VU Einführung in Wissensbasierte Systeme

WS 2010/11

Hans Tompits

Institut für Informationssysteme
Arbeitsbereich Wissensbasierte Systeme

www.kr.tuwien.ac.at

3. Constraint Satisfaction Problems

Constraint Satisfaction Problems (CSPs)

- Standard search problem:
  - From the point of view of a search algorithm, a *state* is a "black box" with no discernible internal structure.
  - It is represented by an arbitrary data structure that can be accessed only by the *problem specific* routines:
    - the successor function,
    - heuristic function,
    - and goal test.

- Constraint satisfaction problem (CSP):
  - The states and the goal test conform to a standard, structured, and simple representation.
  - Search algorithms can be defined that take advantage of the structure of states and use *general-purpose* rather than *problem-specific* heuristics.

Constraint Satisfaction Problems (ctd.)

- In a constraint satisfaction problem
  - a *state* is defined by variables with values from an associated domain
  - the *goal test* is a set of constraints 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.

CSP: Formal Definition

A constraint satisfaction problem (CSP) consists of the following components:

- a finite set $\mathcal{V} = \{V_1, V_2, \ldots, V_n\}$ of variables;
- each variable $V_i \in \mathcal{V}$ has an associated non-empty domain D_i of possible values;
- a finite set $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ of constraints.
    - A constraint $C \in \mathcal{C}$ between variables $V_{i_1}, \ldots, V_{i_j}$ is a subset of $D_{i_1} \times \cdots \times D_{i_j}$.

CSP: Formal Definition (ctd.)

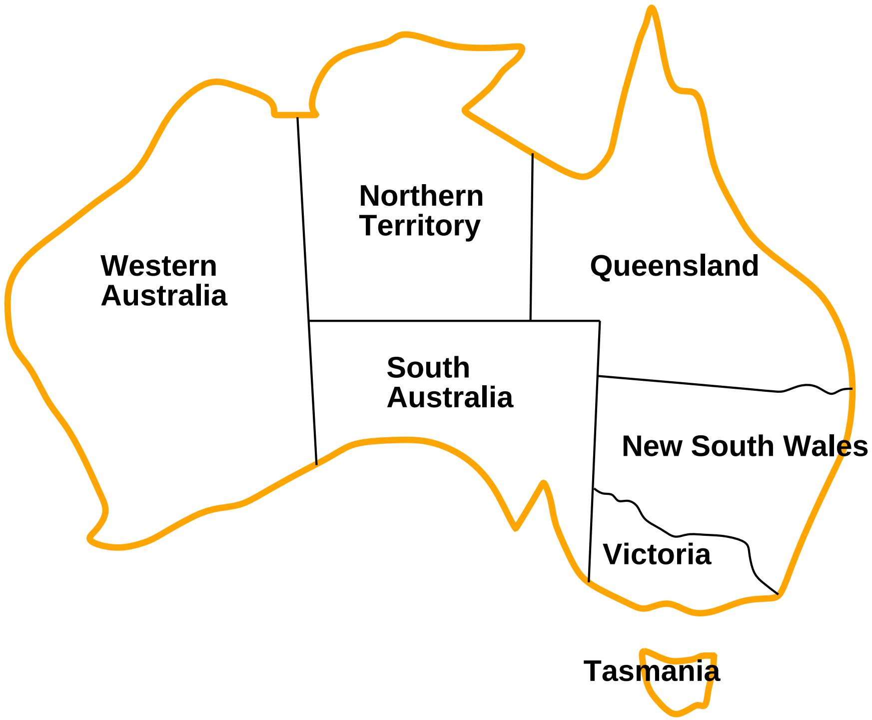
- Each constraint limits the values that variables can take, e.g., $V_1 \neq V_2$.
- There are constraints of different arities:
  - *n*-ary constraints restrict the possible assignment of n variables, i.e., n-ary constraints are *n-ary relations*.
  - In particular:
    - Unary constraints restrict the domain D_i of one variable V_i.
      E.g., $C(V_i) = \{1, 3, 5, 7, 8\}$.
    - Binary constraints restrict the domains $D_i \times D_j$ of a pair of variables V_i, V_j.
      E.g., $C(V_i, V_j) = \{(1, 2), (3, 5), (7, 3), (8, 2)\}$.
    - Ternary constraints,...

CSP: Further notions

- A state of a CSP is defined by an assignment of values to some or all of the variables.
- An assignment that does not violate any constraints is consistent or legal.
- An assignment is complete iff it mentions every variable.
- A solution to a CSP is a complete assignment satisfying all constraints.
- Some CSPs also require a solution that maximises an objective function
  - ↪ these are called constrained optimisation problems.

Example: Map-colouring

Consider the task of colouring a map of Australia with the colours red, green, and blue such that no neighbouring region have the same colour.



Example: Map-colouring (ctd.)

We can formulate this problem as the following CSP:

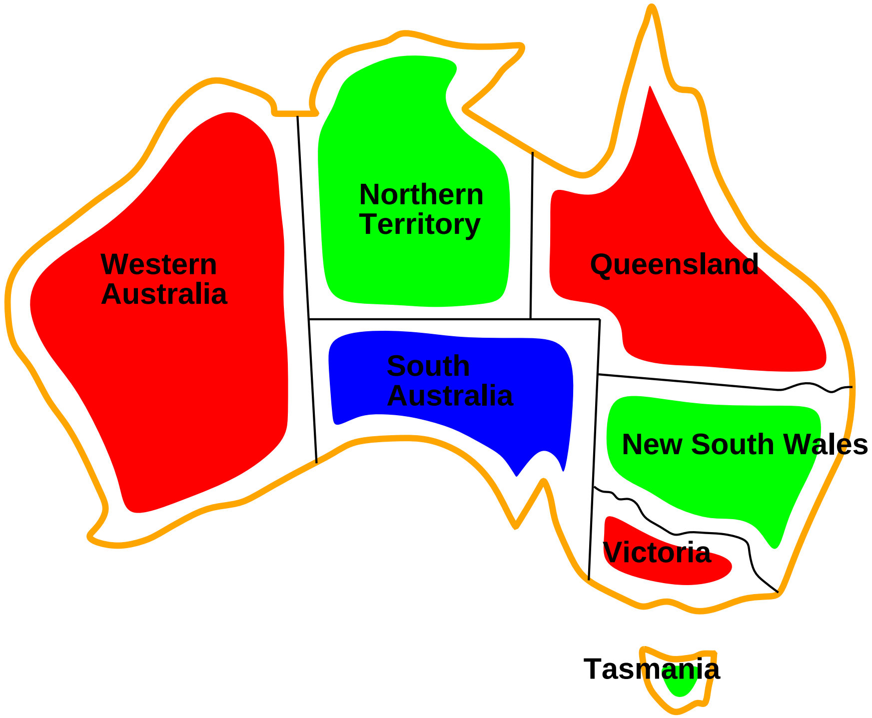
- *Variables:* WA, NT, Q, NSW, V, SA, T
- *Domains:* $D_i = \{red, green, blue\}$
- *Constraints:* adjacent regions must have different colors
  - e.g., the allowable combinations of WA and NT are

$$C(WA, NT) = \{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\},$$

  - or simply written as $WA \neq NT$ (if the language allows this).

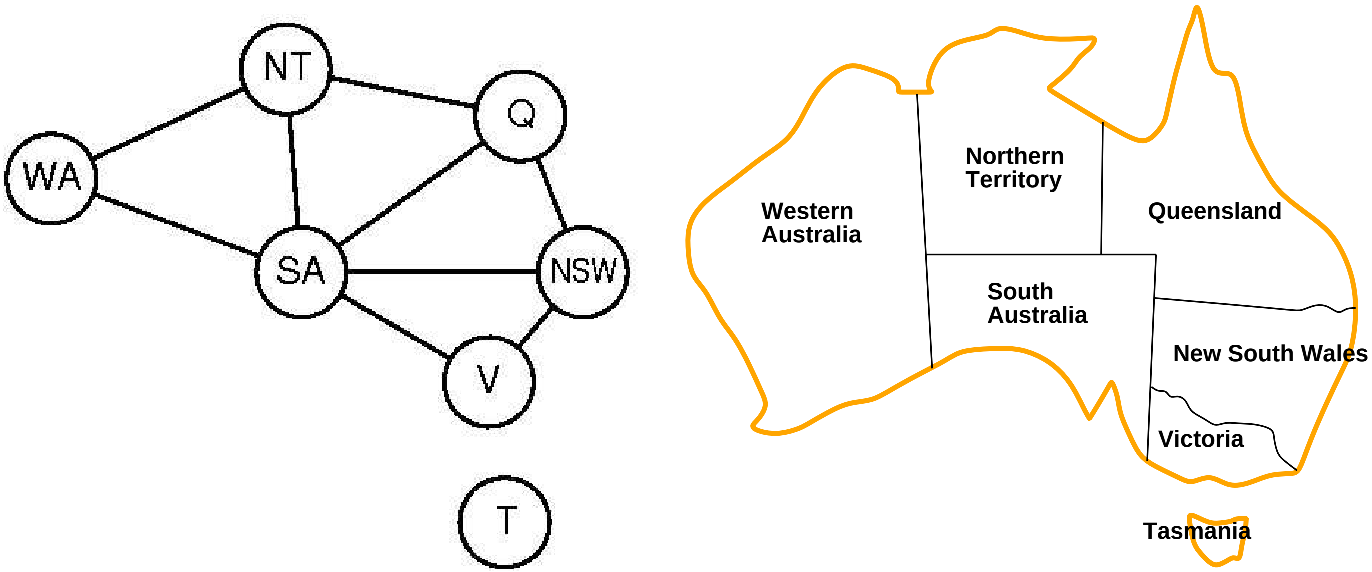
Example: Map-colouring (ctd.)

There are many possible solutions, e.g.,
$\{WA = red, NT = green, Q = red, NSW = green, V = red,$
$SA = blue, T = green\}$



Constraint graph

- For a *binary CSP* (in which all constraints are binary), it is helpful to visualize the problem as a constraint graph.
  - The nodes are the variables,
  - the arcs correspond to the constraints.
- E.g., our map-colouring problem has the following constraint graph:



  - General-purpose CSP algorithms use the *graph structure* to speed up search.
  - E.g., Tasmania is an independent subproblem!

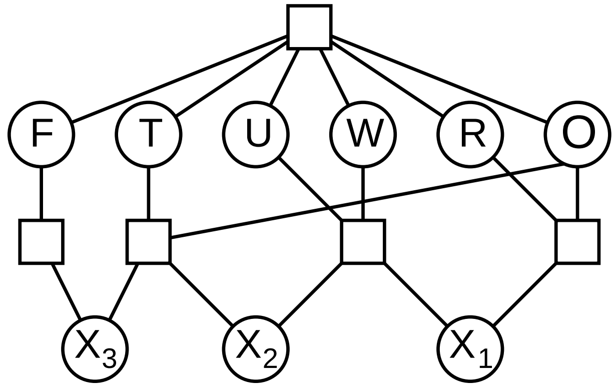
Constraint graph (ctd.)

- Higher-order constraints can be represented by a constraint hypergraph.
    - Reminder: a hypergraph is a pair (X, E), where X is a set of nodes and E is a set of non-empty subsets of X, the hyperedges.
- *Cryptarithmetic puzzles* are examples of higher-order constraints.
    - Usually, one assumes that each letter in a cryptarithmetic puzzle represents a different digit.

Constraint graph (ctd.)

Example:

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



➤ This is formulated as the following CSP:

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

➤ A solution for this CSP is, e.g., $938 + 938 = 1876$.

Varieties of CSPs

- The simplest kind of CSPs involves variables that are *discrete* and have *finite domains*.
  - E.g., map-colouring problems are of this kind.
- If the maximum domain size of any variable in a CSP is d, and there are n variables, then the number of possible complete assignments is $O(d^n)$
  - ↪ exponential in the number of variables!

Varieties of CSPs (ctd.)

- Finite domain CSPs whose variables can be either *true* or *false* are called Boolean CSPs.
- E.g., 3SAT can be expressed as a Boolean CSP
  - a clause like $X_1 \vee \neg X_2 \vee X_3$ corresponds to the constraint

$$C(X_1, X_2, X_3) = \\ (\{true, false\} \times \{true, false\} \times \{true, false\}) \setminus \{(false, true, false)\}.$$

- Since 3SAT is an NP-complete problem we cannot expect to solve finite-domain CSPs in less than exponential time (unless P=NP).
- However, in most *practical* applications, CSP algorithms can solve problems orders of magnitude larger than those solvable via general search algorithms.

Varieties of CSPs (ctd.)

- Discrete variables can also have *infinite domains*, e.g., the set of integers or the set of strings.
  - E.g., for construction job scheduling, variables are the start dates and the possible values are integer numbers of days from the current date.

- Note:
  - With infinite domains it is no longer possible to describe constraints by enumerating all allowed combinations of values.
  - Rather, a constraint language must be used.
    - E.g., if Job_1, which takes 5 days, must precede Job_3, then we need a language of algebraic inequalities like $StartJob_1 + 5 \leq StartJob_3$.

Varieties of CSPs (ctd.)

- It is also no longer possible to solve constraints with infinite domains by enumerating all possible assignments
  - ➥ there are infinitely many of them!
- Special solution algorithms exist for *linear constraints* on integer values
  - linear constraint = variables appear only in *linear* form
  - e.g., $StartJob_1 + 5 \leq StartJob_3$ is linear.
- Non-linear constraints are *undecidable*—no algorithm exists for solving such constraints!

Varieties of CSPs (ctd.)

- Finally, there are CSPs with *continuous domains*
  - very common in real-world applications and widely studied in operations research
  - e.g., scheduling the start/end times for the Hubble Space Telescope.
- Linear constraints can be solved with *linear programming* methods in polynomial time.

Some real-world CSPs

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

☞ Notice that many real-world problems involve real-valued variables.

CSPs as standard search problems

- It is straightforward to give an *incremental formulation* of a CSP as a standard search problem.
  - States are defined by the values assigned so far.
  - *Initial state*: the empty assignment, $\emptyset$.
  - *Successor function*: assign a value to an unassigned variable providing it does not conflict with the current assignment.
  - *Goal test*: the current assignment is complete.
- This is the same for all CSPs!
  - ➥ Any standard search algorithm can be used to solve CSPs.

CSPs as standard search problems (ctd.)

Caveat: Suppose we use breadth-first search.

- If there are n variables and d values, the branching factor at the top level is nd.
- At the next level, the branching factor is $(n-1)d$, and so on for n levels.

➥ We generate a tree with $n!d^n$ leaves although there are only d^n possible complete assignments!

Backtracking search

The naive formulation ignored one crucial property of CSPs:

- ➤ Variable assignments are commutative, i.e., the order of application of any given set of actions has no effect on the outcome
  - ➥ when assigning values to variables, we reach the same partial assignment regardless of order.
- ➤ All CSP search algorithms generate successors by considering possible assignments for a *single* variable at each node in the search tree!
  - E.g., in the map-colouring problem, initially we may have a choice between $SA = red$, $SA = green$, and $SA = blue$,
  - but we would not choose between $SA = red$ and $WA = blue$.

➥ With this restriction, we generate only d^n leaves as expected.

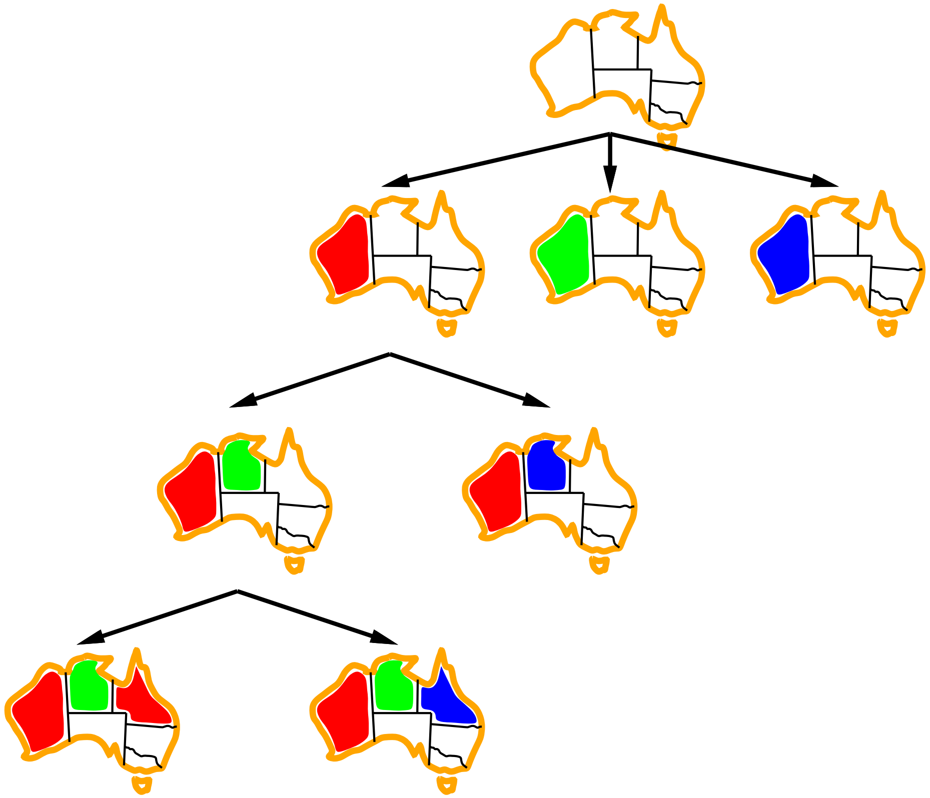
Backtracking search (ctd.)

Depth-first search for CSPs with single-variable assignments is called backtracking search.

- Backtracking search is the basic uninformed algorithm for CSPs
- Can solve n-queens for $n \approx 25$.

Backtracking search (ctd.)

Below gives part of the search tree for the Australia problem, where the variables are assigned in the order WA, NT, Q, . . .

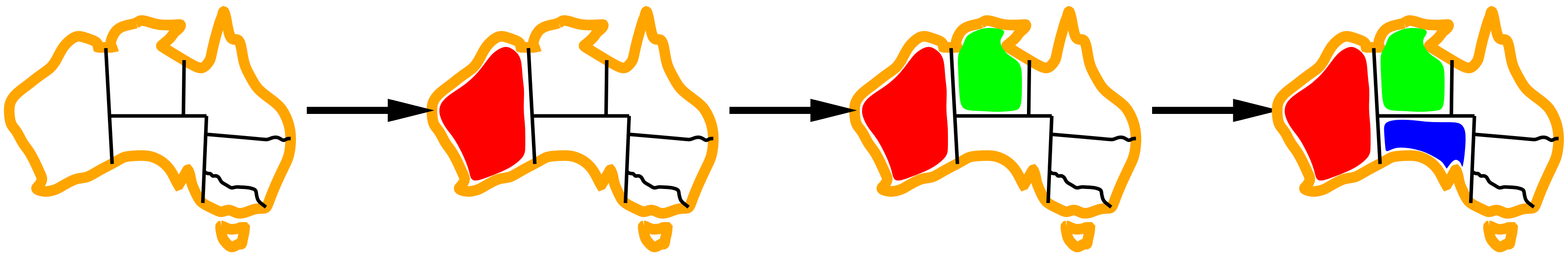


Backtracking search (ctd.)

- Since plain backtracking search is an uninformed algorithm, we do not expect it to be very effective for large problems.
- Different *general-purpose methods* help improving the performance, addressing the following issues:
  - Which variable should be assigned next, and in what order should its values be tried?
  - What are the implications of the current variable assignments for the other unassigned variables?
  - When a path fails, can the search avoid repeating this failure in subsequent paths?

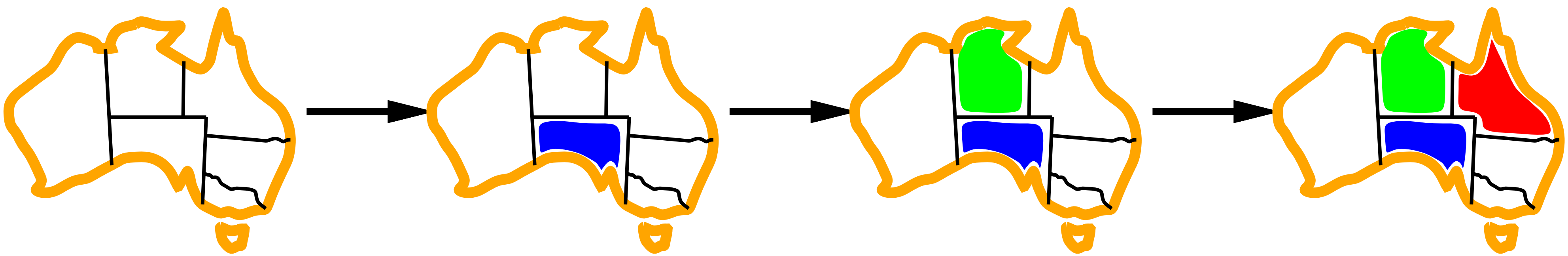
Minimum-remaining-values heuristic

- The minimum-remaining-values (MRV) heuristic:
  - choose the variable with the fewest legal values.
- If there is a variable X with 0 legal values remaining, the MRV heuristic will select X and failure will be detected immediately
  - avoiding pointless searches through other variables.
- E.g., in the Australia example, after the assignments for $WA = red$ and $NT = green$, there is only one possible value for SA.
  - ↪ it makes sense to assign $SA = blue$ next rather than assigning Q.
  - Actually, after SA is assigned, the choices for Q, NSW, and V are all forced.



Degree heuristic

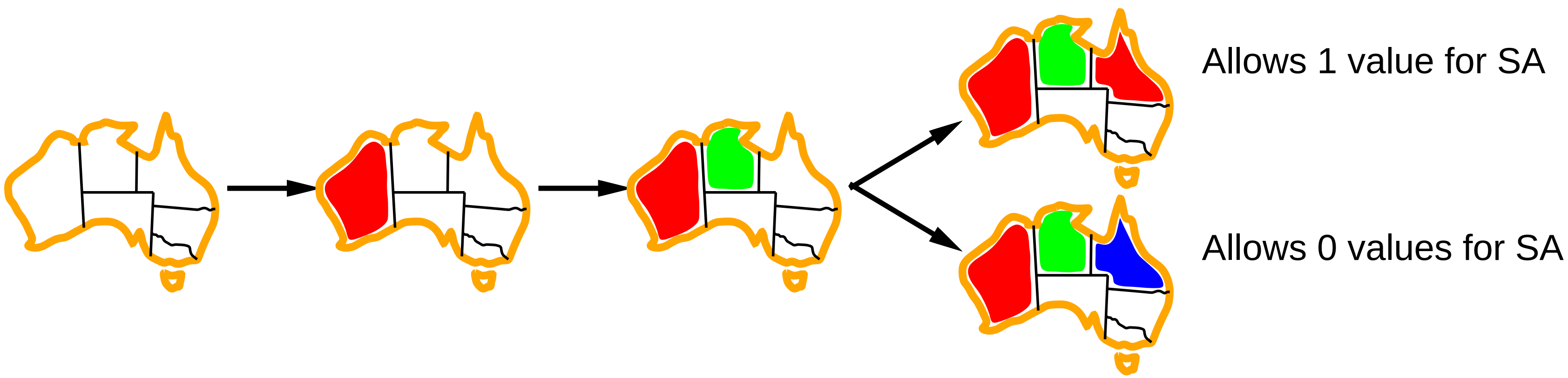
- The MRV heuristic does not help at all in choosing the *first* region to colour.
- In this case, the degree heuristic comes in:
  - it selects the variable that is involved in the *largest number of constraints* on other unassigned variables.
- In the Australia example, *SA* is the variable with highest degree, 5.
  - The others have degree 2 or 3.
  - Actually, once *SA* is chosen, applying the degree heuristic one more time solves the problem without any false steps.



Least-constraining-value heuristic

- Once a variable has been selected, to decide on the order in which to examine its values, the least-constraining-value heuristic can be effective:
  - it prefers the value that rules out the *fewest* choices for the neighbouring variables in the constraint graph.
- In the Australia example, suppose we have the partial assignment $WA = red$ and $NT = green$, and our next choice is for Q.
  - Blue would be a bad choice, because it eliminates the last legal value for Q's neighbour SA.

➥ The least-constraining-value heuristic thus prefers red to blue.

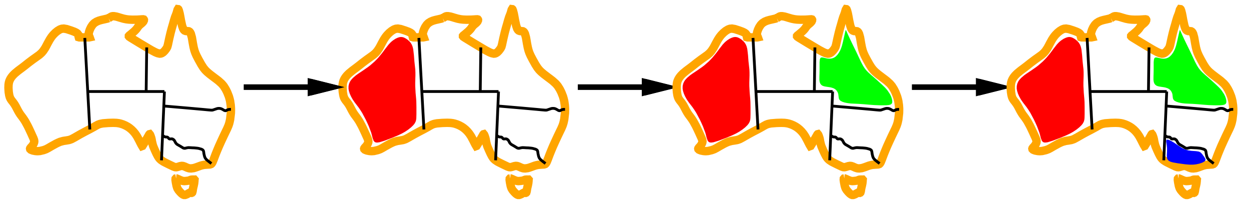


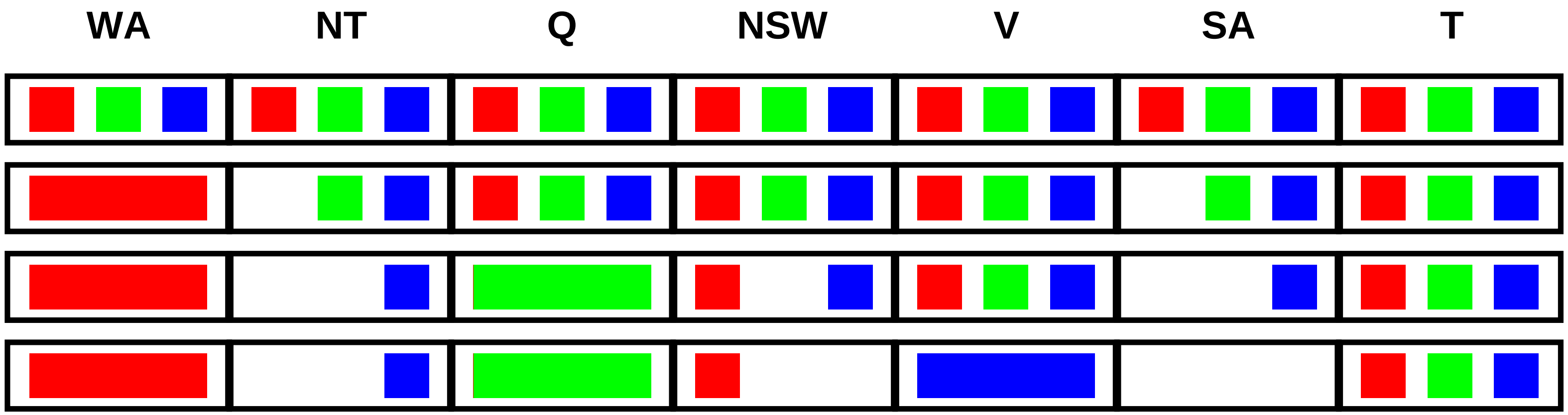
Forward checking

- The methods discussed so far consider the constraints on a variable *only at the time that the variable is chosen.*
- By looking at some of the constraints earlier in the search, or even before the search, the search space can be drastically reduced.
- One such method is forward checking:
  - whenever a variable X is assigned, it looks at each unassigned variable Y that is connected to X by a constraint
  - and deletes from the domain of Y any value that is inconsistent with the value chosen for X.

Forward checking (ctd.)

➤ Consider colouring Australia using forward checking:



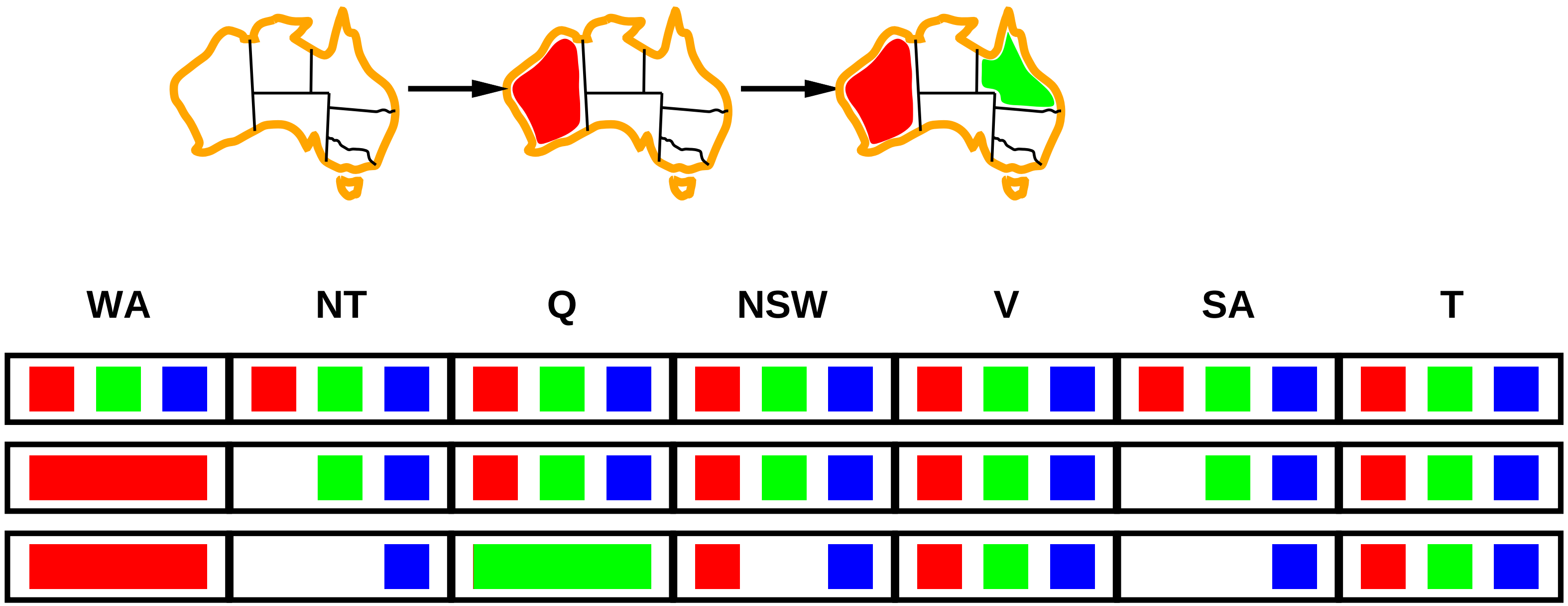


➤ Note:

- After assigning $WA = red$ and $Q = green$, the domains of NT and SA are reduced to a single value.
  ↪ The MRV heuristic would select SA and NT next.
- After assigning $V = blue$, the domain of SA is empty, so we get failure and the algorithm backtracks.

Forward checking (ctd.)

➤ Forward checking does not provide early detection for all failures:

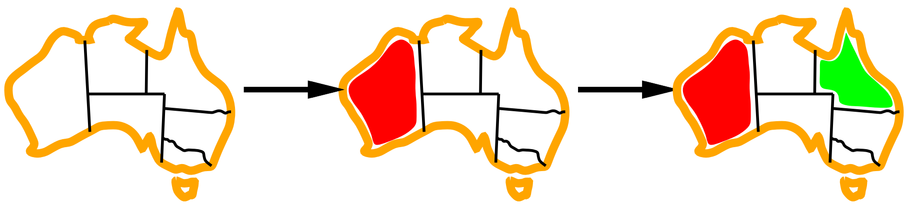


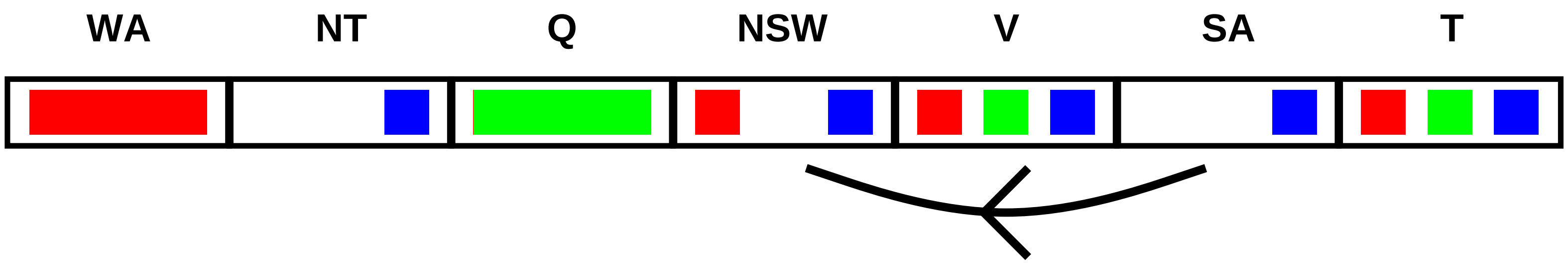
➤ *NT* and *SA* cannot both be blue!

☞ Constraint propagation is the general term for propagating the implications of a constraint on one variable onto other variables.

Arc consistency

- The simplest form of constraint propagation is arc consistency:
  - "arc" refers to a *directed* arc in the constraint graph;
  - $X \rightarrow Y$ is consistent iff for *every* value x of X there is *some* allowed value y of Y.
- For $SA = blue$ in the Australia colouring, there is a consistent assignment for NSW, namely red $\Longrightarrow$ the arc from SA to NSW is consistent
  - the reverse arc is *not* consistent, but can be made so by deleting blue from the domain of NSW.





Further techniques

- Intelligent backtracking:
  - do not backtrack to preceding variable if failure occurs, but go back to one in the set of variables that *caused the failure*
    - this set is the conflict set
    - e.g., backjumping goes to the most recent variable in this conflict set.
- Local search algorithms are very effective for solving CSPs
  - the *million*-queens problem can be solved in an average of 50 steps.
- The structure of the constraint graph can be taken into account.
  - E.g., colouring Tasmania is an independent subproblem of colouring Australia.
  - Tree-structured problems can be solved in linear time.