CONSTRAINT SATISFACTION PROBLEMS

ARTIFICIAL INTELLIGENCE | COMP 131

TODAY ON AI

- Constraint Satisfaction Problems
- Solving CSPs
- Filtering
- Variable ordering
- Value ordering
- Smart backtracking
- Problem structure
- Questions?

SECTION 01

Constraint Satisfaction Problems

Constraint satisfaction problems (or **CSPs**) belong to a class of problems for which the goal itself is the most important part, not the path used to reach it.

EXAMPLES

- Map coloring!
- Sudokus
- Crossword puzzles
- Job scheduling
- Cryptarithmetic puzzles
- N-Queens problems
- Hardware configuration
- Assignment problems
- Transportation scheduling
- Fault diagnosis
- More…

The state of a CSP is defined by n **variables X_i** with values from **domain D_i**:

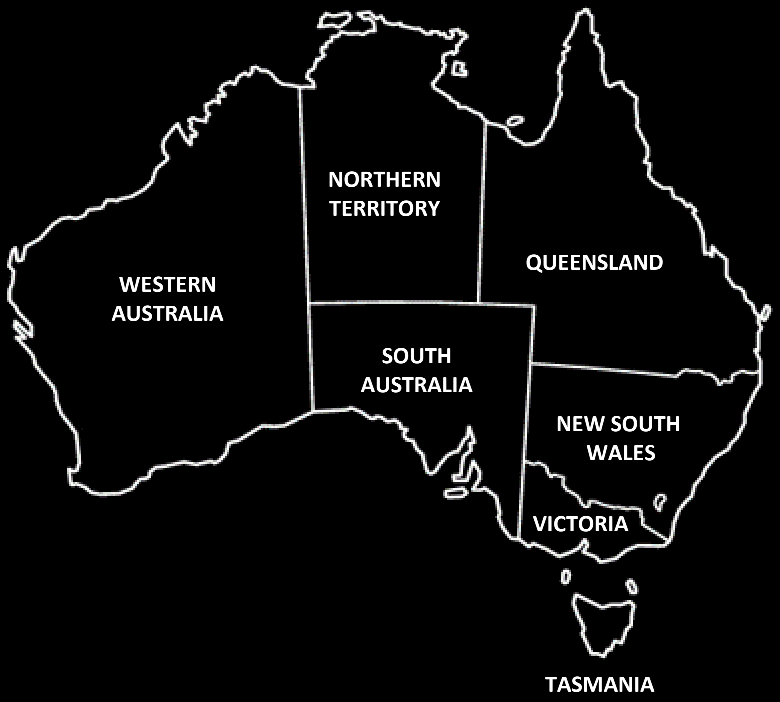
- Discrete variables:
  - Domains can be **finite**: a finite of size d set of values or things (means d^n complete assignments). Examples: Boolean values, specific meaningful numbers, set of colors, etc.
  - or **infinite**: integers or strings. Examples: strings for a crossword puzzle, duration of jobs in seconds, etc.

- Continuous variables:
  - Domains are **infinite**. Examples are: start/end times for Hubble Telescope observations as they obey to astronomical time laws

- The goal test is a set of **constraints** that specifies allowable combinations of values for subsets of variables:
  - Constraints can be **explicit** (explicitly enumerated)
  - or **implicit** (a formula describes it)
  - Constraints can be **unary**, **binary**, **global**, **alldiff**

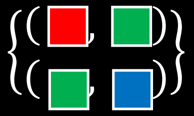
- Constraints are generally represented with a graph, called **hypergraph**, that shows the relationship between the variables.

- **Soft constraints** represent preferences about some values of the variables. They usually come with a cost value that expresses the strength of the preference.

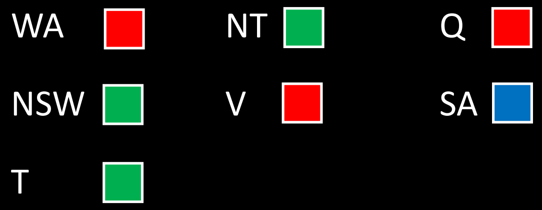


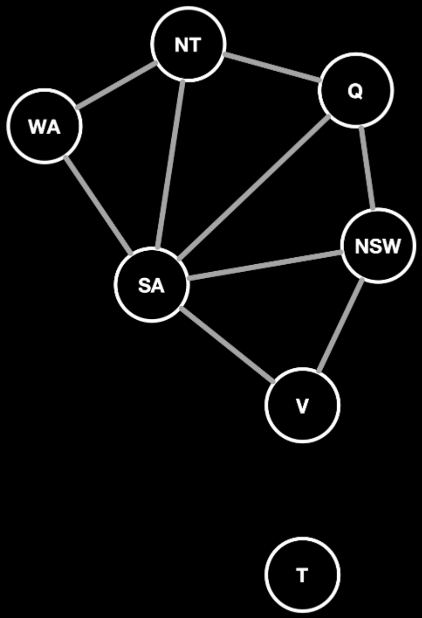
- **VARIABLES**
  WA, NT, Q, NSW, V, SA, T
- **DOMAINS**
  
- **CONSTRAINTS**
  Adjacent regions must have different colors:

  Implicit: $WA \neq SA$, $WA \neq NT$, $NT \neq SA$, $NT \neq Q$, $Q \neq SA$, $Q \neq NSW$, $NSW \neq SA$, $NSW \neq V$, $V \neq SA$

  Explicit: $(WA, NT) \in$ 
  etc.

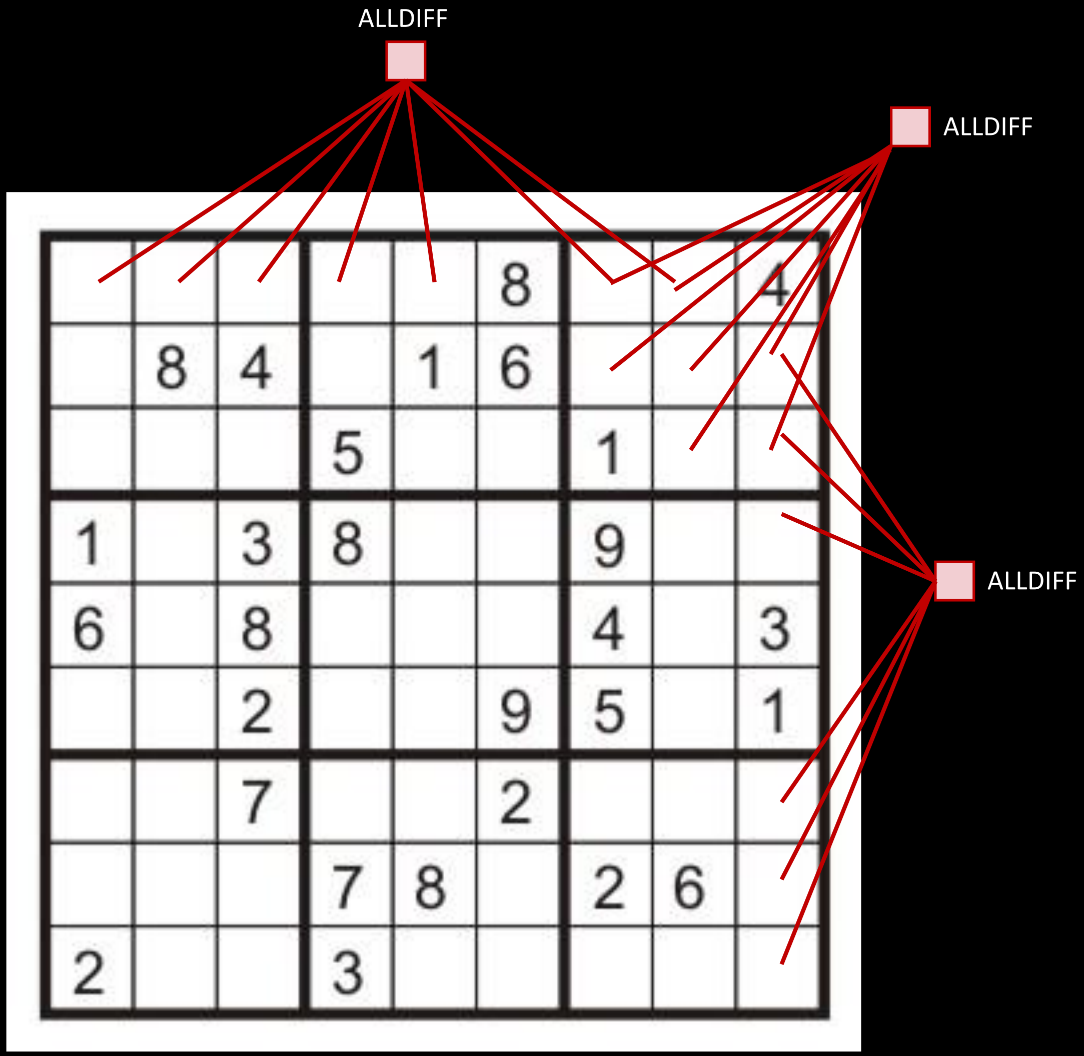
- Solutions are assignments that satisfying all constraints:





Constraint Satisfaction problems (CSPs)

EXAMPLE: COLORING AUSTRALIA



- **VARIABLES**
  Open squares
- **DOMAINS**
  {1, 2, 3, … 9}
- **CONSTRAINTS**

  9-way alldiff for each column
  9-way alldiff for each row
  9-way alldiff for each region
- **RULES**

  Fill all empty squares so that the numbers 1 to 9 appear once in each row, column and 3x3 box.

SECTION 02

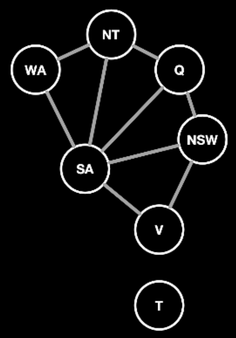
Solving CSPs

The idea is to use standard search algorithms (DFS and BFS) to find a solution that satisfies all the constraints.

- **STATES**
  The variables assigned with values so far
- **INITIAL STATE**
  All variable assignments are empty
- **POSSIBLE ACTIONS**
  Variable assignment
- **SUCCESSOR FUNCTION**
  All possible assignments
- **GOAL TEST**
  The current assignment is complete and satisfies all constraints

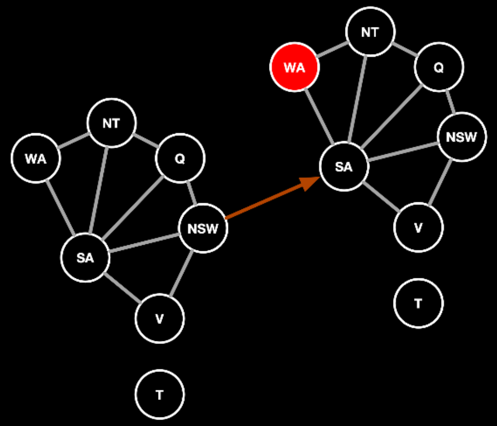
Chronological backtracking search is an uninformed searching algorithm based on the Depth-first searching algorithm with some improvements related to CSPs.

- **IMPROVEMENT 1**
  Each step considers only one assignment at the time
- **IMPROVEMENT 2**
  Check constraints as the search continues. Consider only new assignments which do not conflict previous assignments (incremental goal test)



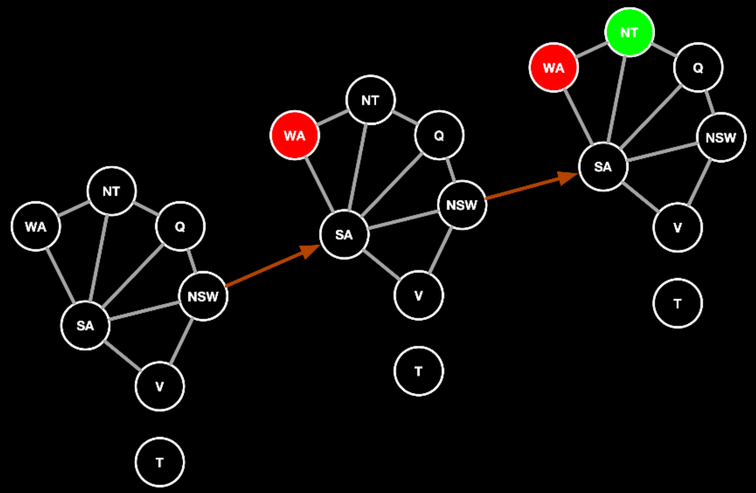
Backtracking search

COLORING EXAMPLE



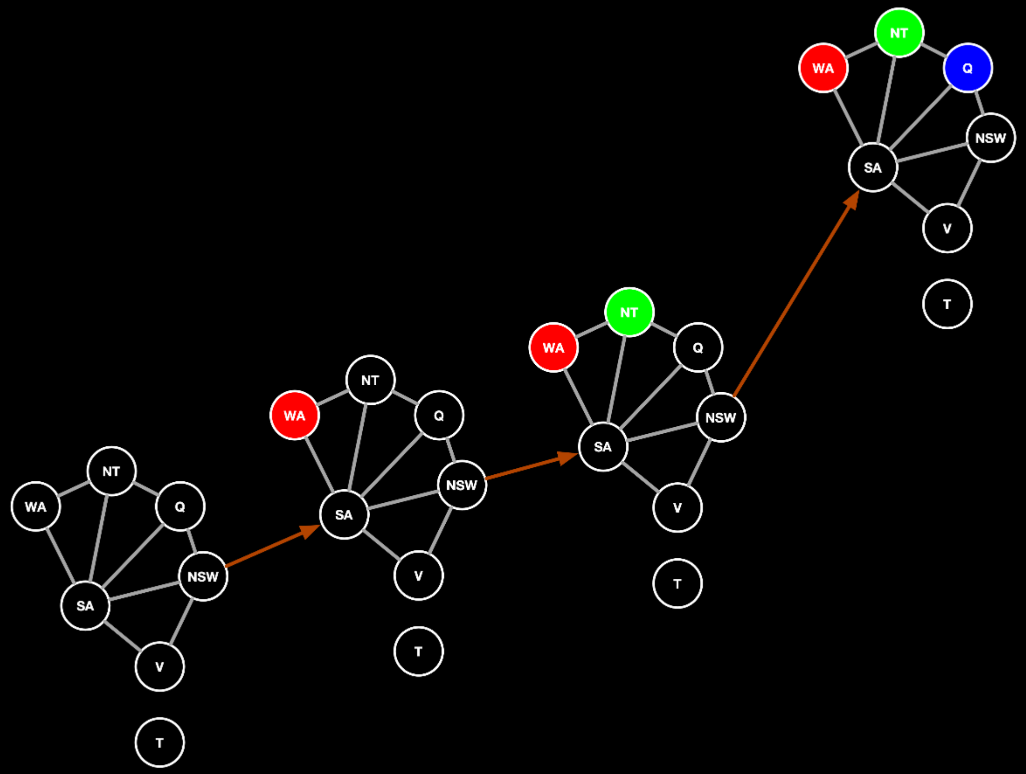
Backtracking search

COLORING EXAMPLE



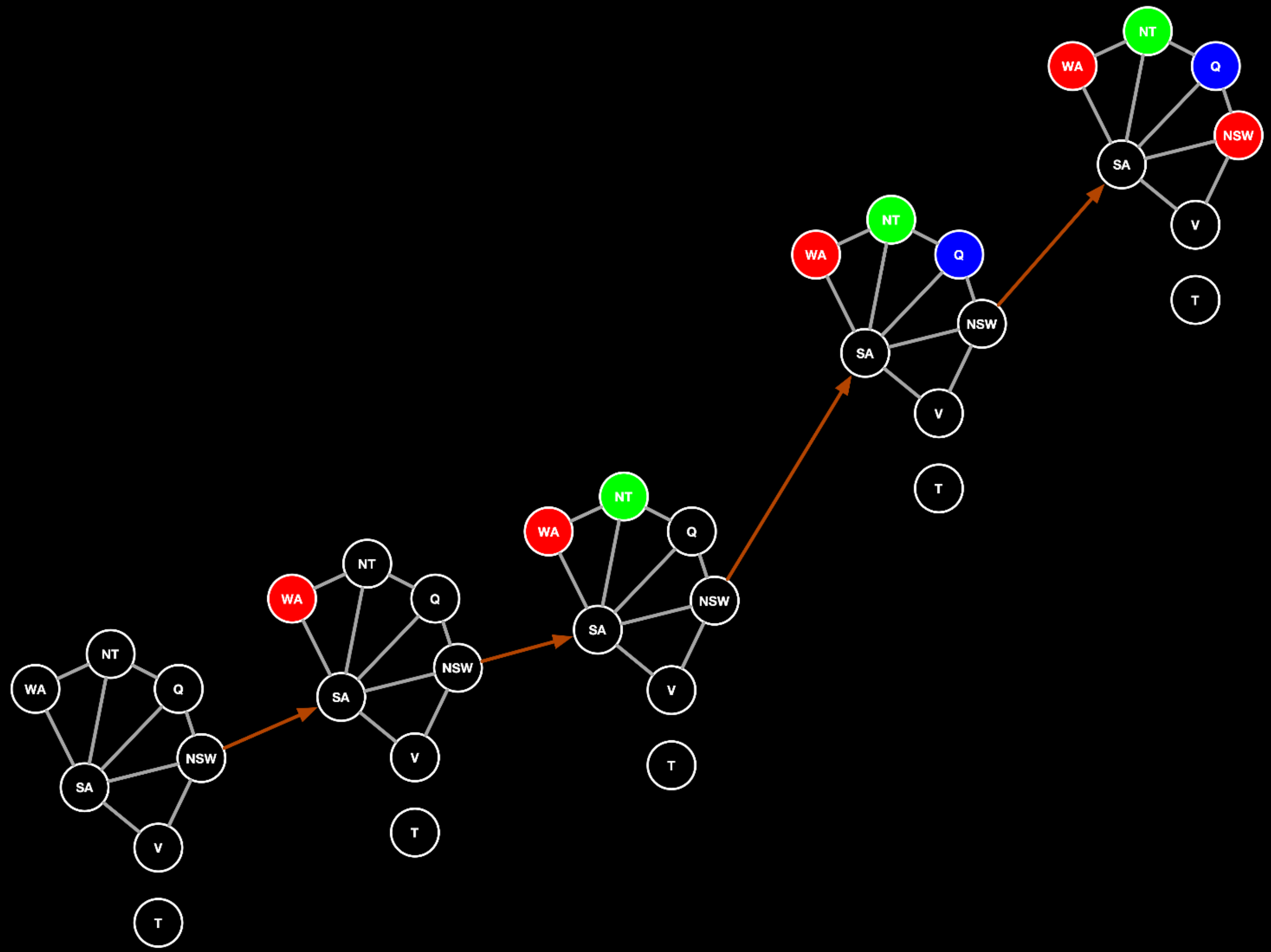
Backtracking search

COLORING EXAMPLE

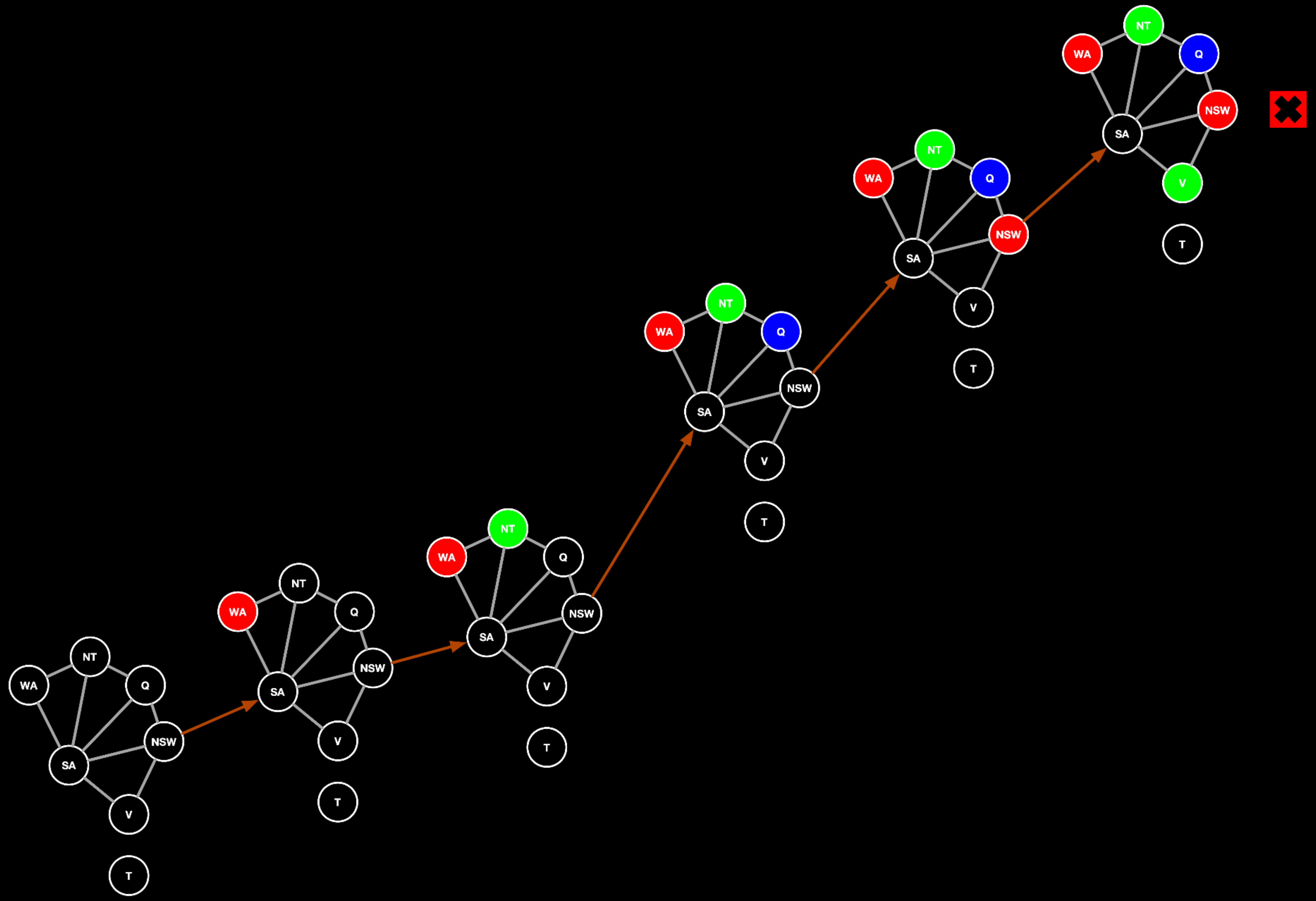


Backtracking search

COLORING EXAMPLE

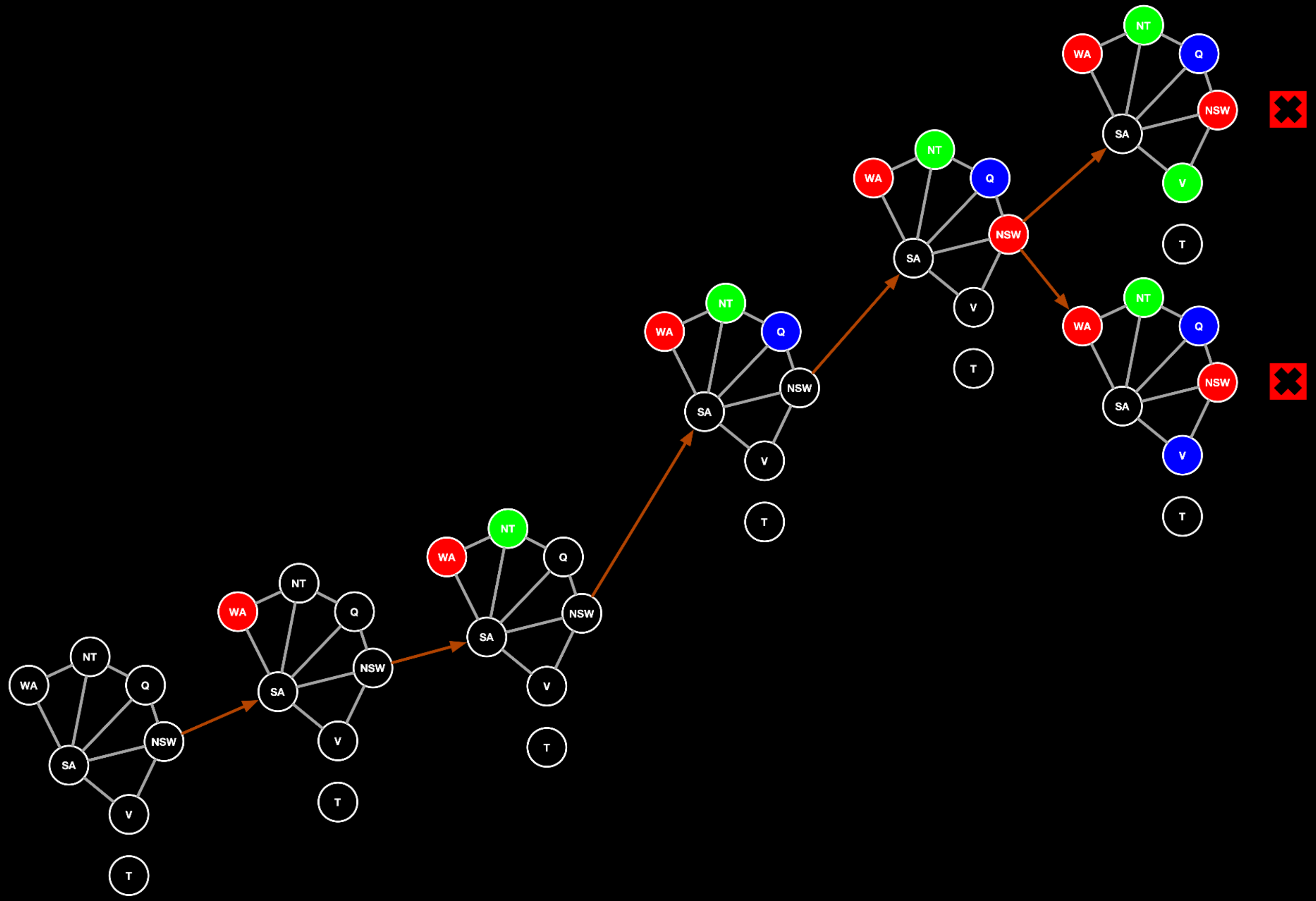


Backtracking search
COLORING EXAMPLE



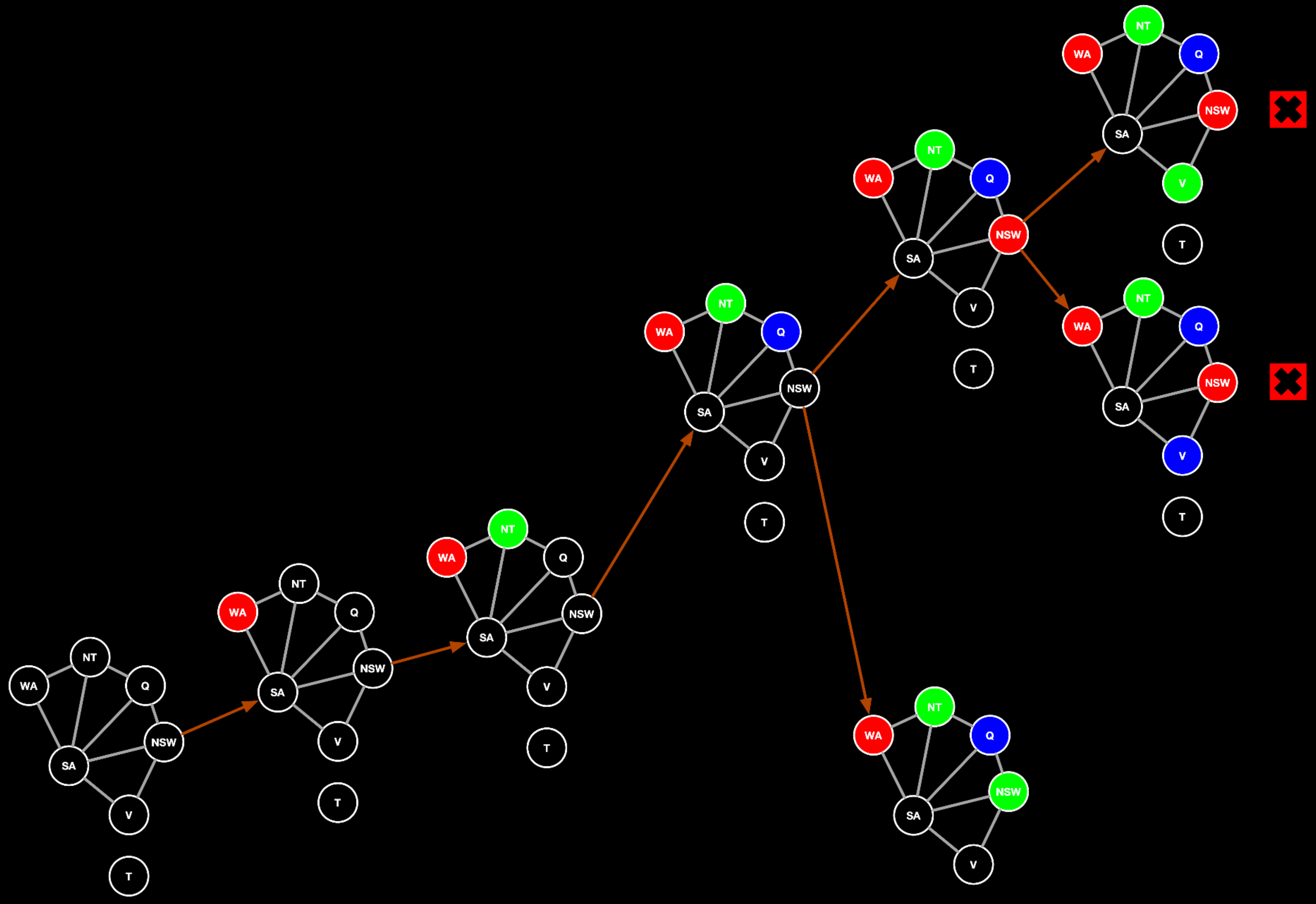
Backtracking search

COLORING EXAMPLE



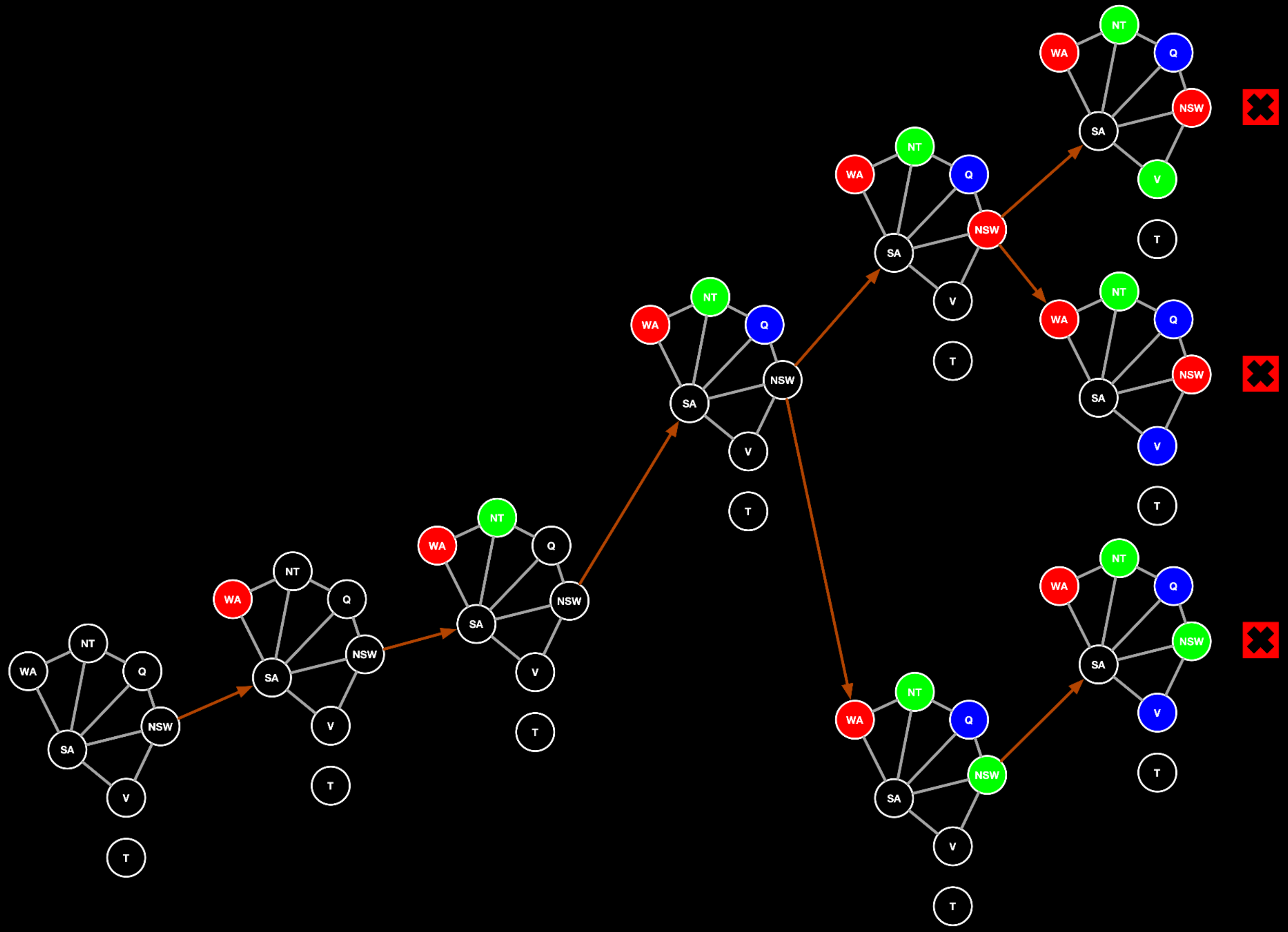
Backtracking search

COLORING EXAMPLE



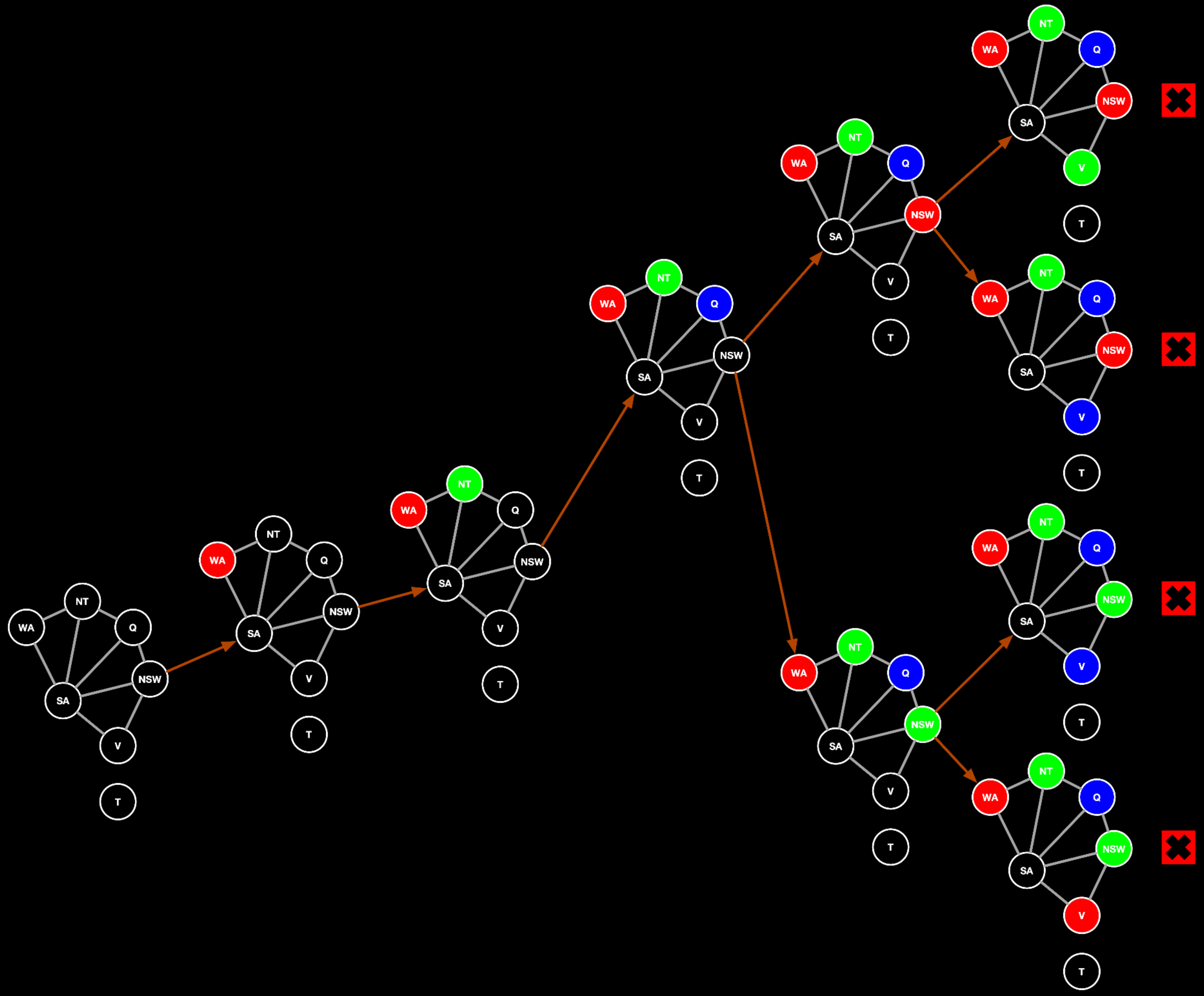
Backtracking search

COLORING EXAMPLE



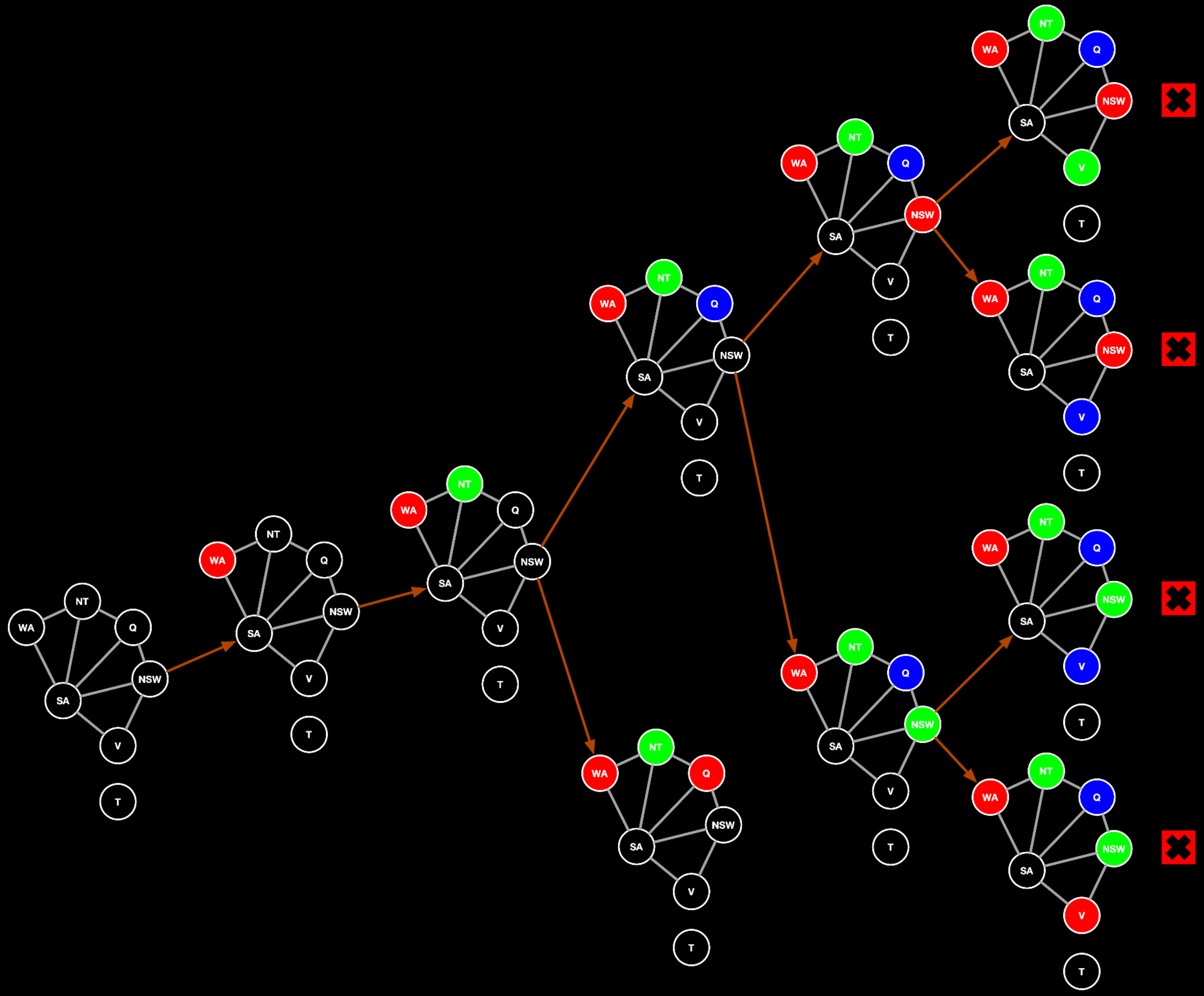
Backtracking search

COLORING EXAMPLE



Backtracking search

COLORING EXAMPLE



Backtracking search

COLORING EXAMPLE

```
function Backtracking-search(csp) returns SOLUTION, or FAILURE
  return Recursive-backtracking({}, csp)

function Recursive-backtracking(assignment, csp) returns SOLUTION, or FAILURE
  if assignment is complete then
    return assignment

  variable = Select-unassigned-variable(variables[csp], assignment, csp)
  for each value in Order-Domain-Value variable, assignment, csp) do
    if value is consistent with assignment given Constraints[csp] then
        add {variable = value} to assignment
        result = Recursive-backtracking(assignment, csp)

        if result ≠ failure then
          return result
        remove {variable = value} from assignment

  return FAILURE
```


Backtracking search

PSEUDO-CODE

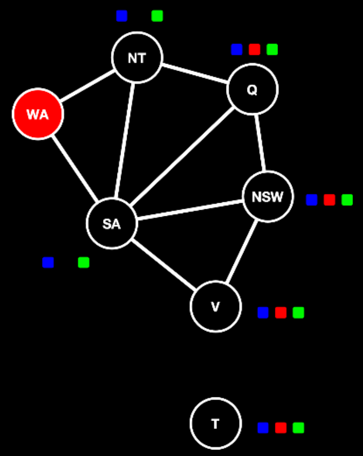
We can improve backtracking even more with some additional improvements:

- **IMPROVEMENT 1**
  Taking divination class: filter out inevitable failures as early as possible
- **IMPROVEMENT 2**
  Do not choose poorly: choose carefully which variable for assignment
- **IMPROVEMENT 3**
  You should never, never doubt something that no one is sure of: choose judicially what value to use
- **IMPROVEMENT 4**
  Where we're going, we don't need... roads: choose judicially where to backtrack to
- **IMPROVEMENT 5**
  See the whole board: use the topology of the problem, or its structure, to assign variables

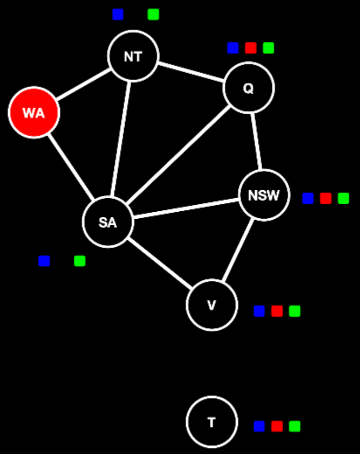
SECTION 03

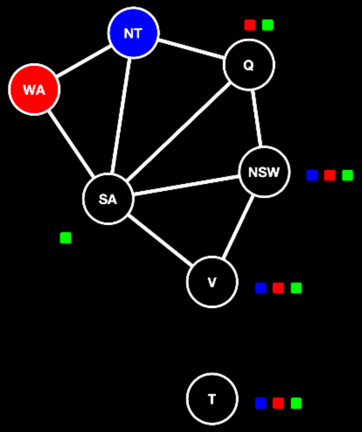
Filtering

Forward checking keeps track of the domains for the unassigned variables and remove possible bad options right away

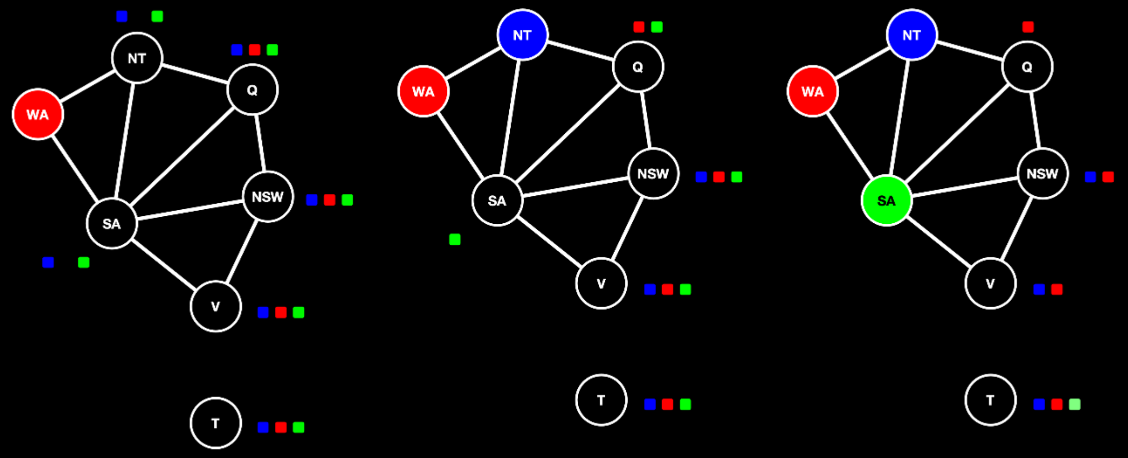


Forward checking keeps track of the domains for the unassigned variables and remove possible bad options right away

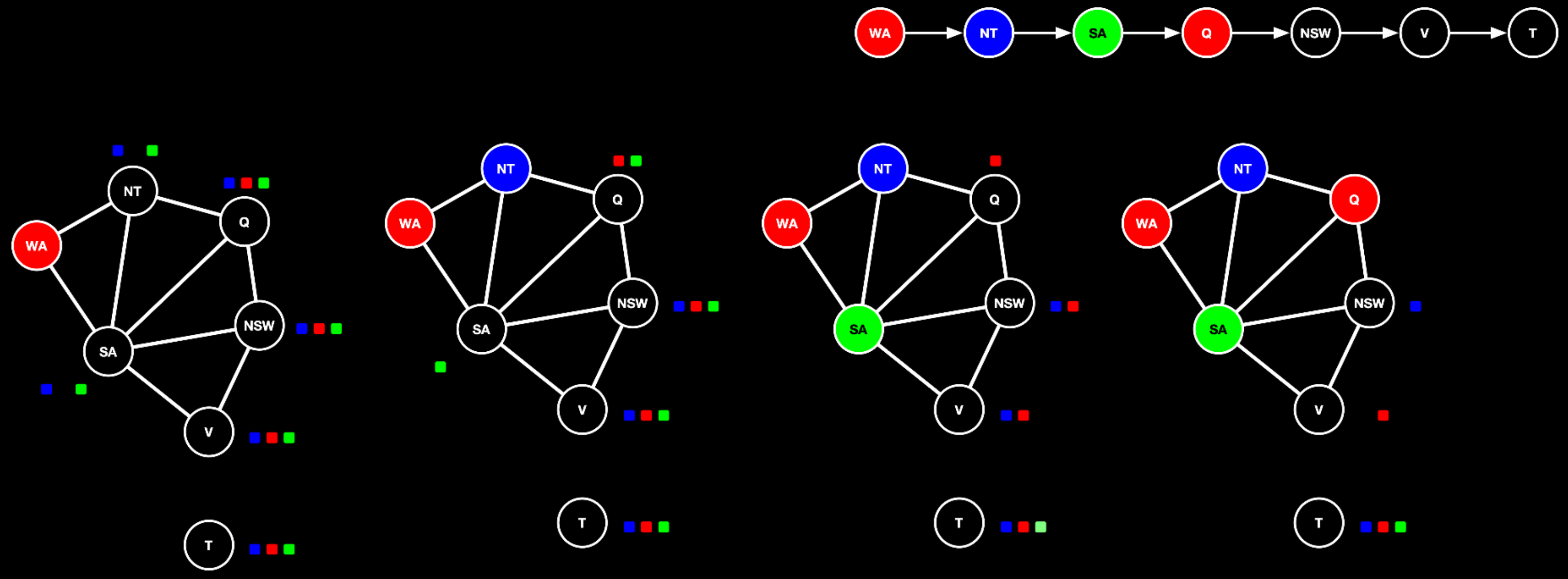




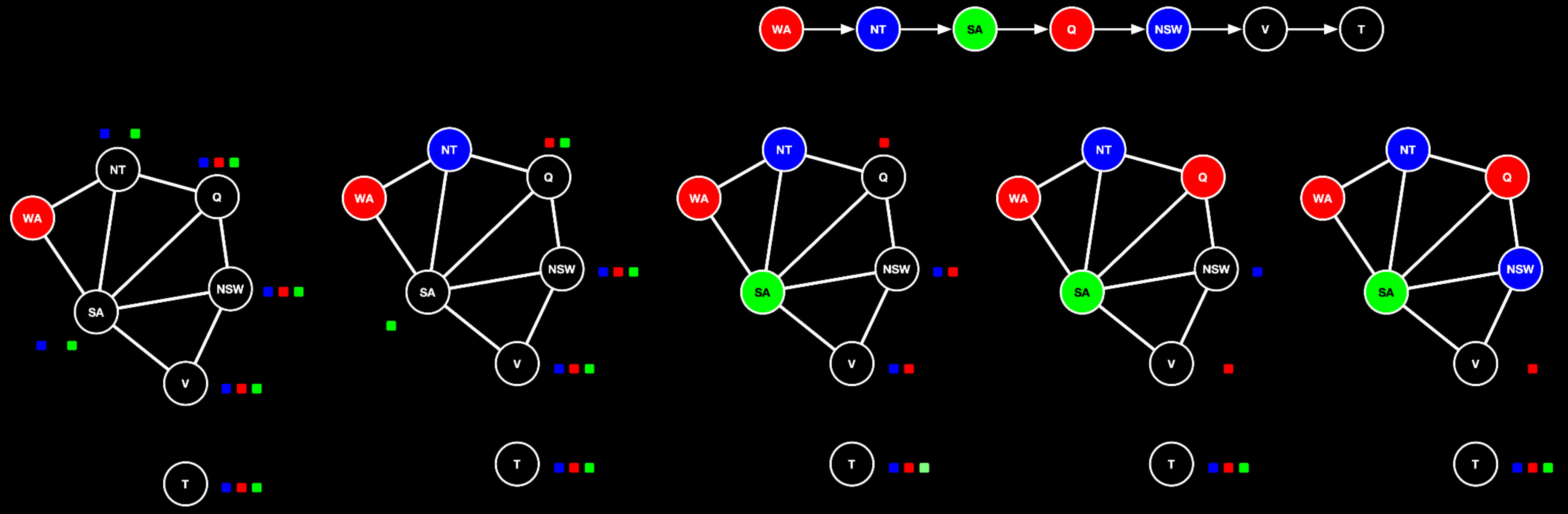
Forward checking keeps track of the domains for the unassigned variables and remove possible bad options right away



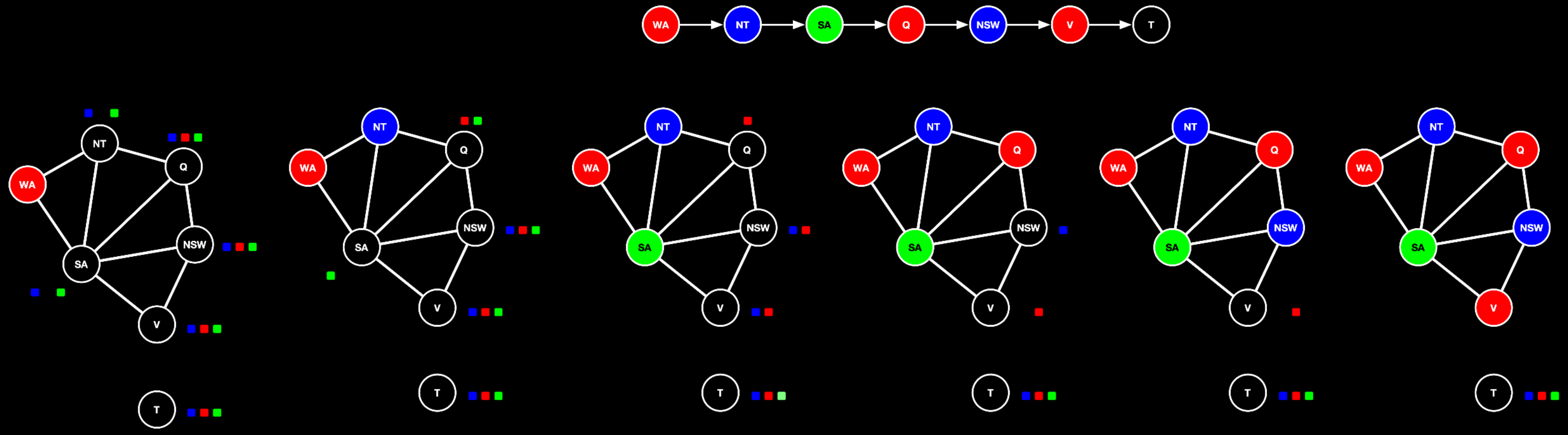
Forward checking keeps track of the domains for the unassigned variables and remove possible bad options right away



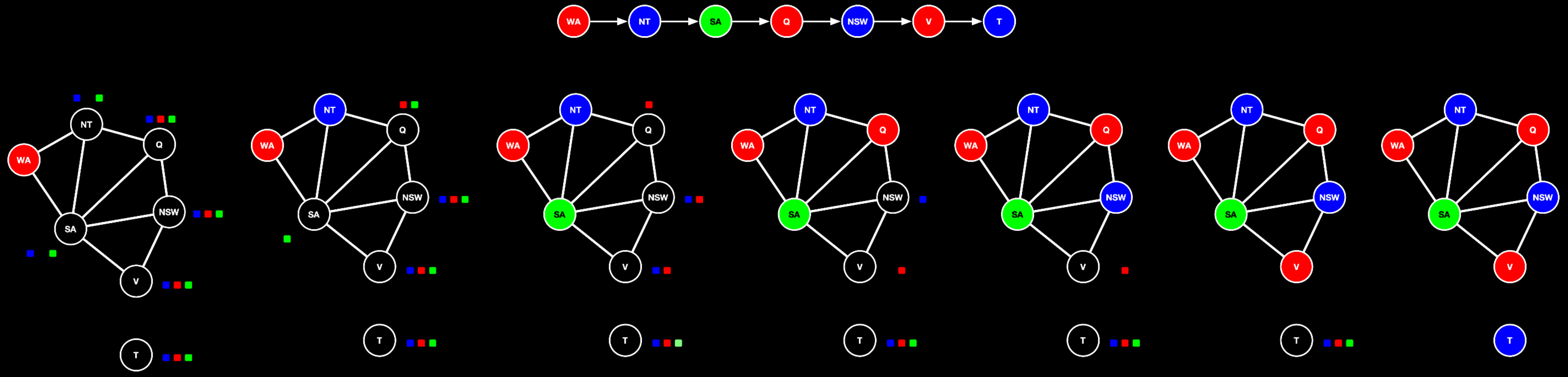
Forward checking keeps track of the domains for the unassigned variables and remove possible bad options right away



Forward checking keeps track of the domains for the unassigned variables and remove possible bad options right away



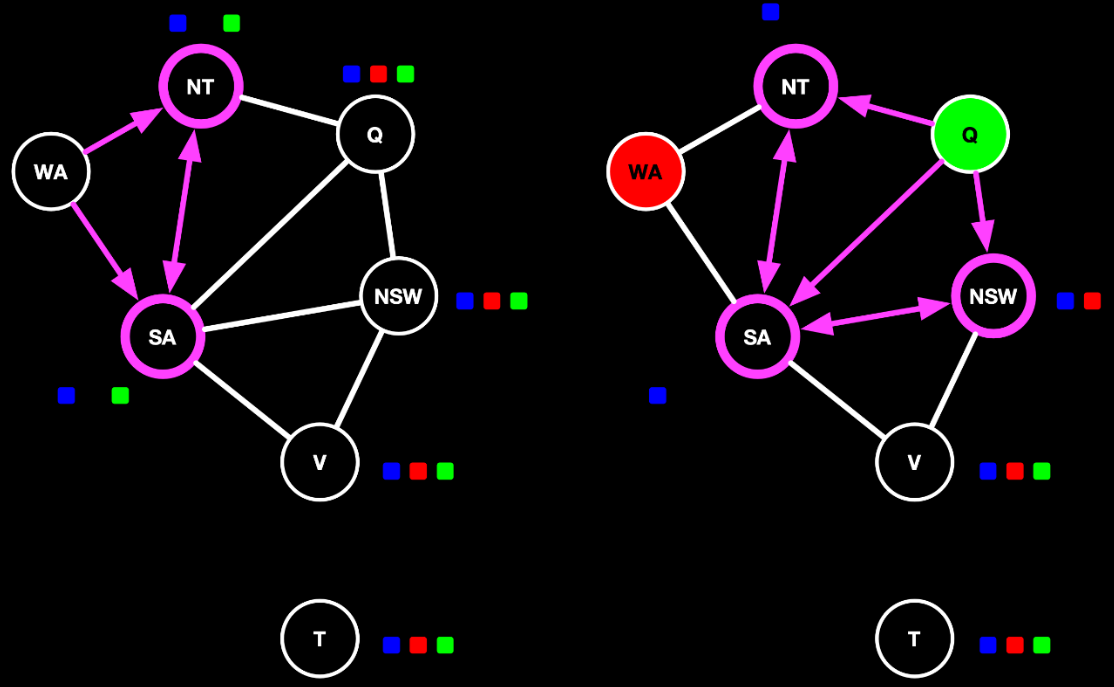
Forward checking keeps track of the domains for the unassigned variables and remove possible bad options right away



Arc consistency is one form of constraint propagation that tries to prune illegal assignment before they happen

While evaluating N_Y, an arc $N_X \rightarrow N_Y$ from a neighbor N_X is **consistent** if and only if every $x \in X$ there is some $y \in Y$ which could be assigned without violating a constraint:

Arc consistency on N_X is also triggered if the domain of N_X (X that is) changes.



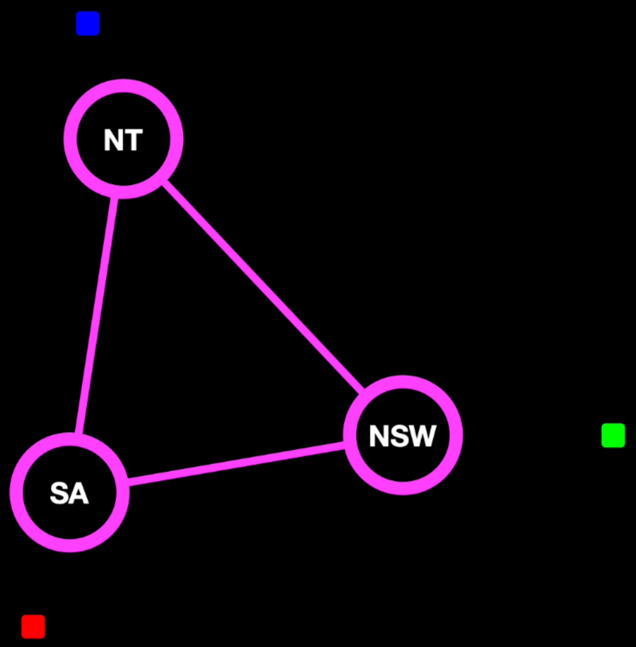
```
function AC-3(csp) returns SOLUTION
push all arcs in queue
while queue is not empty do
  pop arc(Xi,Xj) from queue
  if Remove-Inconsistent-Values(Xi,Xj) then
    for each Xk in Neighbors Xi) do
      add(Xk,Xi) to queue

function Remove-Inconsistent-Values(Xi,Xj)
  removed = false
  for each x in Domain(Xi) do
    if no value y in Domain(Xj)
        allow (x, y) to satisfy the constraint Xi ↔ Xj then
      delete x from Domain
      removed = true
  return removed
```


Filtering

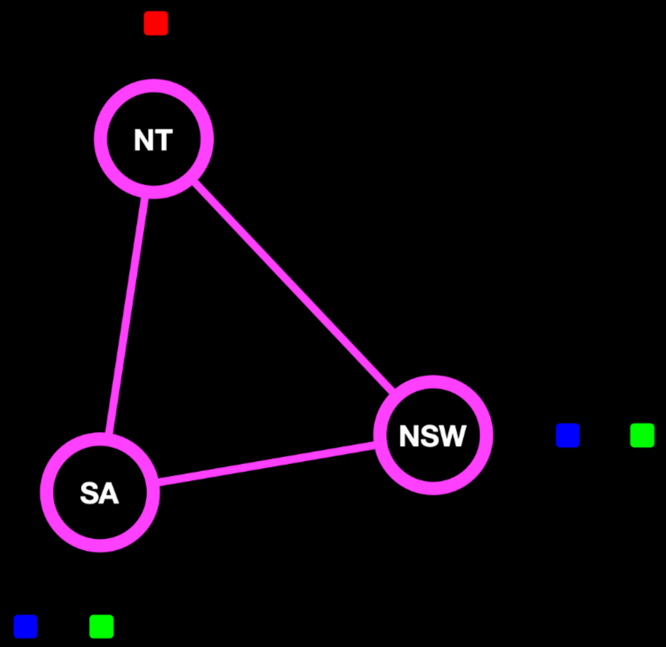
ARC CONSISTENCY: THE ALGORITHM

Arc consistency **must run inside** a backtracking search because:

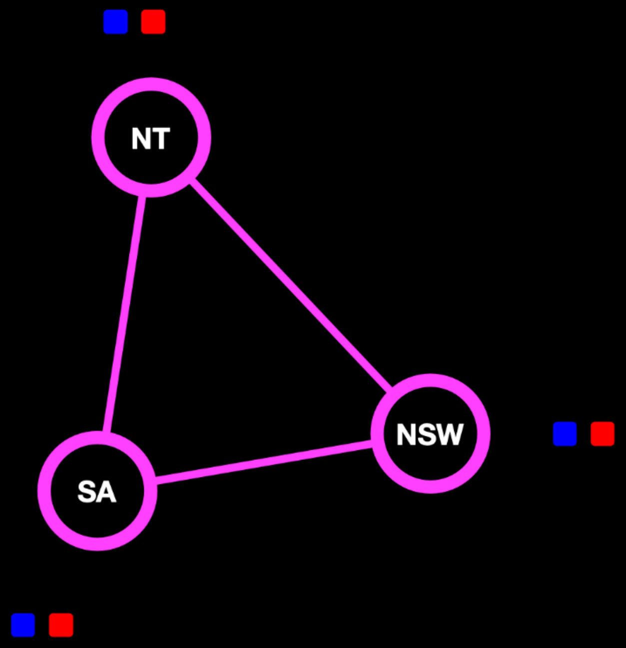
- There might still be one or more solutions left
- There might be no solution left



1 SOLUTION



? SOLUTIONS



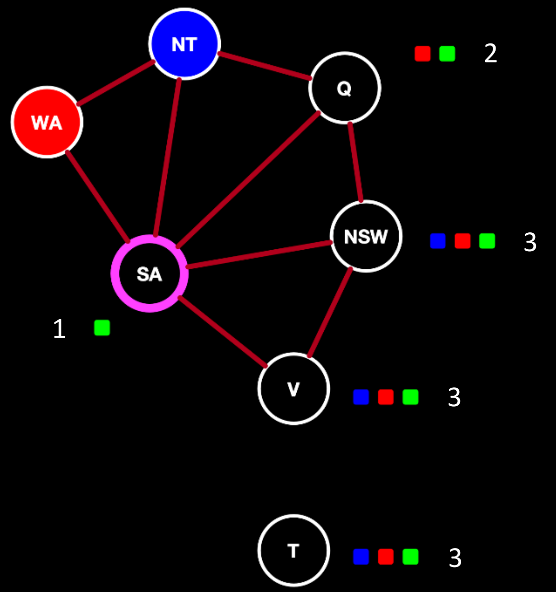
? SOLUTIONS

SECTION 04

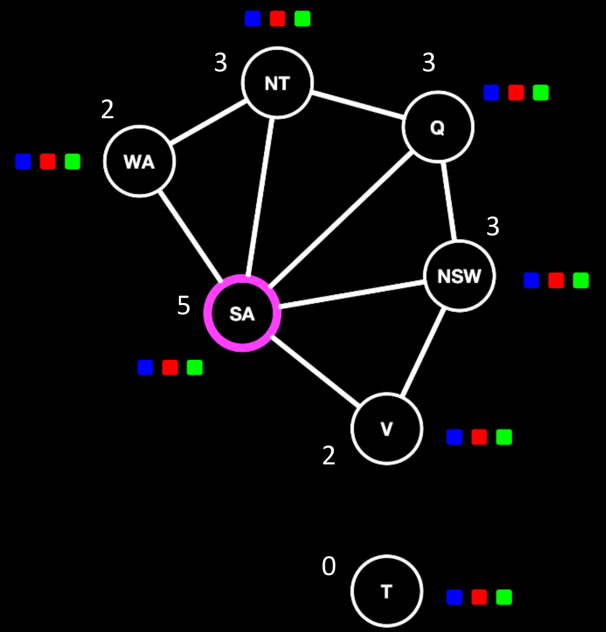
Variable ordering

In order to prune even more illegal assignments, we also have to consider how we choose the variables:

Minimum Remaining Values (MRV): choose the variable with the fewest legal values in its domain



Degree Heuristic: choose the variable with the highest number of constraints

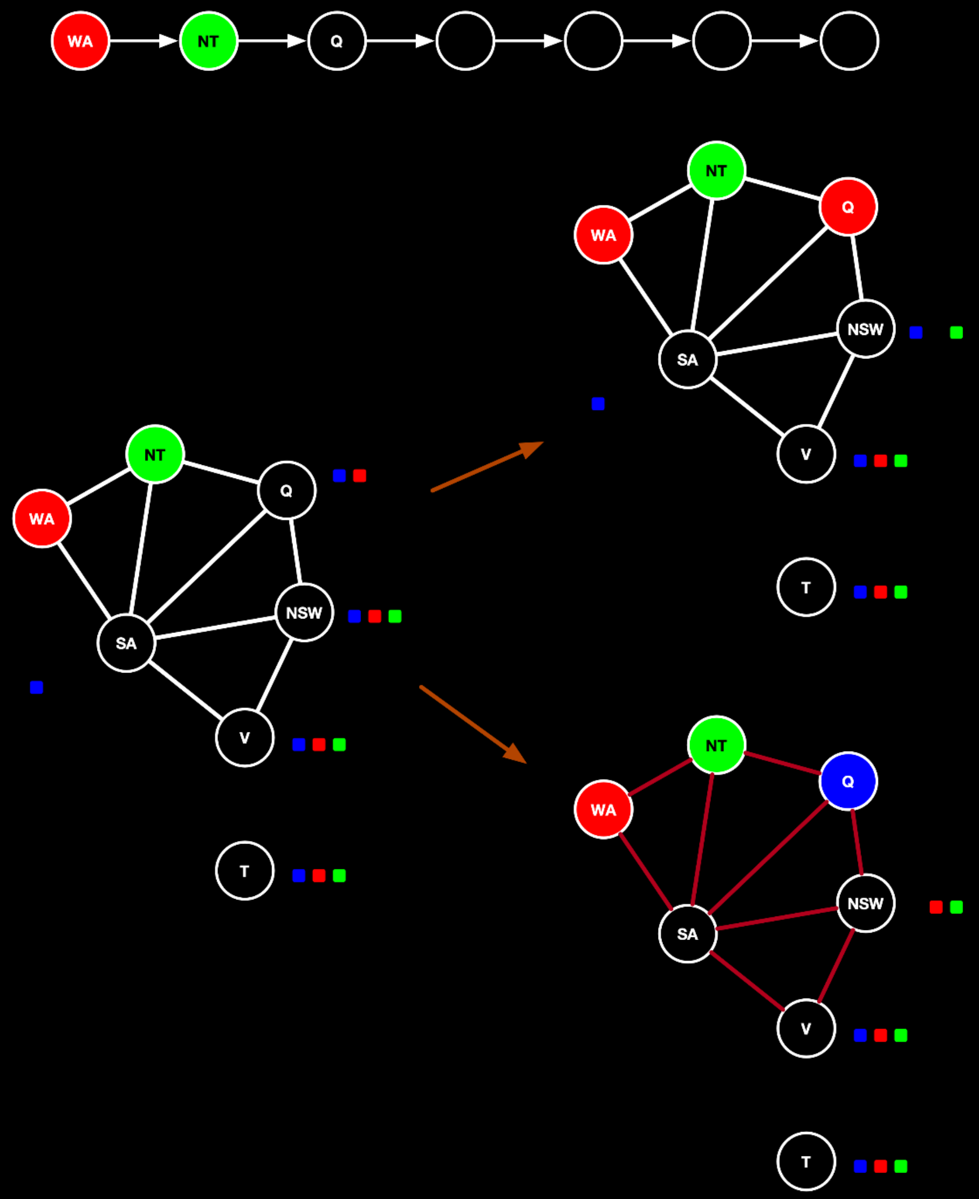


SECTION 05

Value ordering

Least Constraining Value (LCV): Once the variable is selected, choose the value that rules out the fewest choices for the neighbors:

- **First choice**: Only NSW is affected
- **Second choice**: Both SA and NSW are affected



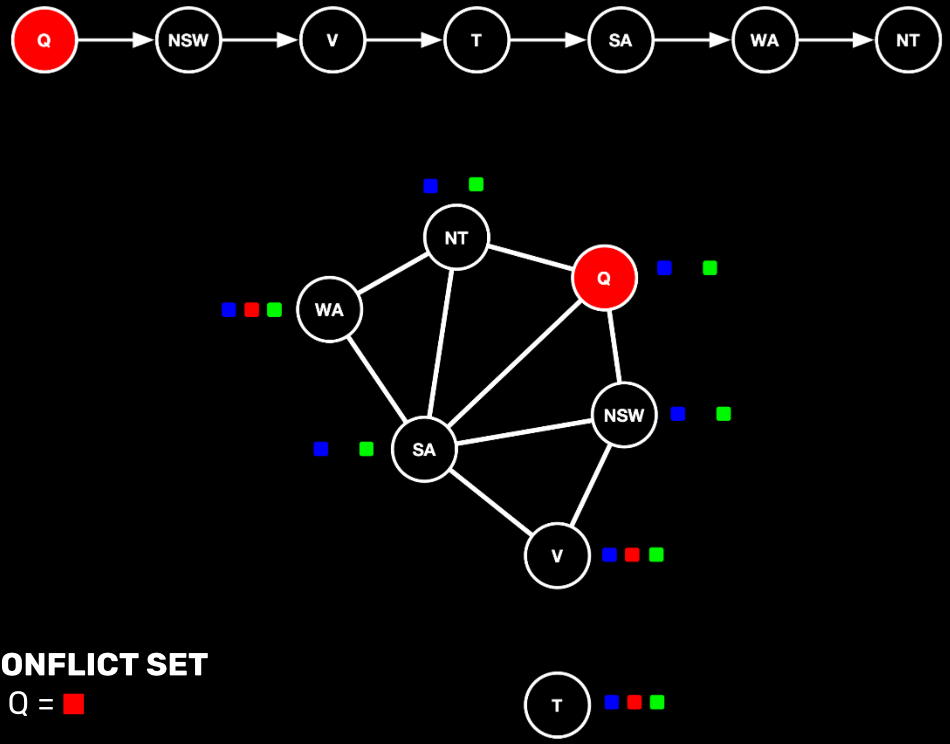
SECTION 06

Smart backtracking

Backjumping is a smarter approach is to jump back to the most recent conflict using the idea of **conflict sets.**

A **conflict set** is a stack that tracks the latest chosen conflicting assignment.

A **conflicting assignment** remove values from the domain of neighboring variables.

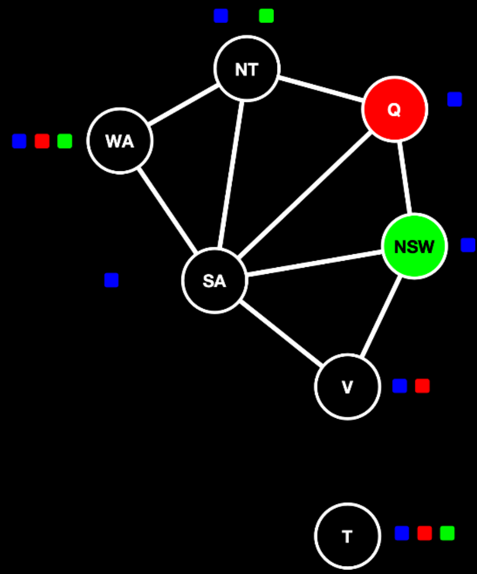


CONFLICT SET
1: Q =

Backjumping is a smarter approach is to jump back to the most recent conflict using the idea of **conflict sets.**

A **conflict set** is a stack that tracks the latest chosen conflicting assignment.

A **conflicting assignment** remove values from the domain of neighboring variables.

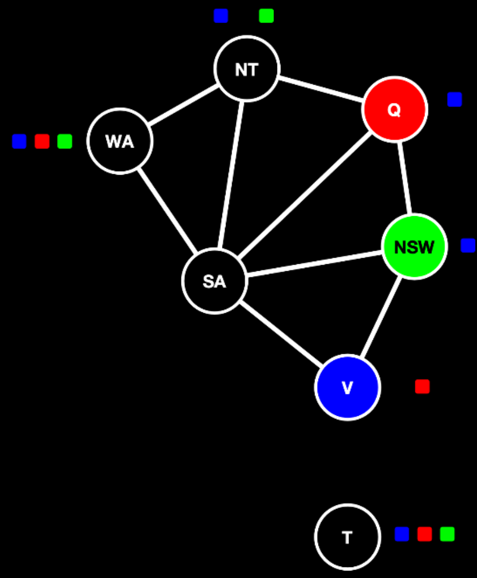


CONFLICT SET
2: NSW =
1: Q =

Backjumping is a smarter approach is to jump back to the most recent conflict using the idea of **conflict sets.**

A **conflict set** is a stack that tracks the latest chosen conflicting assignment.

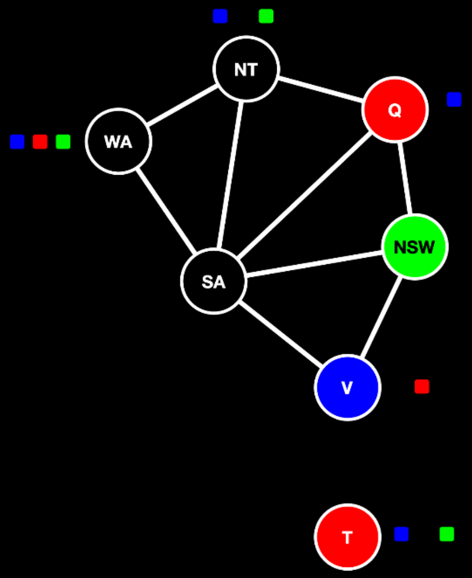
A **conflicting assignment** remove values from the domain of neighboring variables.



CONFLICT SET
3: V =
2: NSW =
1: Q =

Backjumping is a smarter approach is to jump back to the most recent conflict using the idea of **conflict sets.**

A **conflict set** is a stack that tracks the latest chosen conflicting assignment.



A **conflicting assignment** remove values from the domain of neighboring variables.

CONFLICT SET

3: V =

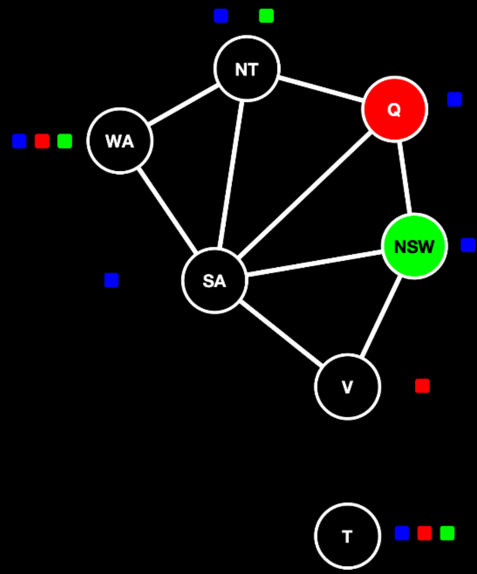
2: NSW =

1: Q =

Backjumping is a smarter approach is to jump back to the most recent conflict using the idea of **conflict sets.**

A **conflict set** is a stack that tracks the latest chosen conflicting assignment.

A **conflicting assignment** remove values from the domain of neighboring variables.

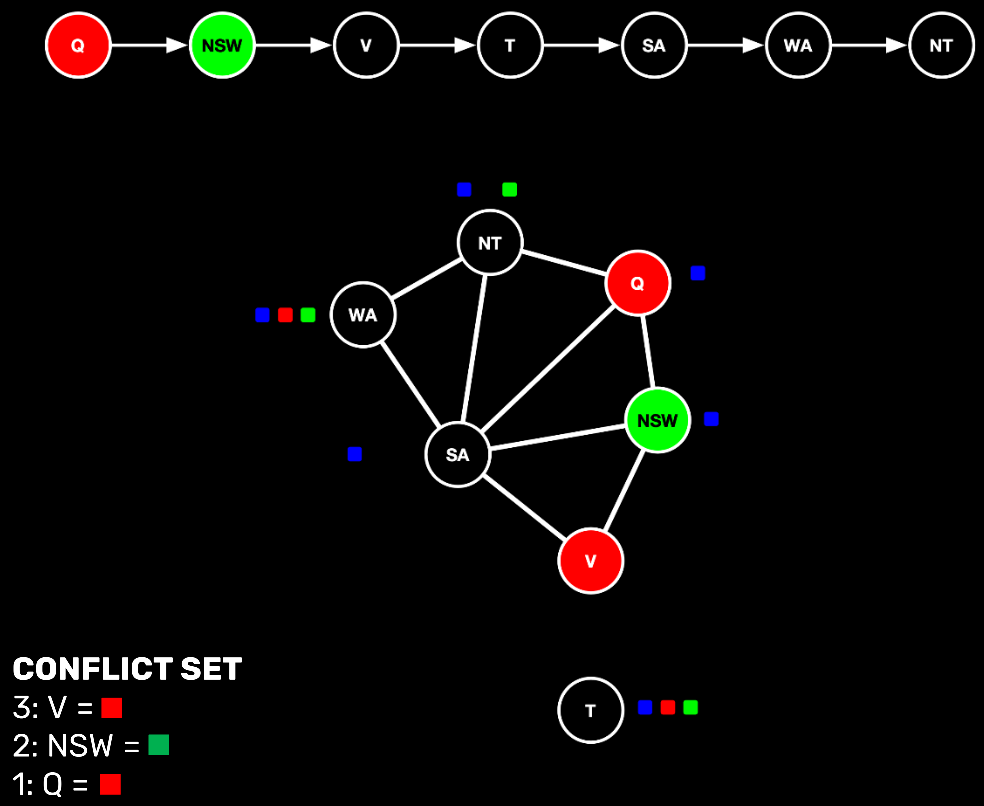


CONFLICT SET
2: NSW = ■
1: Q = ■

Backjumping is a smarter approach is to jump back to the most recent conflict using the idea of **conflict sets.**

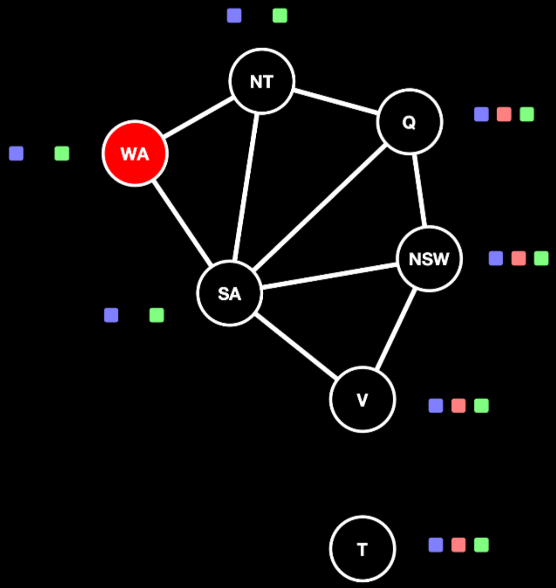
A **conflict set** is a stack that tracks the latest chosen conflicting assignment.

A **conflicting assignment** remove values from the domain of neighboring variables.



In **Conflict-directed Backjumping** each variable has a conflict set.

To find a new assignment after a conflict, the conflict sets **migrate** from one variable to another **until there is a value available** to try.



WA | CONFLICT SET

NSW | CONFLICT SET

T | CONFLICT SET

NT | CONFLICT SET
WA

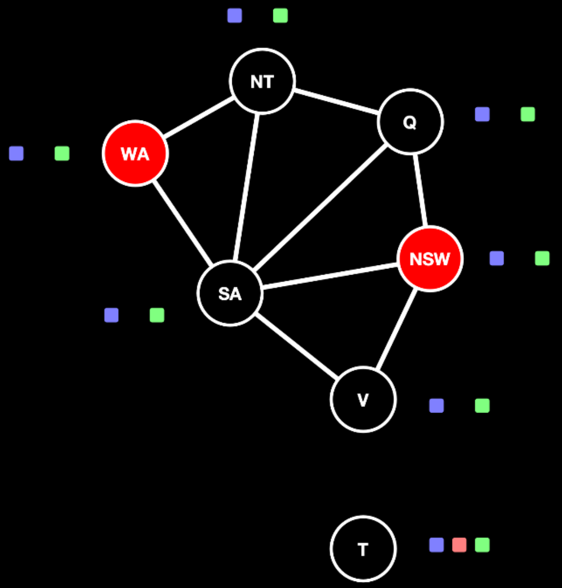
Q | CONFLICT SET

SA | CONFLICT SET
WA

V | CONFLICT SET

In **Conflict-directed Backjumping** each variable has a conflict set.

To find a new assignment after a conflict, the conflict sets **migrate** from one variable to another **until there is a value available** to try.



WA | CONFLICT SET

NSW | CONFLICT SET

T | CONFLICT SET

NT | CONFLICT SET
- WA

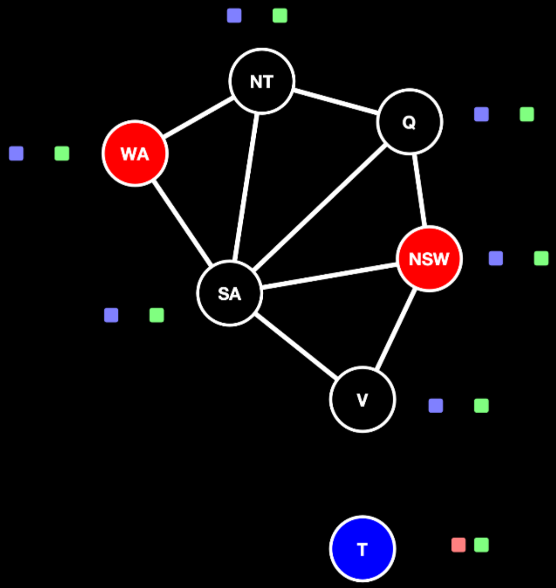
Q | CONFLICT SET
- NSW

SA | CONFLICT SET
- NSW
- WA

V | CONFLICT SET
- NSW

In **Conflict-directed Backjumping** each variable has a conflict set.

To find a new assignment after a conflict, the conflict sets **migrate** from one variable to another **until there is a value available** to try.



WA | CONFLICT SET

NSW | CONFLICT SET

T | CONFLICT SET

NT | CONFLICT SET
- WA

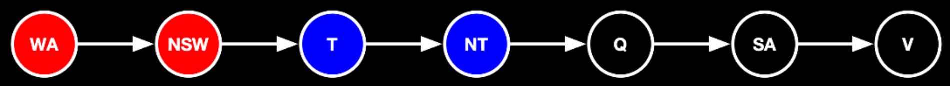
Q | CONFLICT SET
- NSW

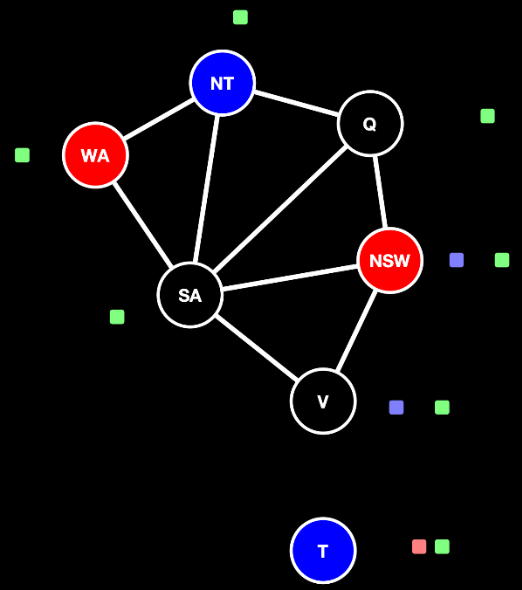
SA | CONFLICT SET
- NSW
- WA

V | CONFLICT SET
- NSW

In **Conflict-directed Backjumping** each variable has a conflict set.

To find a new assignment after a conflict, the conflict sets **migrate** from one variable to another **until there is a value available** to try.





WA | CONFLICT SET
- NT

NSW | CONFLICT SET

T | CONFLICT SET

NT | CONFLICT SET
- WA

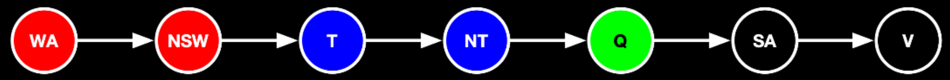
Q | CONFLICT SET
- NT
- NSW

SA | CONFLICT SET
- NT
- NSW
- WA

V | CONFLICT SET
- NSW

In **Conflict-directed Backjumping** each variable has a conflict set.

To find a new assignment after a conflict, the conflict sets **migrate** from one variable to another **until there is a value available** to try.



WA | CONFLICT SET
- NT

NSW | CONFLICT SET
- Q

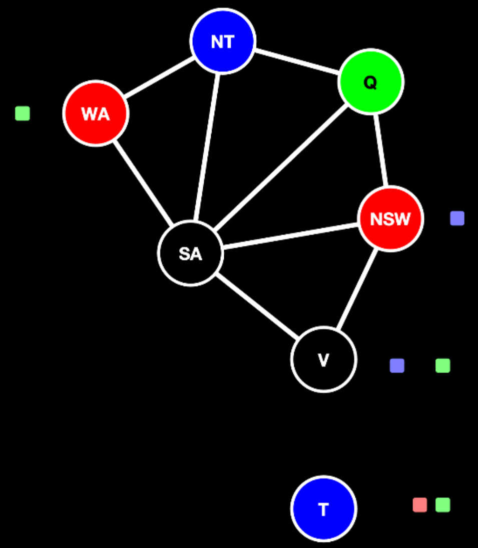
T | CONFLICT SET

NT | CONFLICT SET
- Q
- WA

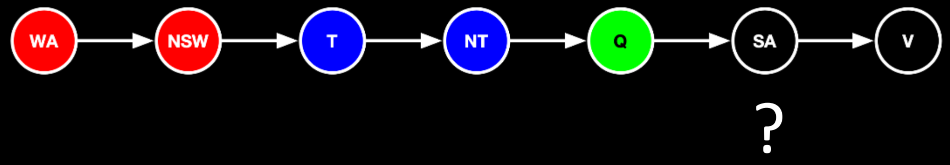
Q | CONFLICT SET
- NT
- NSW

SA | CONFLICT SET
- Q
- NT
- NSW
- WA

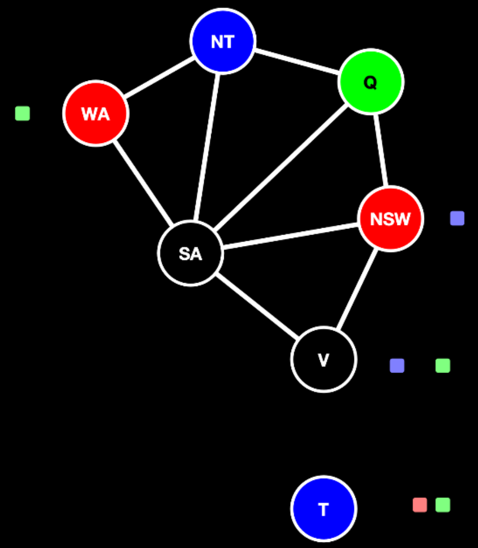
V | CONFLICT SET
- NSW



In **Conflict-directed Backjumping** each variable has a conflict set.



To find a new assignment after a conflict, the conflict sets **migrate** from one variable to another **until there is a value available** to try.



WA | CONFLICT SET
- NT

NSW | CONFLICT SET
- Q

T | CONFLICT SET

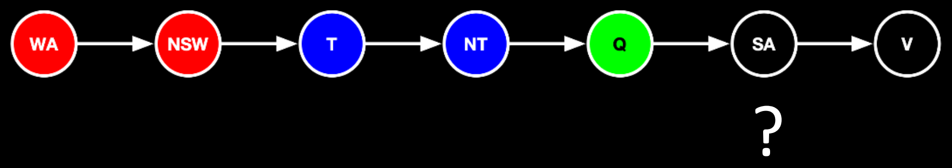
NT | CONFLICT SET
- Q
- WA

Q | CONFLICT SET
- NT
- NSW

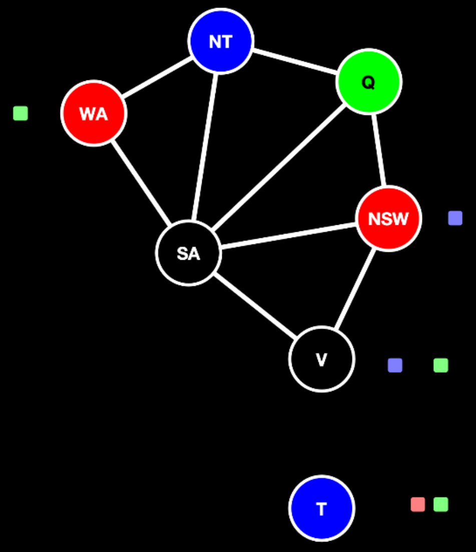
SA | CONFLICT SET
- Q
- NT
- NSW
- WA

V | CONFLICT SET
- NSW

In Conflict-directed Backjumping each variable has a conflict set.



To find a new assignment after a conflict, the conflict sets **migrate** from one variable to another **until there is a value available** to try.



WA | CONFLICT SET
NT

Q | CONFLICT SET
NT
NSW

NSW | CONFLICT SET
Q

SA | CONFLICT SET
Q
NT
NSW
WA

T | CONFLICT SET

NT | CONFLICT SET
Q
WA

V | CONFLICT SET
NSW

SA

CONFLICT SET
4: Q =
3: NT =
2: NSW =
1: WA =

Q

CONFLICT SET
2: NT =
1: NSW =

NEW CONFLICT SET
3: NT =
2: NSW =
1: WA =

NT

CONFLICT SET
1: WA =

NEW CONFLICT SET
2: NSW =
1: WA =

NSW

SECTION 07

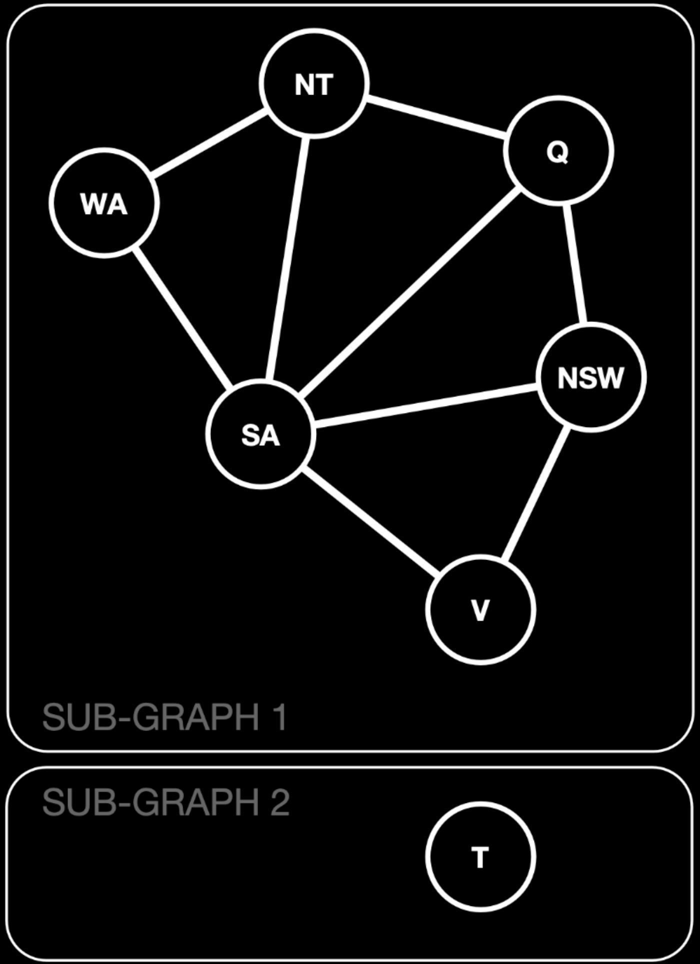
Problem structure

Independent sub-problems can make life much easier:

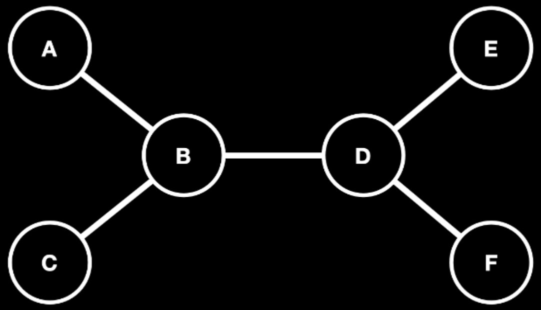
- The worst-case complexity of a solution search is normally $O(d^n)$. For a problem with $n = 60$ and $d = 2$ (a binary domain), and assuming 1M node/s evaluation, the search takes 36,558 years

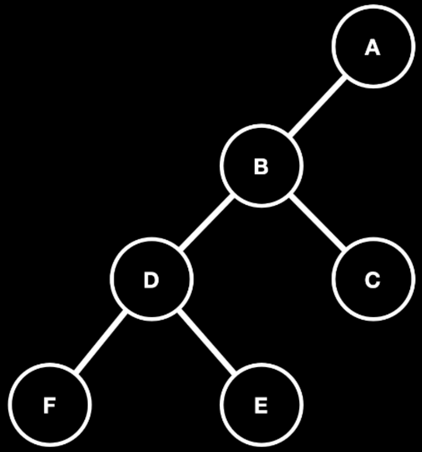
- In the case the problem can be broken into smaller problems with c variables, worst-case complexity is $O\left(\frac{n}{c}d^c\right)$. For the same problem above, with $c = 20$, the search would only be 3s

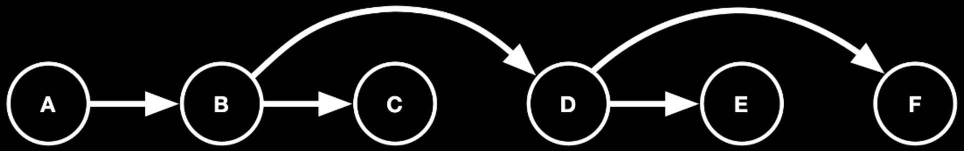
- Independent sub-problems are identifiable as connected components of the constraint graph



If the hyper-graph is a **near-tree graph** (without loops), the CSP can be solved with an arc consistency check with complexity $O(nd^2)$.







1. **Remove backward**: Apply **arc consistency** from the deepest leaf to its parent. After this phase, all arcs are consistent.

2. **Assign forward**: Assign a value to the variable consistent with its parent. Forward assignment will never backtrack.

Sometimes is possible to find one or more variables that, if instantiated, transform the constraint graph into a tree. This process is called **cutset conditioning**.

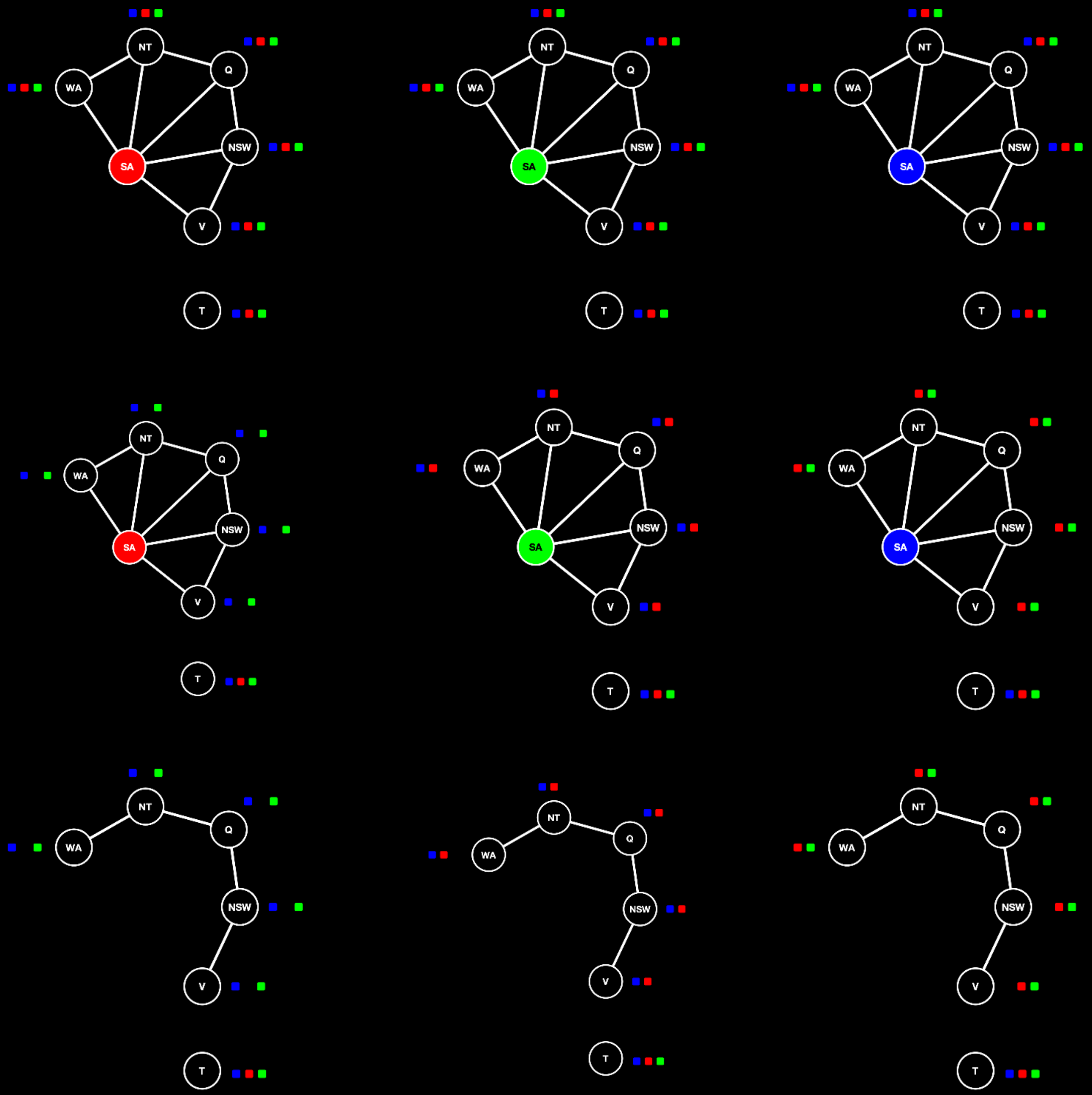
With a cutset of size c, complexity of nearly tree-structured CSPs is $O(d^c(n-c)d^2)$

The process requires to instantiate the variables of the cutset and prune its neighbors' domain.

INSTANTIATE THE CUTSET

PRUNE THE REMAINING DOMAINS

SOLVE THE RESIDUAL CSPS



READINGS

Chapters 6: Constraint Satisfaction Problems

QUESTIONS ?

Tufts UNIVERSITY

ARTIFICIAL INTELLIGENCE

COMP 131

FABRIZIO SANTINI

VERSION 3.0