CMSC 471
Artificial Intelligence

Constraints

Many slides courtesy Tim Finin

Overview

- Constraint satisfaction is a powerful problem-solving paradigm
  - Problem: set of variables to which we must assign values satisfying problem-specific constraints
  - Constraint programming, constraint satisfaction problems (CSPs), constraint logic programming…
- Algorithms for CSPs
  - Backtracking (systematic search)
  - Constraint propagation (k-consistency)
  - Variable and value ordering heuristics
  - Backjumping and dependency-directed backtracking

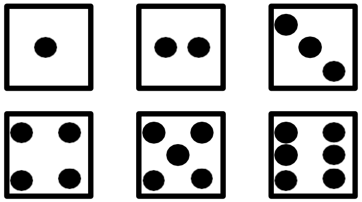
Some Core Terminology

- **(algebraic) variable** is a symbol used to denote features of possible worlds
  - If X is a variable, dom(X) is X's domain (the values X can take on)

UMBC

Example: Variable

Let's consider rolling a standard, six-sided die



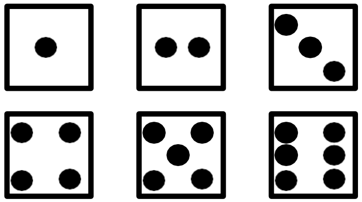
Let X_i be the variable corresponding to the outcome of the ith role

Q: What is $\mathrm{dom}(X_i)$?

Example: Variable

Let's consider rolling a standard, six-sided die

Let X_i be the variable corresponding to the outcome of the *i*th role



Q: What is dom(X_i)?

A: dom(X_i) = {1, 2, 3, 4, 5, 6}

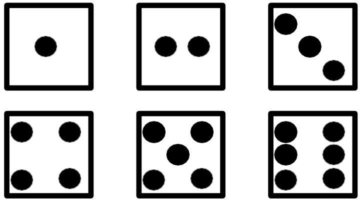
Types of Variables

- Discrete variables have finite or countable domains
  - Binary variables have two values in their domain
  - Boolean variables have two variables, TRUE and FALSE
  - Other examples?

- Continuous have uncountably infinite domains
  - Example types?

Example: Variable

Let’s consider rolling a standard, six-sided die



Let X_i be the variable corresponding to the outcome of the *i*th role

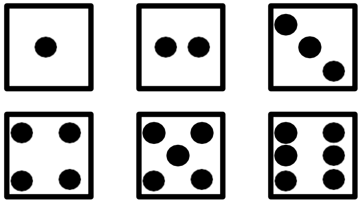
Q: What is dom(X_i)?

Q: Is X_i discrete or continuous?

A: dom(X_i) = {1, 2, 3, 4, 5, 6}

Example: Variable

Let’s consider rolling a standard, six-sided die



Let X_i be the variable corresponding to the outcome of the *i*th role

Q: What is dom(X_i)?

A: dom(X_i) = {1, 2, 3, 4, 5, 6}

Q: Is X_i discrete or continuous?

A: Discrete

Variable Assignments

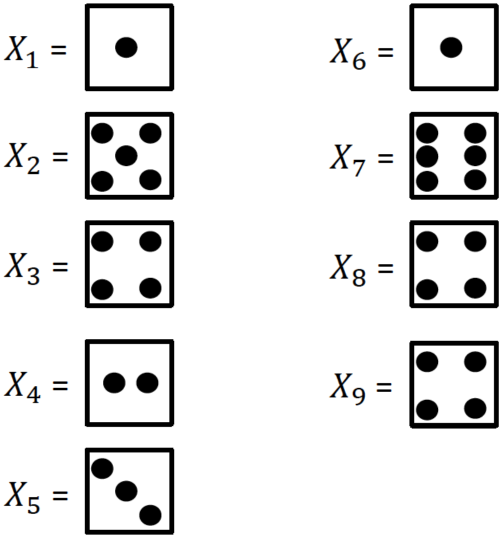
Given N variables $\mathbf{X} = \{X_1, X_2, \ldots, X_N\}$

- An assignment is a setting of a subset X' of those variables
  - Total assignment: $X' = \boldsymbol{X}$
  - Partial assignment: $X' \neq \boldsymbol{X}$
- A **possible world** is a possible way the world (the real world or some imaginary world) could be

Full vs. Partial Assignment Example

Let’s say there are N=9 rolls of the same die

Full assignment

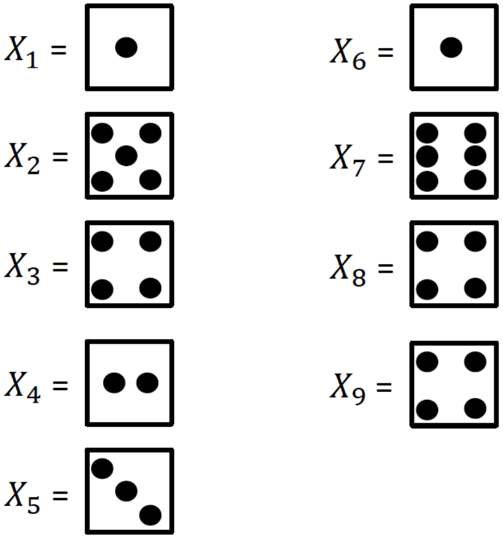


Partial assignment

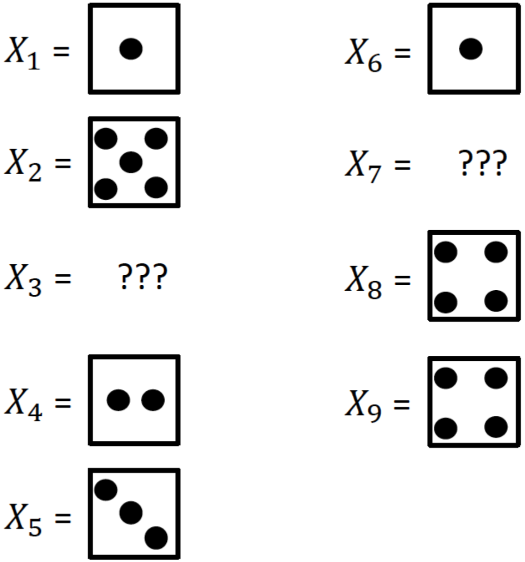
Full vs. Partial Assignment Example

Let's say there are N=9 rolls of the same die

Full assignment



Partial assignment



Thinking About Possible Worlds

Let's say there are N variables. How many possible worlds are there if:

- Each variable's domain is of size 2?
- Each variable's domain is of size 10?
- Each variable's domain is uncountably infinite (the real numbers)?

Rethinking Problem Space

- Reasoning explicitly in terms of states
  - How do you define relation between components (features) in a state?
- Typically, better to describe states in terms of **features**
  - reason in terms of these features
- Features are described using **variables**.
- Features are not independent and there are **hard constraints**

First-Order Logic (FOL)

- Variables: $X_1, X_2, \ldots, X_N$
- Constants: 1, 2, 3, 4, … 6
- Connectives: ¬, ∧ , ∨, =>, ⇔, Type equation here.
- Equality: =
- Quantifiers: ∀, ∃

First-Order Logic (FOL)

- ¬ means ‘not’
- ⋀ means ‘and’
- ⋁ means ‘or’

- R1 = A ⋀ B means A ‘and’ B
  R1 = true
    A = ?
    B = ?
- R2 = A ⋁ B means A ‘or’ B
  R2 = true
    A = ?
    B = ?
- R3 = A ⋀ B ⋀ ¬C means A ‘and’ B ‘and’ ‘not’ C
  R3 = true
    A = ?
    B = ?
    C= ?

Constraints

Many possible worlds… but are all of those possible worlds "possible?"

Constraint: a specification of allowed / disallowed combinations of assignments to individual variables

- **Scope**: the set of variables involved in the constraint
- **Relation**: Boolean function on the scope that indicates if the constraint is satisfied

Constraints

Many ***possible worlds***… but are all of those possible worlds "possible?"

Constraint: a specification of allowed / disallowed combinations of assignments to individual variables

- **Scope**: the set of variables involved in the constraint
- **Relation**: Boolean function on the scope that indicates if the constraint is satisfied

Scheduling example (4.7)

A, B, C are variables representing dates of events

Each has possible values {Jan, Feb, March, April}

"A can't happen later than B; and B must happen in January or February; and B must be before C; and either A and B can't happen at the same time, or C can't occur in April"

Constraints

Many **possible worlds**… but are all of those possible worlds "possible?"

Constraint: a specification of allowed / disallowed combinations of assignments to individual variables

- **Scope**: the set of variables involved in the constraint
- **Relation**: Boolean function on the scope that indicates if the constraint is satisfied

Scheduling example (4.7)

A, B, C are variables representing dates of events

Each has possible values {Jan, Feb, March, April}

"A can't happen later than B; and B must happen in January or February; and B must be before C; and either A and B can't happen at the same time, or C can't occur in April"

Constraints

Many **possible worlds**… but are all of those possible worlds "possible?"

Constraint: a specification of allowed / disallowed combinations of assignments to individual variables

- **Scope**: the set of variables involved in the constraint
- **Relation**: Boolean function on the scope that indicates if the constraint is satisfied

$$A \leq B \quad \wedge$$
$$B < \text{March} \wedge$$
$$B < C \wedge$$
$$A \neq B \quad \vee C < \text{April}$$

Scheduling example (4.7)

A, B, C are variables representing dates of events

Each has possible values {Jan, Feb, March, April}

"A can't happen later than B; and B must happen in January or February; and B must be before C; and either A and B can't happen at the same time, or C can't occur in April"

Constraints

Many possible worlds… but are all of those possible worlds "possible?"

Constraint: a specification of allowed / disallowed combinations of assignments to individual variables

- **Scope**: the set of variables involved in the constraint
- **Relation**: Boolean function on the scope that indicates if the constraint is satisfied

Scheduling example (4.7)

A, B, C are variables representing dates of events

Each has possible values {Jan, Feb, March, April}

"A can't happen later than B; and B must happen in January or February; and B must be before C; and either A and B can't happen at the same time, or C can't occur in April"

$A \leq B \quad \land$

$B < \text{March} \land$

$B < C \land$

$A \neq B \quad \lor C < \text{April}$

Scope ({A, B})

Constraints

Many possible worlds… but are all of those possible worlds "possible?"

Constraint: a specification of allowed / disallowed combinations of assignments to individual variables

- **Scope**: the set of variables involved in the constraint
- **Relation**: Boolean function on the scope that indicates if the constraint is satisfied

Scheduling example (4.7)

A, B, C are variables representing dates of events

Each has possible values {Jan, Feb, March, April}

"A can't happen later than B; and B must happen in January or February; and B must be before C; and either A and B can't happen at the same time, or C can't occur in April"

$$A \le B \quad \wedge$$

$$B < \text{March} \wedge$$

$$B < C \wedge$$

$$A \neq B \quad \vee C < \text{April}$$

Scope ({A, B})

Relation (≤)

Constraints

Many possible worlds… but are all of those possible worlds "possible?"

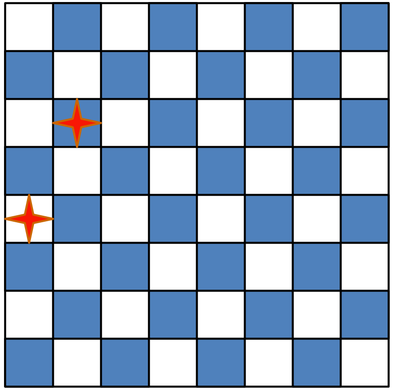
Constraint: a specification of allowed / disallowed combinations of assignments to individual variables

- **Scope**: the set of variables involved in the constraint
- **Relation**: Boolean function on the scope that indicates if the constraint is satisfied

Constraints are **satisfied** (an assignment that makes all constraints TRUE) or **violated**

Motivating example: 8 Queens

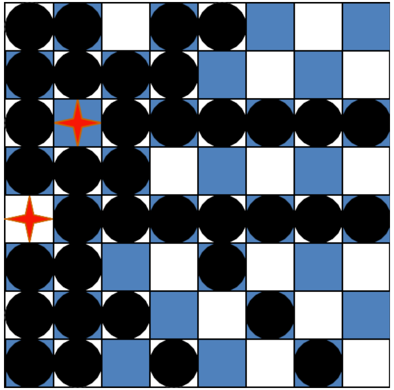
Place 8 queens on a chess board such That none is attacking another.



Generate-and-test, with no redundancies ➔ "only" 8^8 combinations

8**8 is 16,777,216

Motivating example: 8-Queens



After placing these two queens, it's trivial to mark the squares we can no longer use

What more do we need for 8 queens?

- Not just a successor function and goal test
- But also
  - a means to propagate constraints imposed by one queen on others
  - an early failure test

→ Explicit representation of constraints and constraint manipulation algorithms

Informal definition of CSP

- **CSP** (Constraint Satisfaction Problem), given

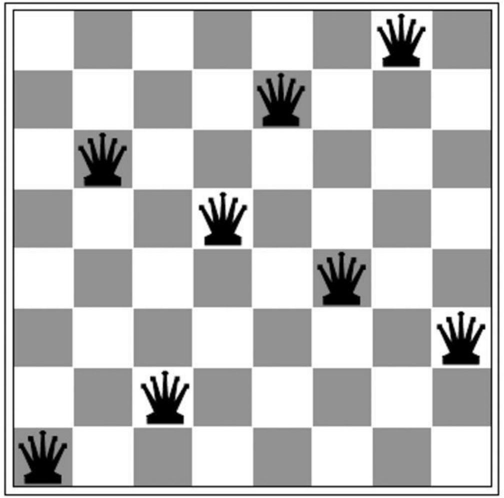
  (1) finite set of variables

  (2) each with domain of possible values (often finite)

  (3) set of constraints limiting values variables can take
- **Solution:** assignment of a value to each variable such that all constraints are satisfied
- **Possible tasks:** decide if solution exists, find a solution, find all solutions, find *best solution* according to some metric (objective function)

Example: 8-Queens Problem

- What are the variables?
- What are the variables domains, i.e., sets of possible values
- What are the constraints between (pairs of) variables?



Example: 8-Queens Problem

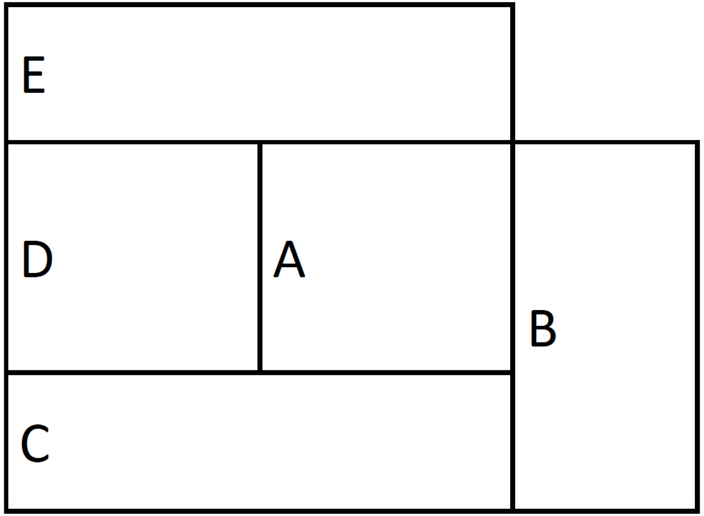
- Eight variables Qi, i = 1..8 where Qi is the row number of queen in column i
- Domain for each variable {1,2,…,8}
- Constraints are of the forms:
  - No queens on same row

    $Qi = k \Rightarrow Qj \neq k$ for $j = 1..8, j \neq i$
  - No queens on same diagonal

    $Qi = rowi, Qj = rowj \Rightarrow |i-j| \neq |rowi-rowj|$ for $j = 1..8, j \neq i$

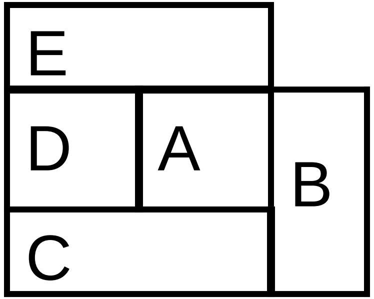
Example: Map coloring

Color this map using three colors (red, green, blue) such that no two adjacent regions have the same color

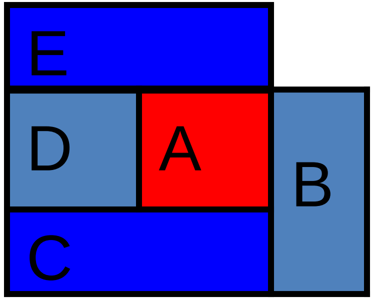


Map coloring

- Variables: A, B, C, D, E all of domain RGB
- Domains: RGB = {red, green, blue}
- Constraints: A≠B, A≠C, A≠E, A≠D, B≠C, C≠D, D≠E
- A solution: A=red, B=green, C=blue, D=green, E=blue



=>



Example: SATisfiability

- Given a set of logic propositions containing variables, find an assignment of the variables to {false, true} that satisfies them
- For example, the two clauses:
  - $(A \lor B \lor \neg C)$
  - $(\neg A \lor D)$

are both made true (i.e. satisfied) by assigning

A = false, B = true, C = false, D = false

- Satisfiability known to be NP-complete
- $\Rightarrow$ worst case, solving CSP problems requires exponential time

Real-world problems

CSPs are a good match for many practical problems that arise in the real world

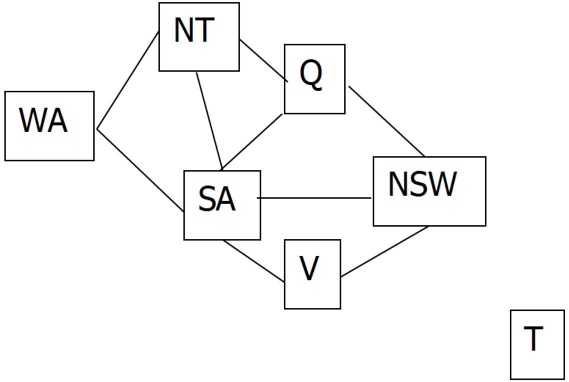
- Scheduling
- Temporal reasoning
- Building design
- Planning
- Optimization/satisfaction
- Vision
- Graph layout
- Network management
- Natural language processing
- Molecular biology / genomics
- VLSI design

Definition of a constraint network (CN)

A constraint network (CN) consists of

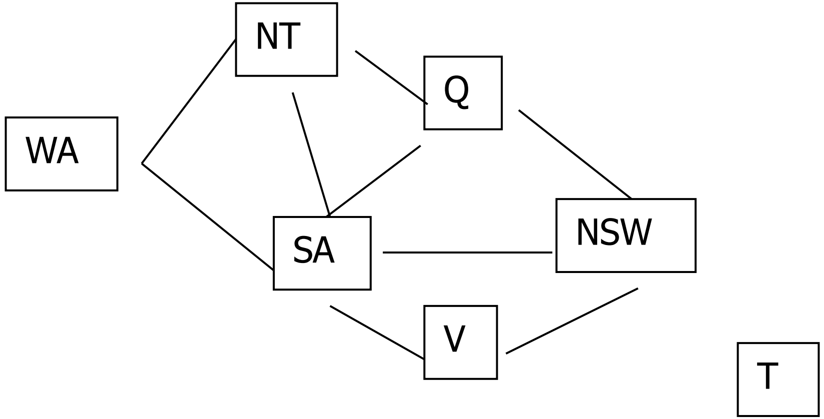
- Set of variables $X = \{x_1, x_2, \ldots x_n\}$
  - with associate domains $\{d_1, d_2, \ldots d_n\}$
  - domains are typically finite
- Set of constraints $\{c_1, c_2 \ldots c_m\}$ where
  - each defines a predicate that is a relation over a particular subset of variables (X)
  - e.g., C_i involves variables $\{X_{i1}, X_{i2}, \ldots X_{ik}\}$ and defines the relation $R_i \subseteq D_{i1} \times D_{i2} \times \ldots D_{ik}$

Running example: coloring Australia



- Seven variables: {WA, NT, SA, Q, NSW, V, T}
- Each variable has same domain: {red, green, blue}
- No two adjacent variables can have same value: WA≠NT, WA≠SA, NT≠SA, NT≠Q, SA≠Q, SA≠NSW, SA≠V, Q ≠NSW, NSW ≠V

Unary & binary constraints most common



- Two variables are adjacent or neighbors if connected by an edge or an arc
- Possible to rewrite problems with higher-order constraints as ones with just binary constraints

Formal definition of a CN

- Instantiations
  - An instantiation of a subset of variables S is an assignment of a value (in its domain) to each variable in S
  - An instantiation is legal iff it violates no constraints
- A solution is a legal instantiation of all variables in the network

Typical tasks for CSP

- Possible solution related tasks:
  - Does a solution exist?
  - Find one solution
  - Find all solutions
  - Given a metric on solutions, find best one
  - Given a partial instantiation, do any of above
- Transform the constraint network into an equivalent one that's easier to solve

Binary CSP

- A **binary CSP** is one where all constraints involve two variables (or just one variable)
- Any non-binary CSP can be converted into a binary CSP by introducing additional variables
- Binary CSPs represented as a constraint graph, with a node for each variable and an arc between two nodes iff there's a constraint involving them
  - Unary constraints appear as self-referential arcs

Brute Force methods

- Finding a solution by a brute force search is easy
  - Generate and test is a *weak method*
  - Just generate potential combinations and test each
- Potentially very inefficient
  - With n variables where each can have one of 3 values, there are 3^n possible solutions to check
- There are ~190 countries in the world, which we can color using four colors
- 4^{190} is a big number!

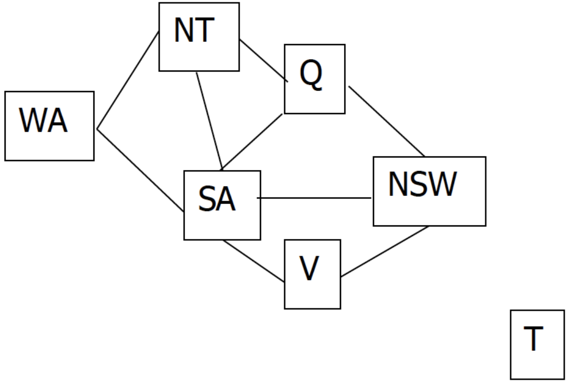
```
solve(A,B,C,D,E) :-
  color(A),
  color(B),
  color(C),
  color(D),
  color(E),
  not(A=B),
  not(A=B),
  not(B=C),
  not(A=C),
  not(C=D),
  not(A=E),
  not(C=D).

color(red).
color(green).
color(blue).
```

generate

test

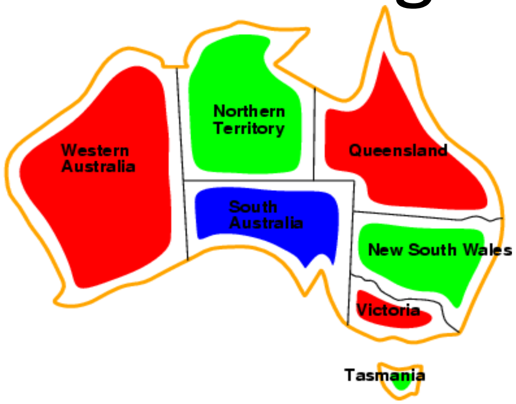
4**190 is 2462625387274654950767440006258975862817483704404090416746768337765357610718575663213391640930307227550414249394176L

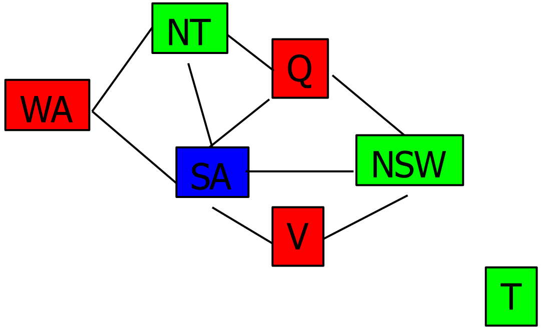
Running example: coloring Australia



- Seven variables: {WA, NT, SA, Q, NSW, V, T}
- Each variable has same domain: {red, green, blue}
- No two adjacent variables can have same value: WA≠NT, WA≠SA, NT≠SA, NT≠Q, SA≠Q, SA≠NSW, SA≠V, Q ≠NSW, NSW ≠V

Running example: coloring Australia

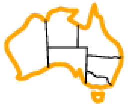




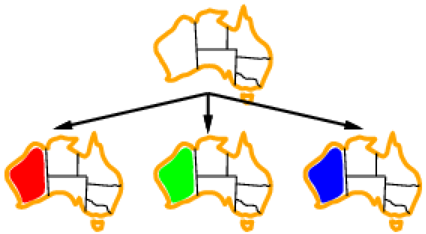
- Solutions: complete & consistent assignments
- Here is one of several solutions
- For generality, constraints can be expressed as relations, e.g., describe WA ≠ NT as

{(red,green), (red,blue), (green,red), (green,blue), (blue,red),(blue,green)}

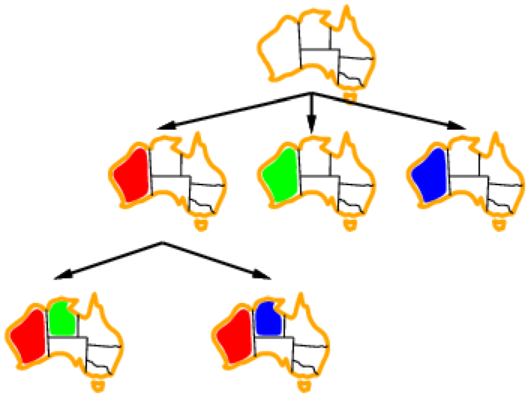
Backtracking example



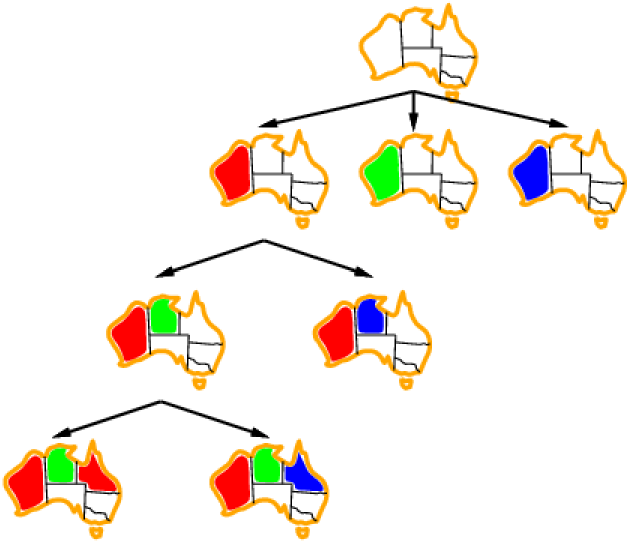
Backtracking example



Backtracking example



Backtracking example



Basic backtracking algorithm

CSP-backtracking(PartialAssignment a)

- If a is complete then return a
- X ← select an unassigned variable
- D ← select an ordering for the domain of X
- For each value v in D do

  If v consistent with a then

  - Add (X=v) to a
  - result ← CSP-BACKTRACKING(a)
  - If result ≠ failure then return result
  - Remove (X= v) from a
- Return failure

Start with CSP-BACKTRACKING({})

Note: depth first search; can solve n-queens problems for n ~ 25

Problems with Backtracking

- Thrashing: keep repeating the same failed variable assignments
- Things that can help avoid this:
  - Consistency checking
  - Intelligent backtracking schemes
- Inefficiency: can explore areas of the search space that aren't likely to succeed
  - Variable ordering can help

Improving backtracking efficiency

Here are some standard techniques to improve the efficiency of backtracking

- Can we detect inevitable failure early?
- Which variable should be assigned next?
- In what order should its values be tried?