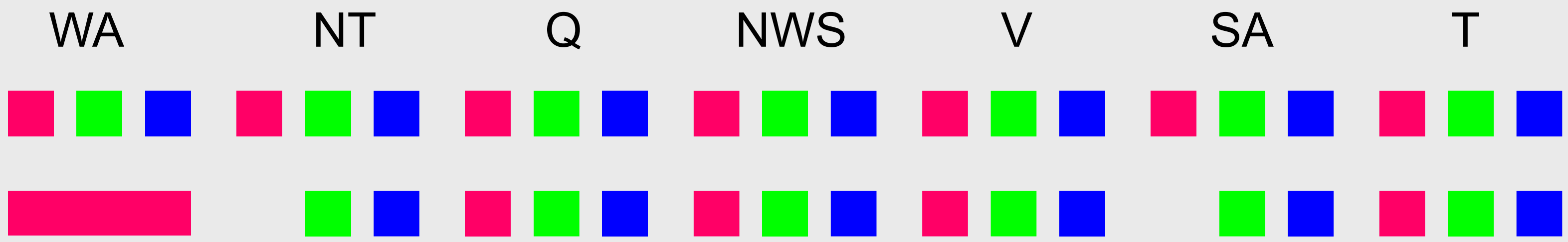
Forward Checking

Idea: Keep track of remaining legal values for unassigned variables that are connected to current variable.

Terminate search when any variable has no legal values







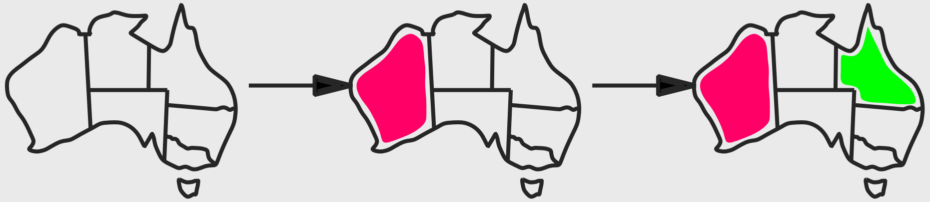
WA	NT	Q	NWS	V	SA	T
🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦
🟥	🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟩 🟦	🟥 🟩 🟦

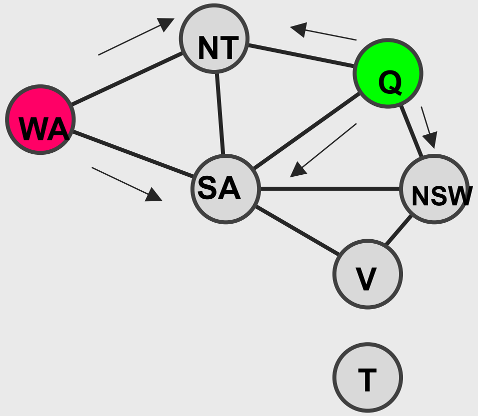
5:53 pm

Forward Checking

Idea: Keep track of remaining legal values for unassigned variables that are connected to current variable.

Terminate search when any variable has no legal values





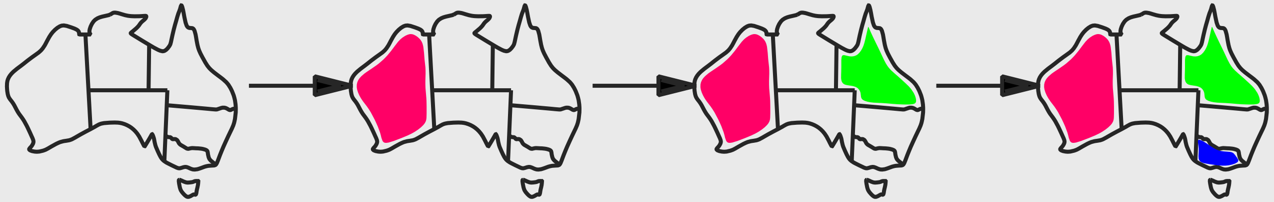
WA	NT	Q	NWS	V	SA	T
🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦
🟥	🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟩 🟦	🟥 🟩 🟦
🟥	🟦	🟩	🟥 🟦	🟥 🟩 🟦	🟦	🟥 🟩 🟦

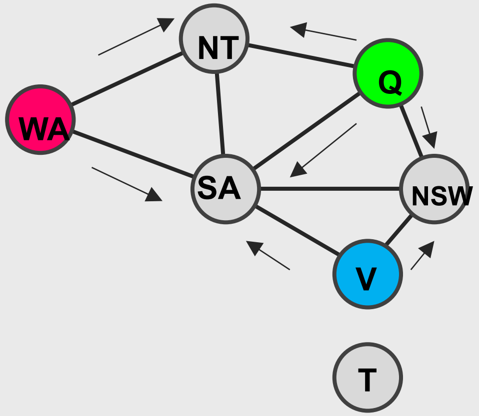
5:53 pm

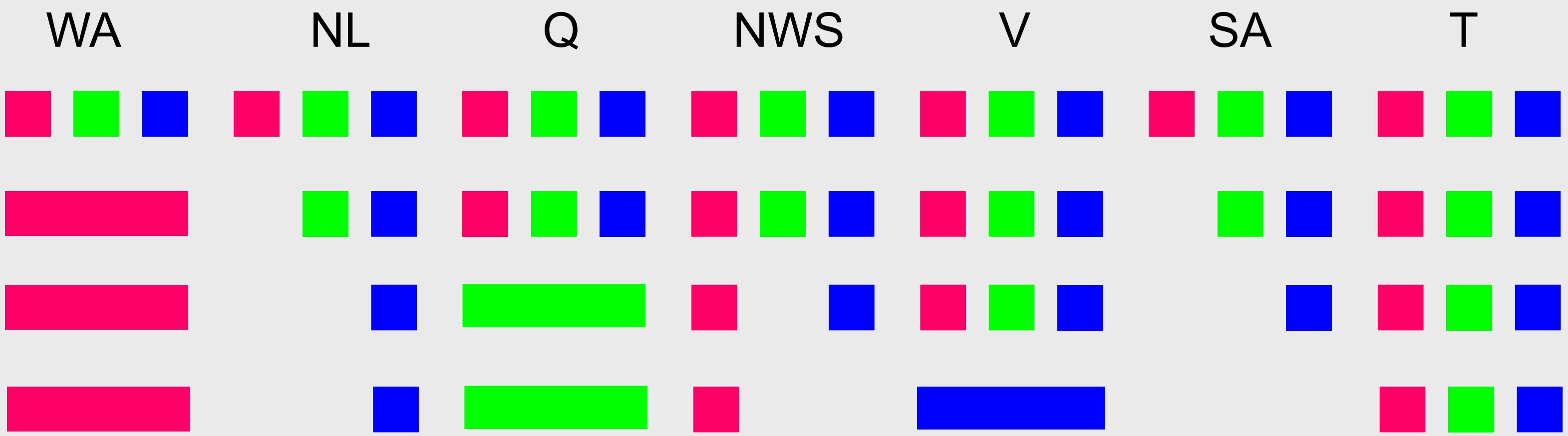
Forward Checking

Idea: Keep track of remaining legal values for unassigned variables that are connected to current variable.

Terminate search when any variable has no legal values



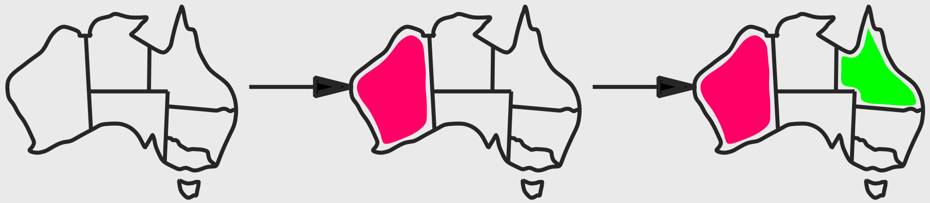


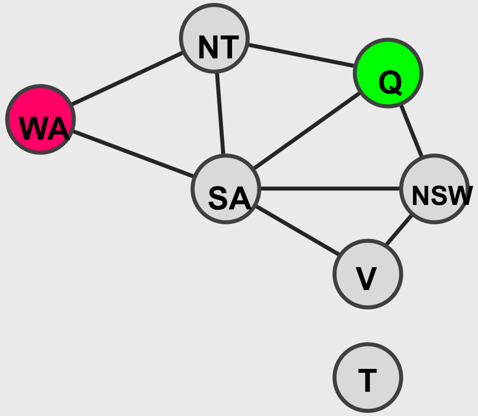


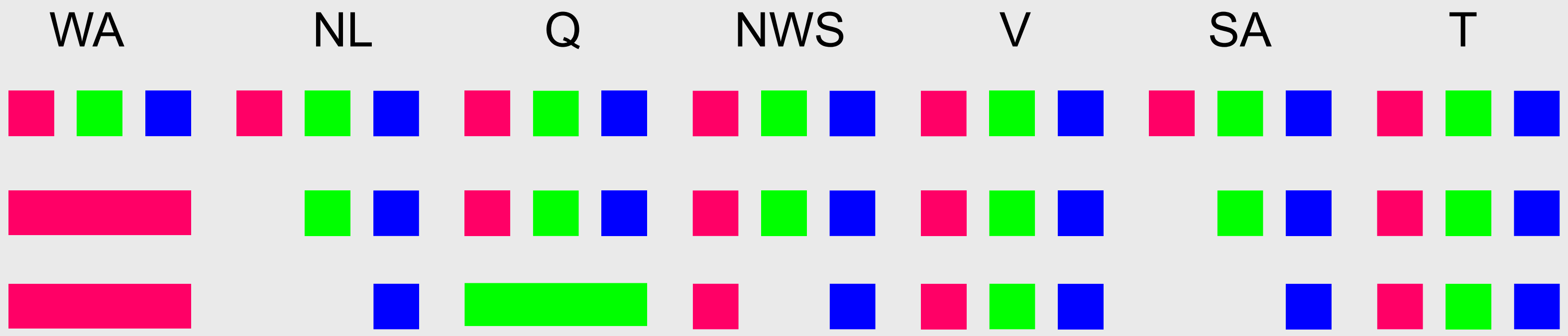
WA	NL	Q	NWS	V	SA	T
🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦
🟥	🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟥 🟩 🟦	🟩 🟦	🟥 🟩 🟦
🟥	🟦	🟩	🟥 🟦	🟥 🟩 🟦	🟦	🟥 🟩 🟦
🟥	🟦	🟩	🟥	🟦		🟥 🟩 🟦

Constraint propagation

Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:







WA	NL	Q	NWS	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦

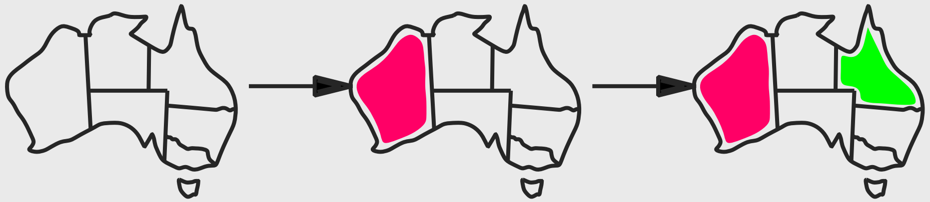
NT and **SA** cannot both be blue!

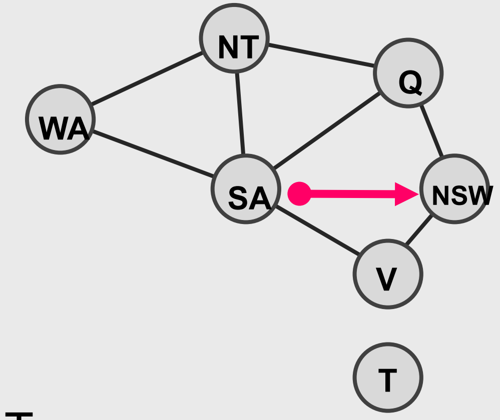
Constraint propagation repeatedly enforces constraints locally

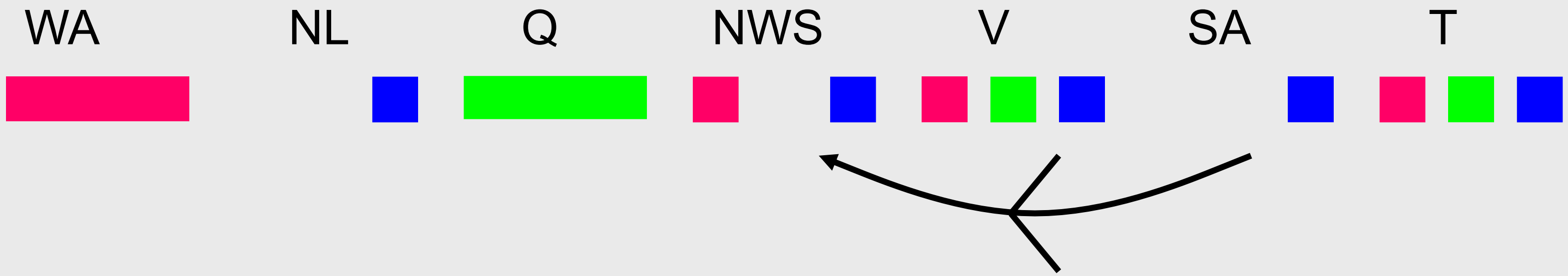
Arc Consistency

Simplest form of propagation makes each arc **consistent**

$X \rightarrow Y$ is consistent iff for **every** value
$x \in X$ there is **some** allowed y



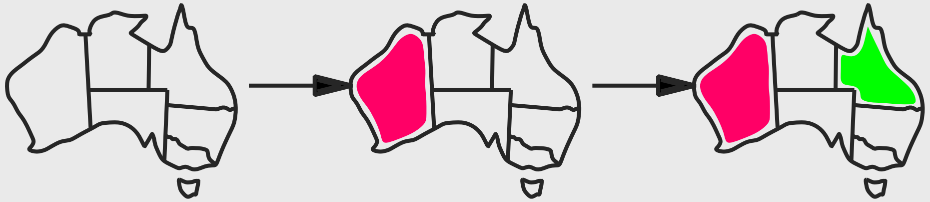


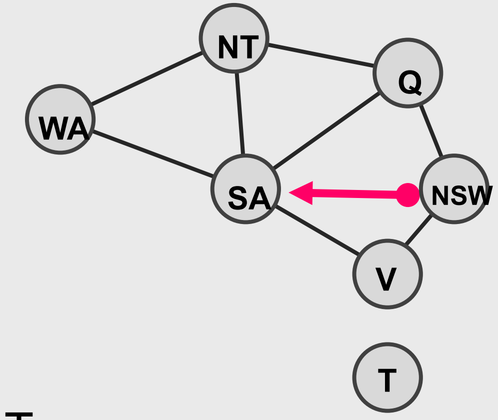


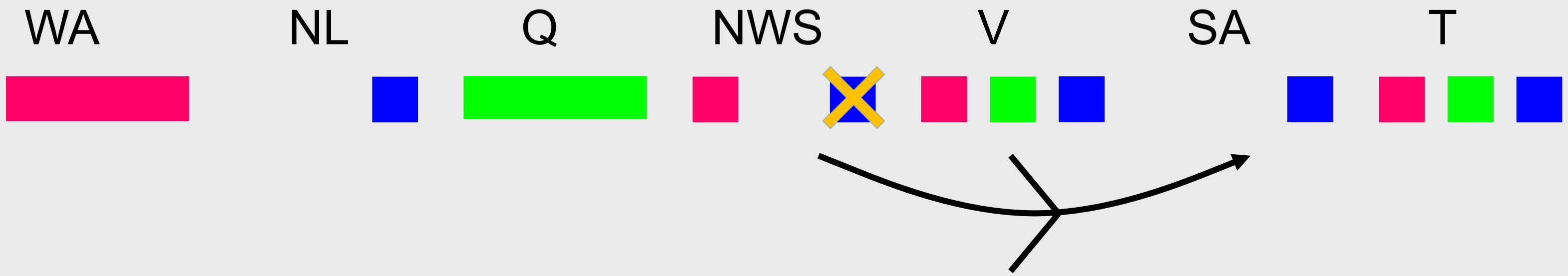
Arc Consistency

Simplest form of propagation makes each arc **consistent**

$X \rightarrow Y$ is consistent iff for **every** value
$x \in X$ there is **some** allowed y





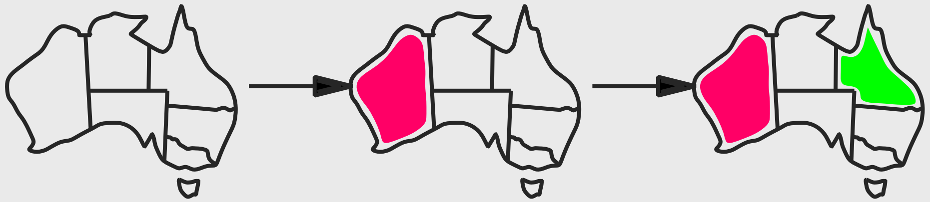


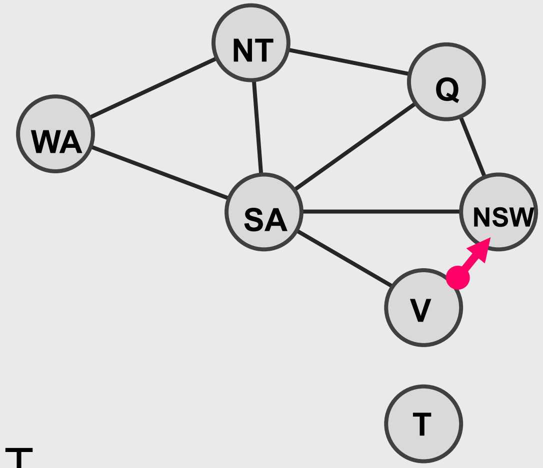
inconsistent arc
remove blue from source consistent arc.

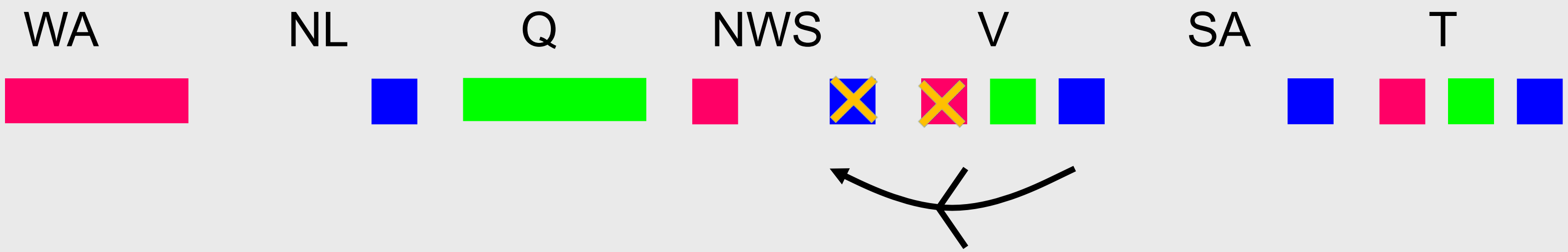
Arc Consistency

Simplest form of propagation makes each arc **consistent**

If X loses a value, neighbours of X need to be rechecked





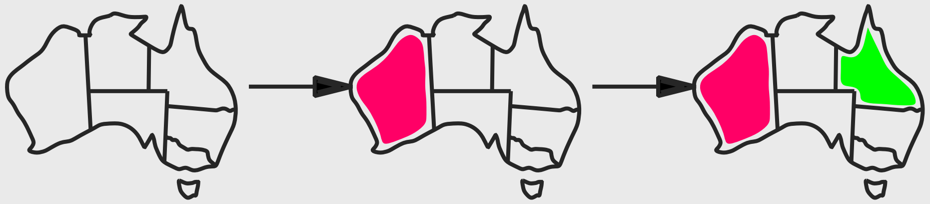


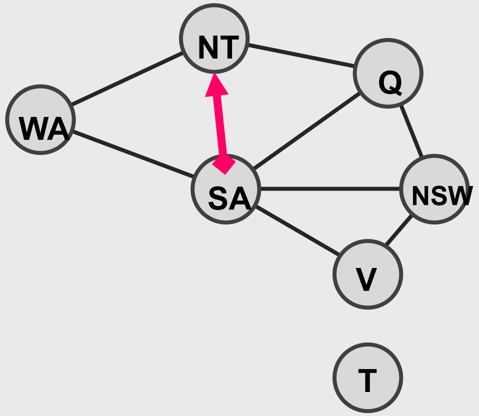
neighbours of this arc just became inconsistent

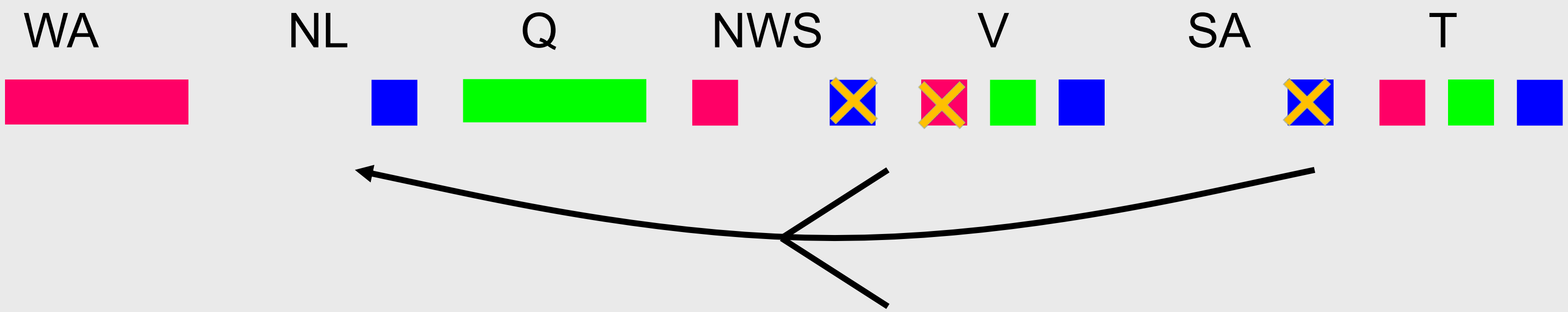
Arc Consistency

Simplest form of propagation makes each arc **consistent**

If X loses a value, neighbours of X need to be rechecked

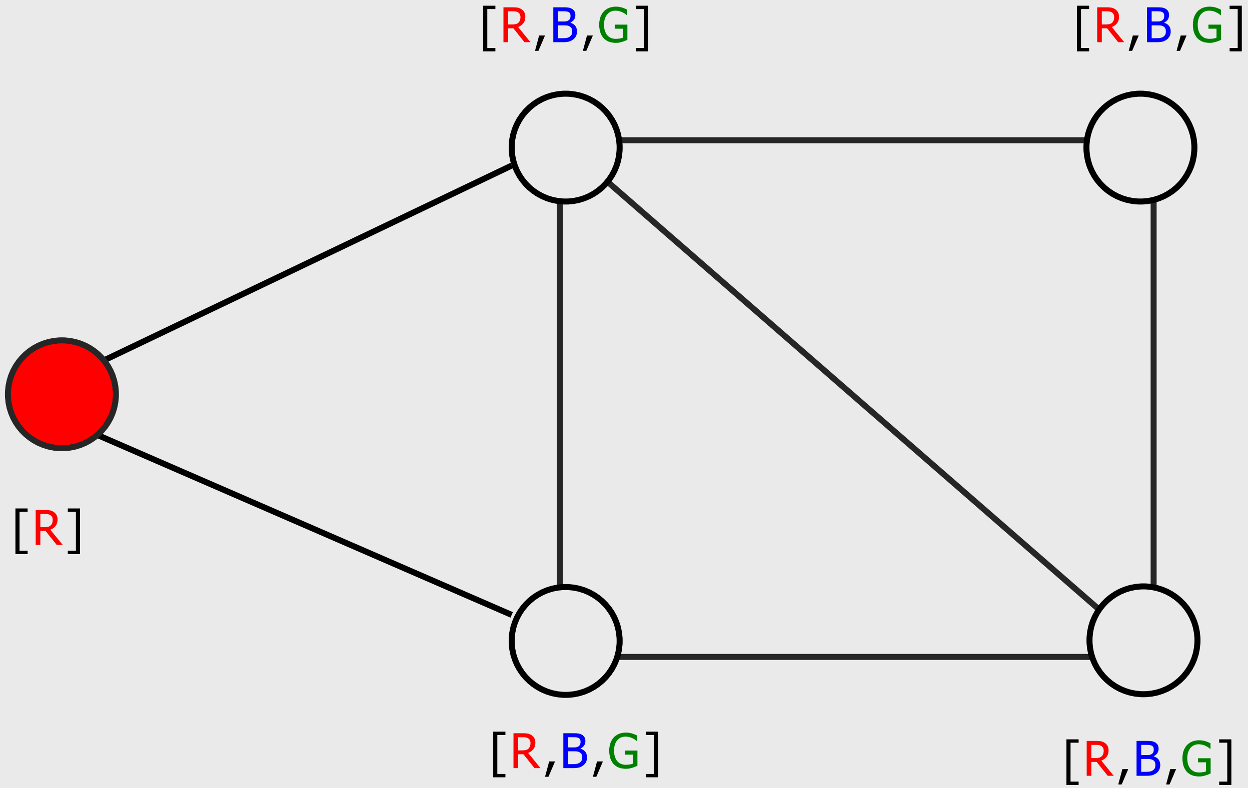






WA NL Q NWS V SA T

Task



2 minutes/ Home work

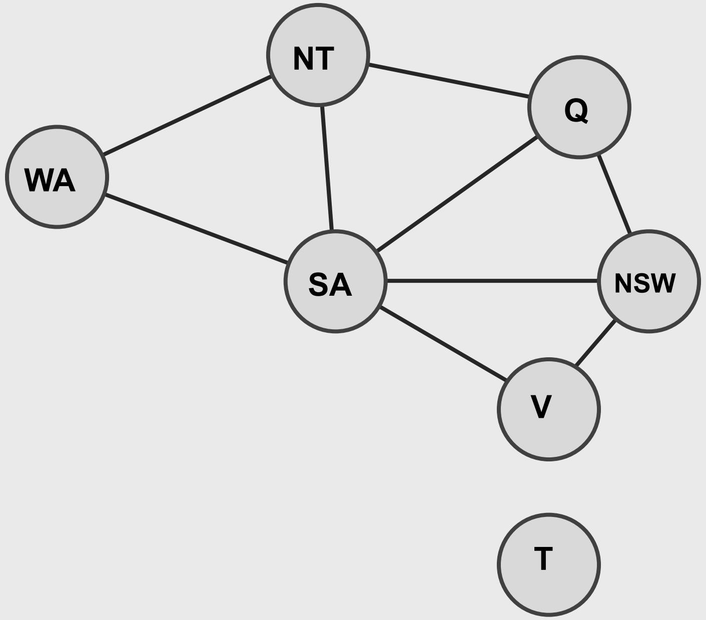
START STOP

④

Can we take advantage of problem structure?

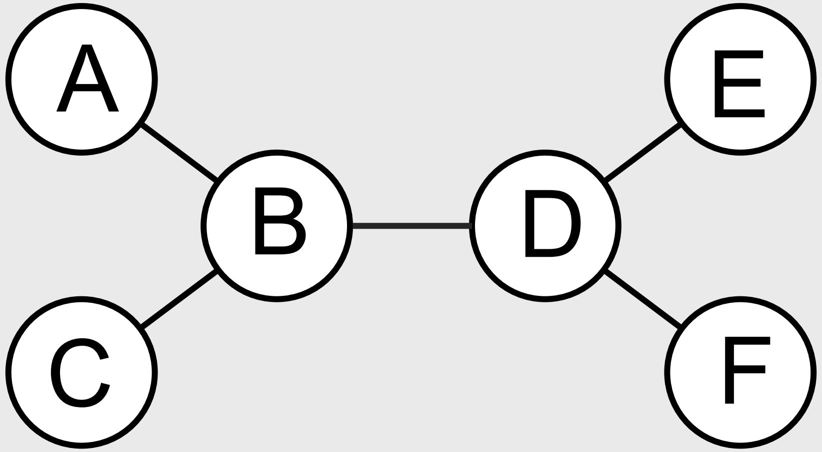
Problem Structure

$O(n^2d^3)$, can be reduced to $O(n^2d^2)$ (but detecting **all** is **NP-hard**)



Tasmania and mainland are **independent subproblems**

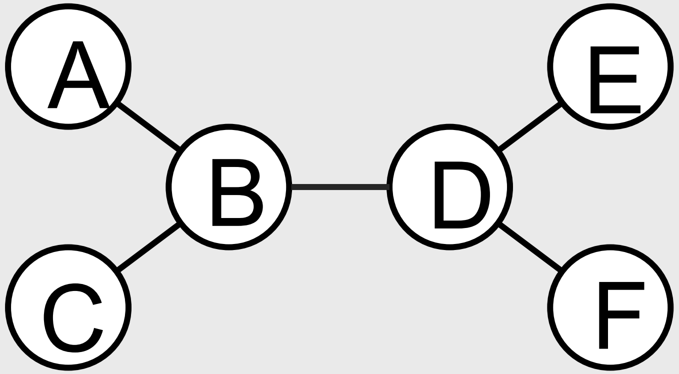
Tree-structured CSPs

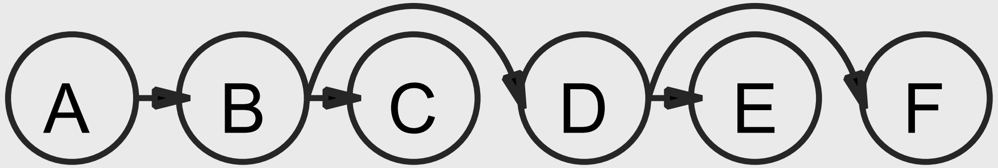


Theorem: if the **constraint graph** has no loops, the CSP can be solved in $\boldsymbol{O(nd^2)}$ time.

General CSPs has **worst-case time** is $\boldsymbol{O(d^n)}$

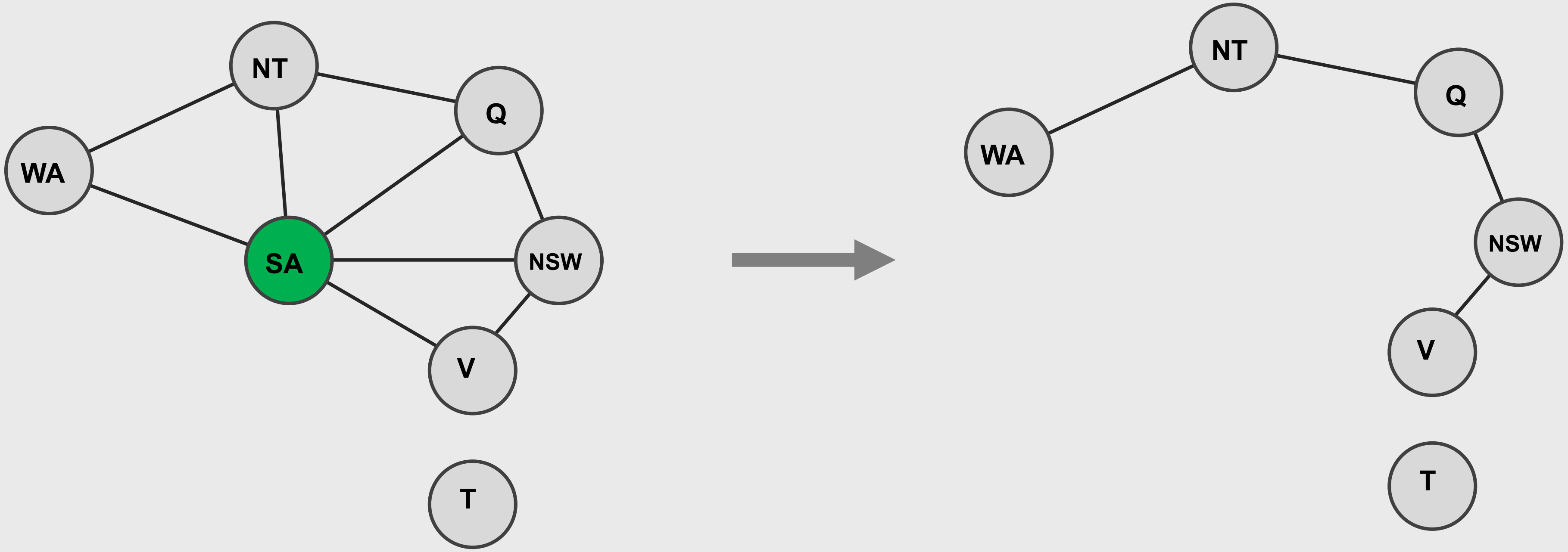
Tree-structured CSPs: Algorithm





1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering
2. For *j* from *n* down to 2, apply RemoveInconsistent(*Parent*(X_j),X_j)
3. For *j* from 1 to *n*, assign X_j consistently with *Parent*(X_j)

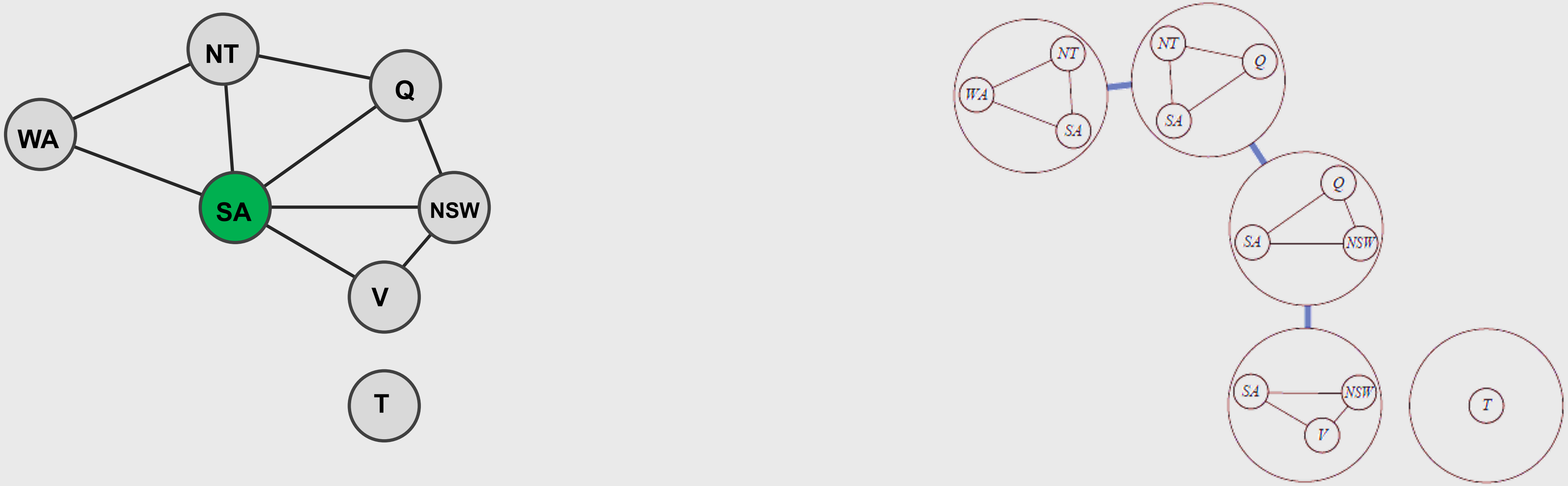
Nearly Tree-structured CSPs



1. Choose a variable as root, order variables from root to leaves such that every node’s parent precedes it in the ordering
2. For *j* from *n* down to 2, apply RemoveInconsistent(*Parent*(X_j),X_j)
3. For *j* from 1 to *n*, assign $\boldsymbol{X_j}$ consistently with *Parent*($\boldsymbol{X_j}$)

Nearly Tree-structured CSPs

Tree decomposition of the constraint graph into a set of connected subproblems

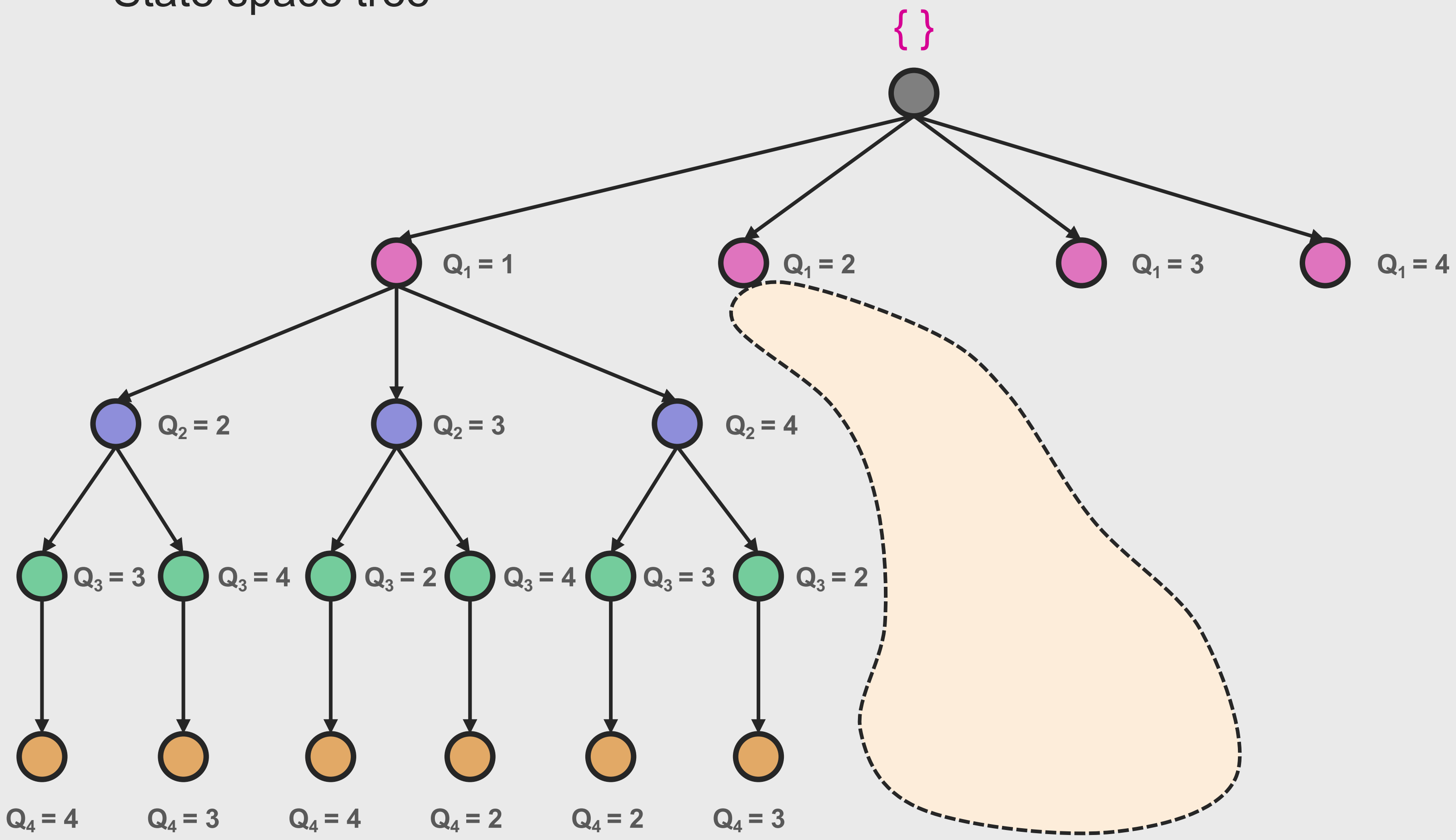


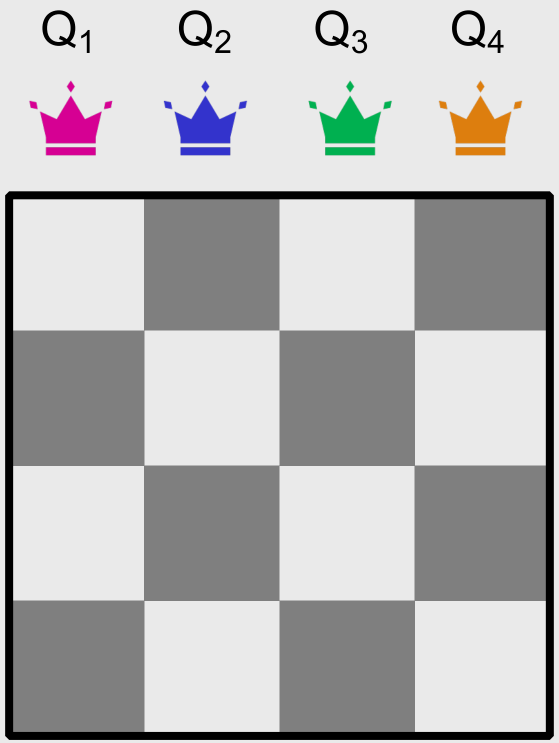
- Every variable in the original problem appears in at least one of the subproblems.
- If two variables are connected by a constraint in the original problem, they must appear together (along with the constraint) in at least one of the subproblems.
- If a variable appears in two subproblems in the tree, it must appear in every subproblemalong the path connecting those subproblems.

Part 4
Heuristics Search

Solving 4-Queen Problem

State space tree

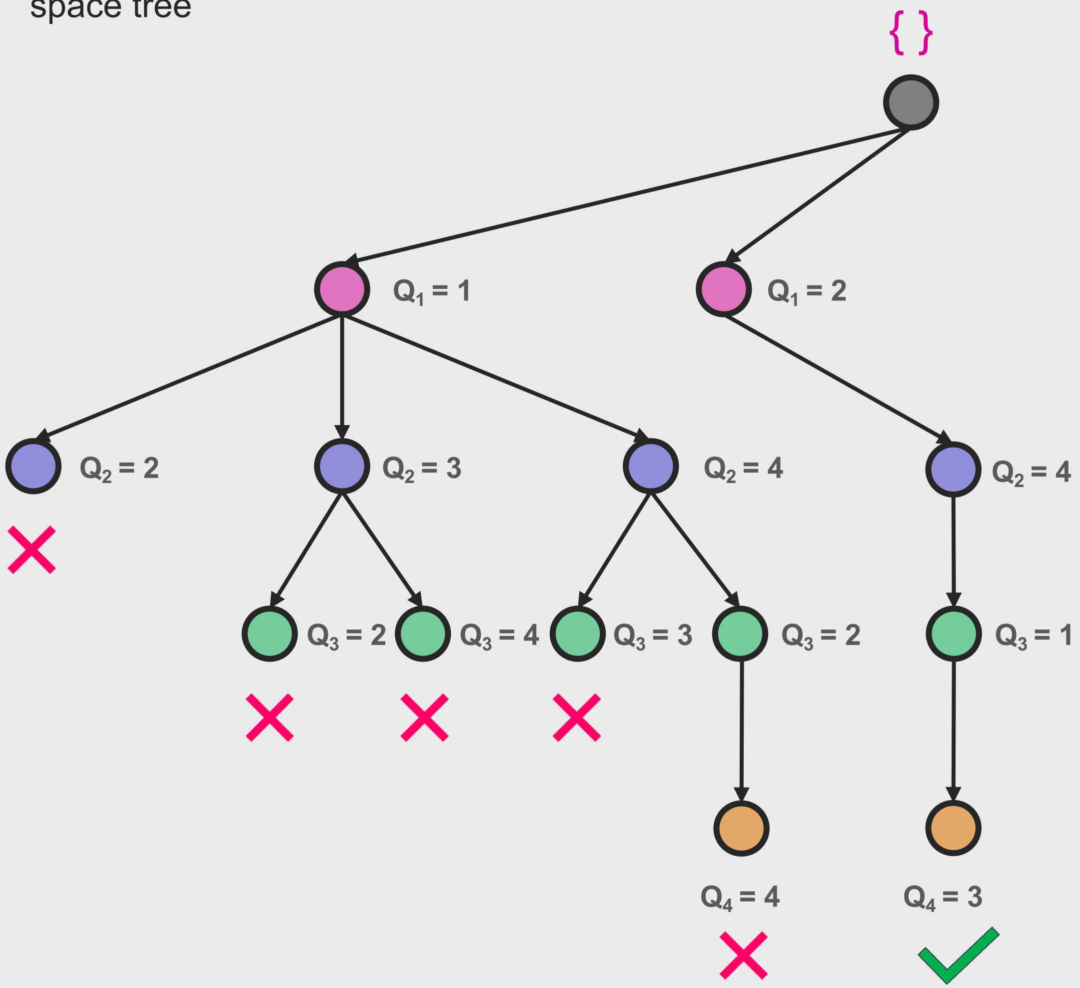


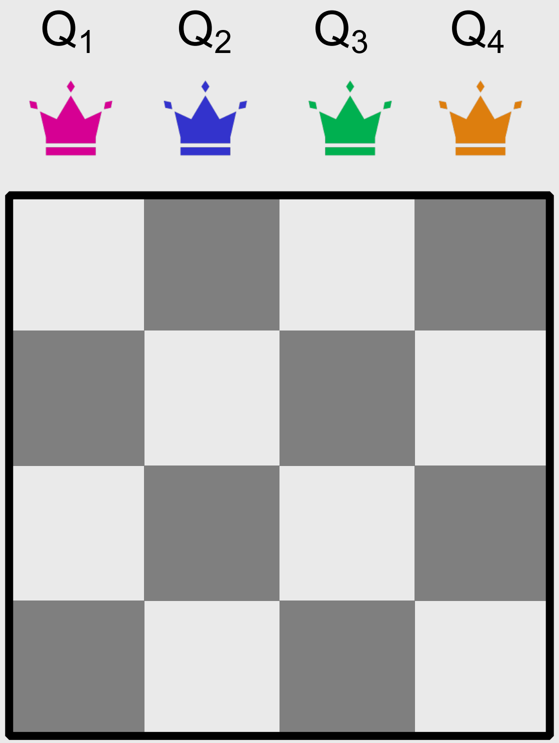


5:53 pm

Solving 4-Queen Problem

Backtracking State
space tree





Hill Climbing

Image source: https://images.app.goo.gl/bePciTN8FsQJm37L9

Simulated Annealing

Iterative algorithms for CSPs

Hill-climbing, **Simulated Annealing** typically works with “complete” states, i.e., all variables assigned

To apply these algorithm to CSPs:

- allow states with unsatisfied constraints
- operators **reassign** variable values

Variable selection: randomly select any conflicted variable

Value selection by min-conflicts heuristic:

- choose value that violates the fewest constraints
- i.e., hillclimb with $h(n)$ = total number of violated constraints

Iterative algorithms for CSPs: Example

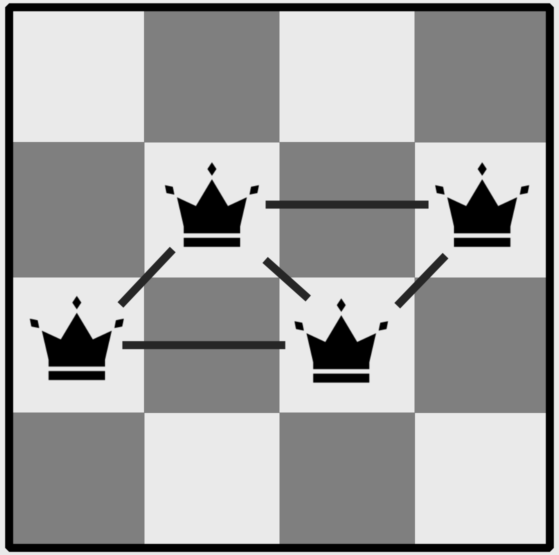
4-Queen problem

States: 4 queens in 4 columns ($4^4 = 256$ states)

Operators: move queen in column

Goal test: no attacks

Evaluation: $h(n)$ = number of attacks



$h(n) = 5$

Iterative algorithms for CSPs: Example

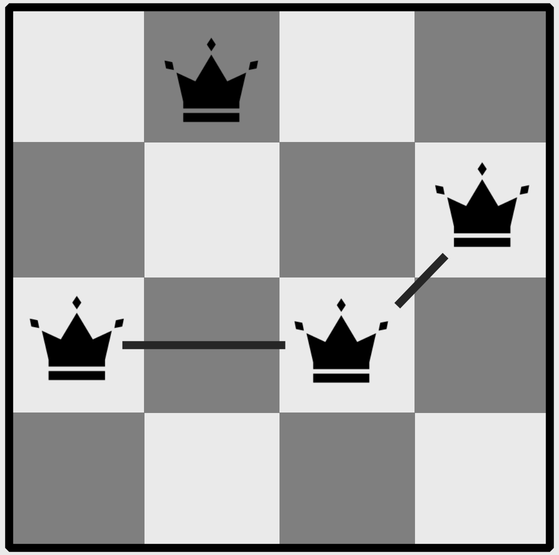
4-Queen problem

States: 4 queens in 4 columns (4^4 = 256 states)

Operators: move queen in column

Goal test: no attacks

Evaluation: *h*(*n*) = number of attacks



h(*n*) = 2

Iterative algorithms for CSPs: Example

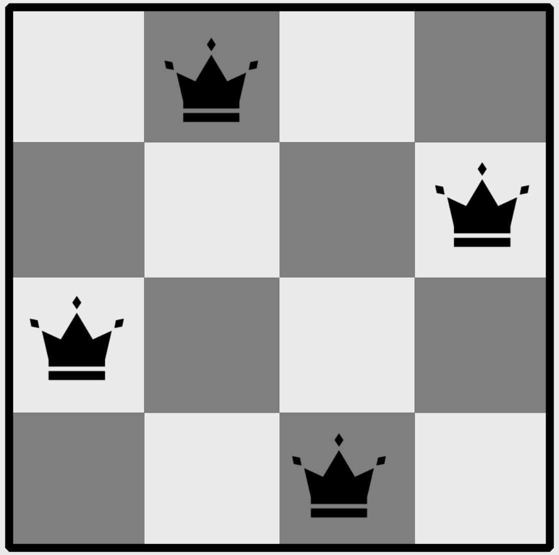
4-Queen problem

States: **4** queens in **4** columns (**4^4 = 256** states)

Operators: move queen in column

Goal test: no attacks

Evaluation: ***h*(*n*)** = number of attacks



***h*(*n*) = 0**

Solve!