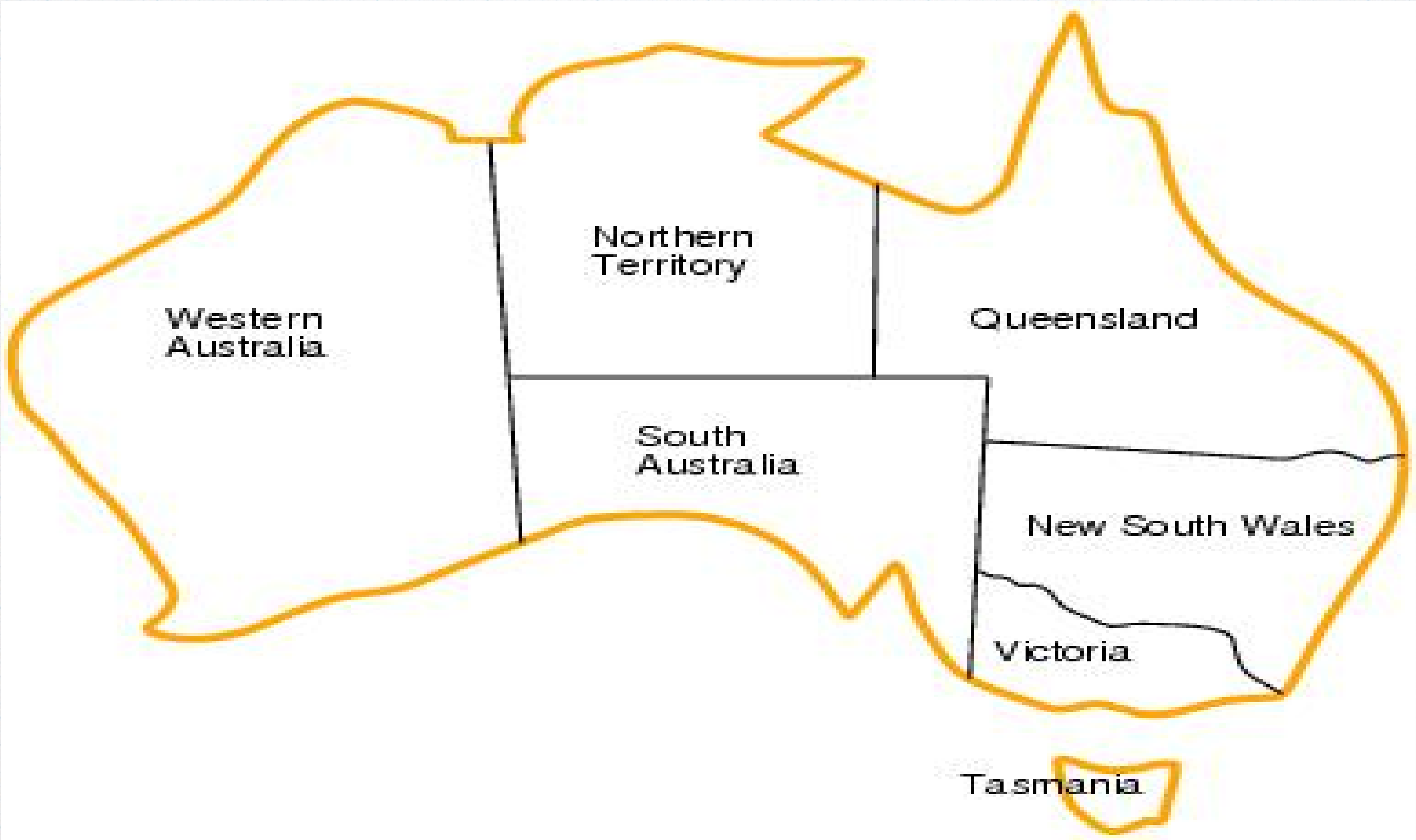
Project One

Graph Coloring: A Constraint Satisfaction Problem

Map Colouring



http://www.uspanteco.org/preschool.htm

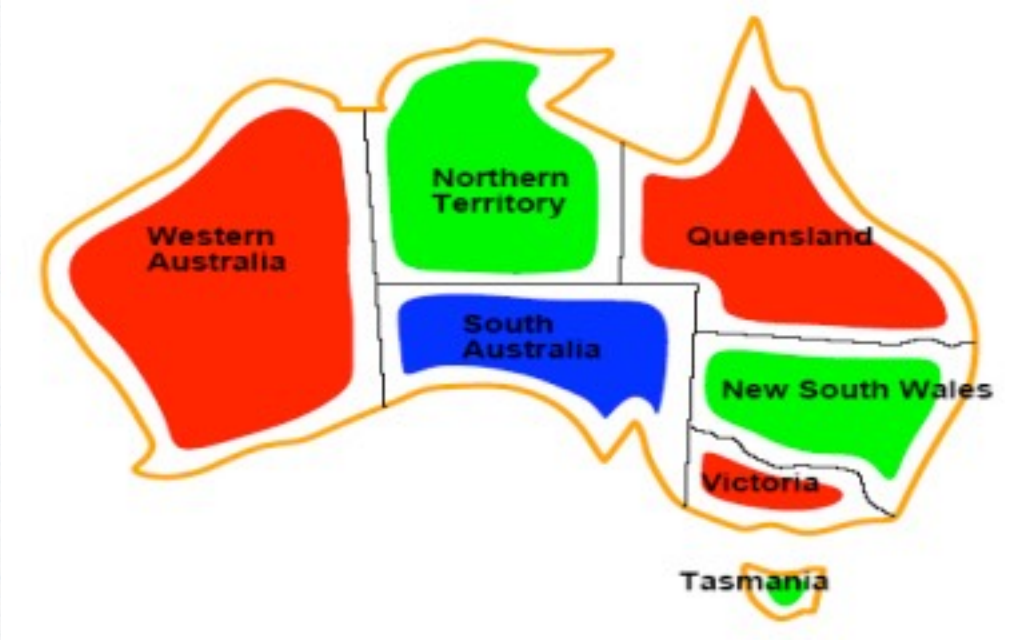
An example CSP
(Constraint Satisfaction Problem)

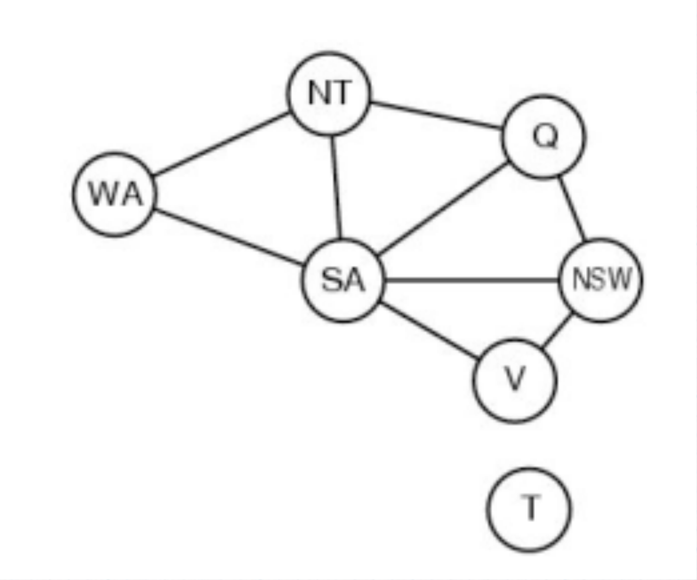
* Variables (Regions): WA, NT, Q, NSW, V, SA, T
* Domains (Colours): {red, green, blue}
* Constraints: Adjacent regions must have different colours.

Solution

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

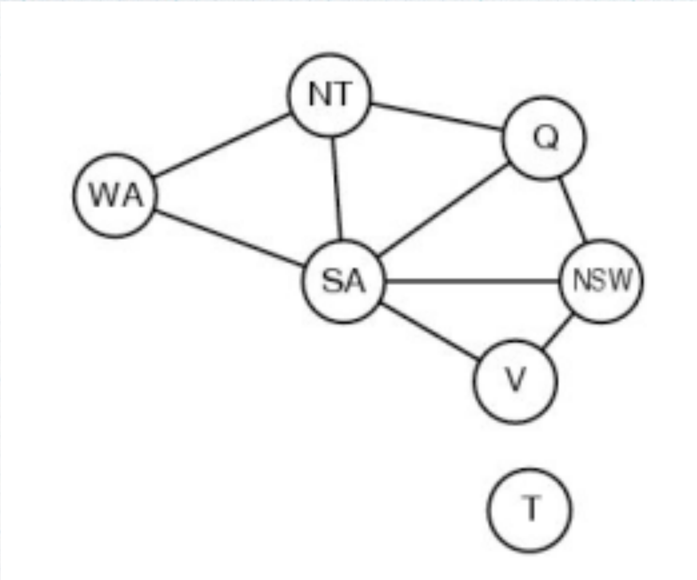
Representation: Graph





Benefits of Representation

Constraint Graph



* Nodes are variables
* Edges show constrains

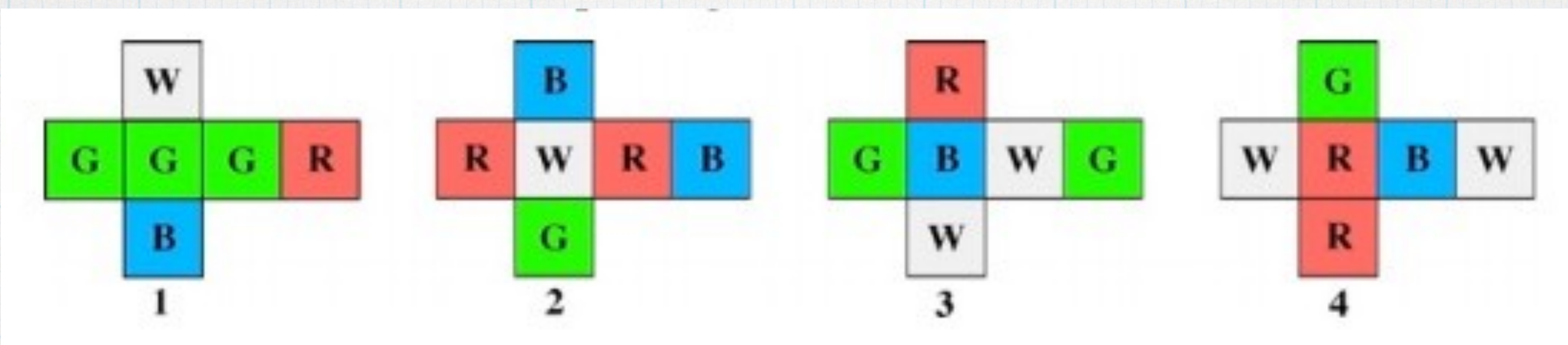
* Standard data structure
* Standard algorithms

More about CSP: Variables

* Discrete Variables
  * Finite Domains
    * (Boolean satisfiability, Map Coloring)
  * Infinite Domains
    * (Job Scheduling - start/ending dates)
* Continuous Variables

More about CSP: Constraints

* Unary Constraints
  * e.g. SA != green (involves one var)
* Binary Constraints
  * e.g. SA != WA (involves two vars)
* Higher Order
  * e.g. cryptarithmetic problems
* Preference (soft constrains)
  * e.g. red is better than blue (use costs)



Patented by F. Schossow 1900, popularized by Parker Brothers 1967

Instant Insanity represented as a Graph

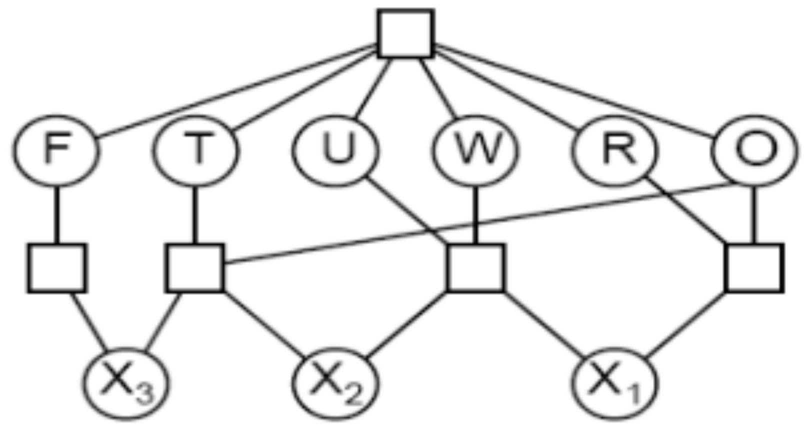
1. Draw four *vertices*, and label them *green*, *white*, *blue*, *red*; it doesn't matter how they are arranged (after solving, the figure below was re-drawn to look nicer).
2. Number the cubes from 1 to 4 and for each of the three pairs of opposite faces on each cube, draw an *edge* between the two vertices of the corresponding colors and label that edge by that cube number (a total of 12 edges).
3. Look for a *Hamilton cycle* (that passes through each vertex once) with a different label on each edge; this cycle is shown with the thick edges below (it is also ok to use set of smaller disjoint cycles but that doesn't help here).
4. Find a second Hamilton cycle (or set of cycles) with a different label on each edge, that does not uses any of the edges in the first cycle; this cycle is shown with the hashed edges in the figure below.
5. Traverse the thick edge cycle to set the top edges (green to blue set the top and bottom of cube 3, blue to white to set the top and bottom of cube 2, white to red to set the top and bottom of cube 4, and red to green to set the top and bottom of cube 1).
6. Set the front/back edges with the hashed cycle by rotating each cube (without changing the top and bottom).

Solution on wikipedia

http://www.cs.brandeis.edu/~storer/JimPuzzles/MATCH/InstantInsanity/InstantInsanity.pdf

Cryptarithmetic

$$\begin{array}{cccc} & T & W & O \\ + & T & W & O \\ \hline F & O & U & R \end{array}$$



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)

$O + O = R + 10 \cdot X_1$, etc.

For n-ary constraints,
add nodes that respresent combinations of values.

CSP as Search

* Initial State: all variables unassigned
* Successor: Assign a value to unassigned variable without violating constraints
* Goal: Assign every variable a value
* Path: Assignment order unneeded; just print result

CSP as Search

- All solutions are found at depth N
  - can (should) use depth first search
- Branching factor is ND (variables * domain) at top level, or (N-L)D at level L.
- This means $N!D^N$ leaves; D^N assignments

Backtracking

* Depth first search
* Choose a value for one variable;
  backtrack if there are no available values
* Uniformed (Blind) Search
  * Generally, poor performance

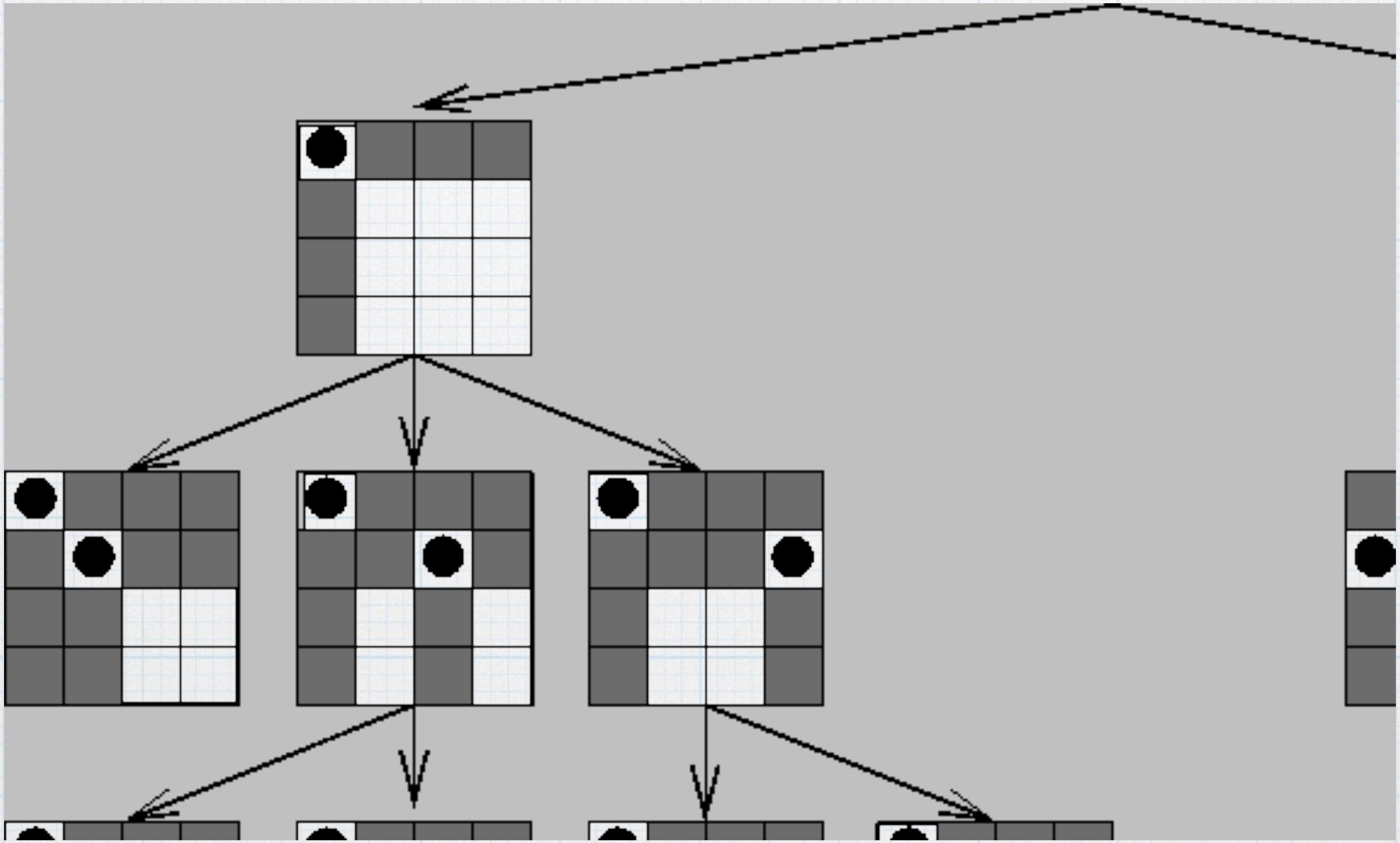
Backtracking Search

```
function BACKTRACKING-SEARCH(csp)
   return RECURSIVE-BACKTRACKING({} , csp)

function RECURSIVE-BACKTRACKING(assignment, csp)
   if assignment is complete then return assignment
   var := SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp],assignment,csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
     if value is consistent with assignment
          according to CONSTRAINTS[csp] then
        add {var = value} to assignment
        result := RECURSIVE-BACTRACKING(assignment, csp)
        if result != failure  then return result
        remove {var = value} from assignment
   return failure
```

[Russel & Norvig 2003]

ex: N-Queens Problem



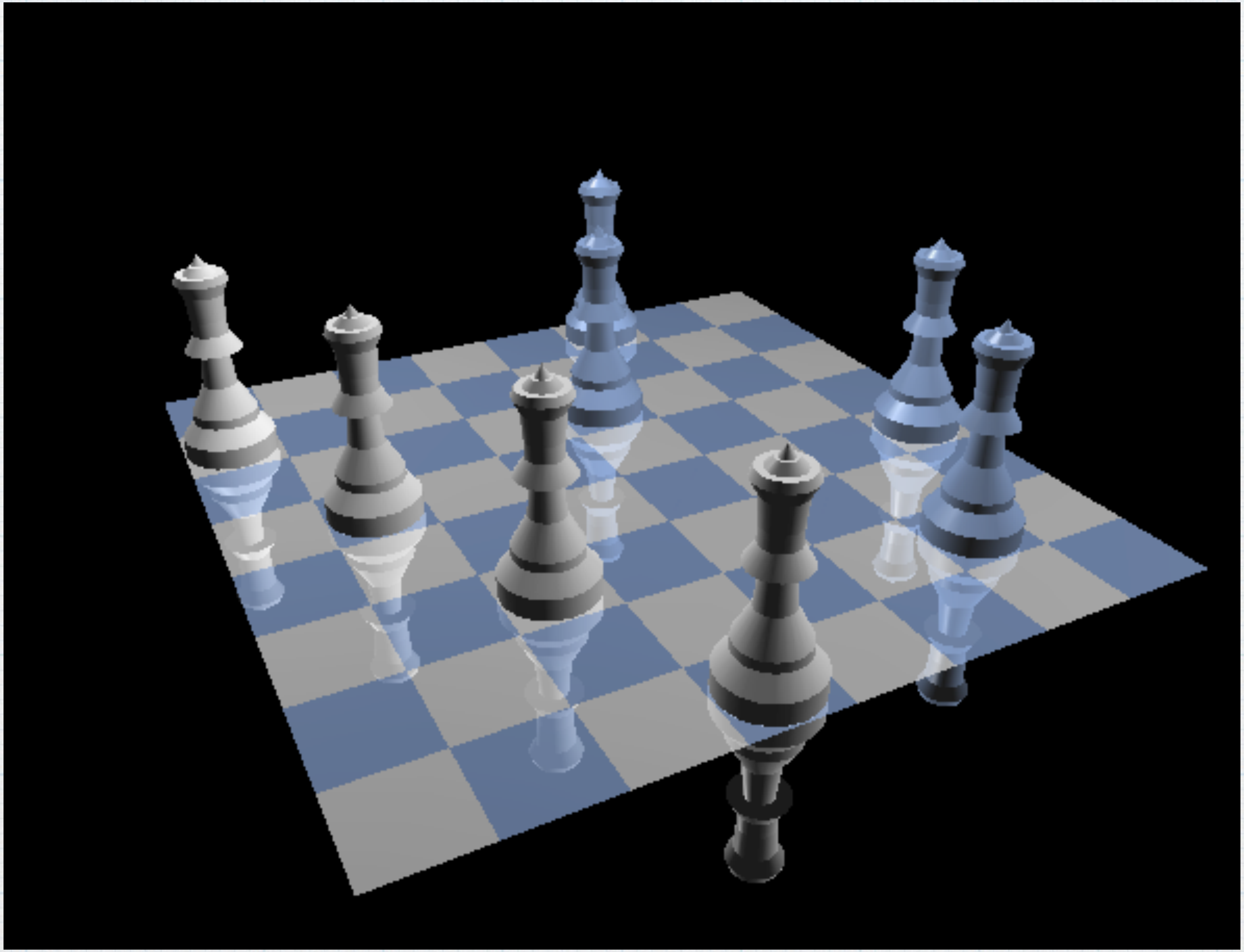
http://clip.dia.fi.upm.es/~logalg/slides/7_clp/img68.png

N-Queens as a CSP

- Chessboard puzzle
- e.g. when n = 8…
  - place 8 queens on a 8x8 chessboard so that no two attack each other
- Variable x_i for each row i of the board
- Domain = {1, 2, 3 … , n} for position in row
- Constraints are:
  - $x_i \neq x_j$ queens not in same column
  - $x_i - x_j \neq i-j$ queens not in same SE diagonal
  - $x_j - x_i \neq i - j$ queens not in SW diagonal

http://www.dcs.st-and.ac.uk/~ipg/AI/Lectures/Constraints1/img007.JPG

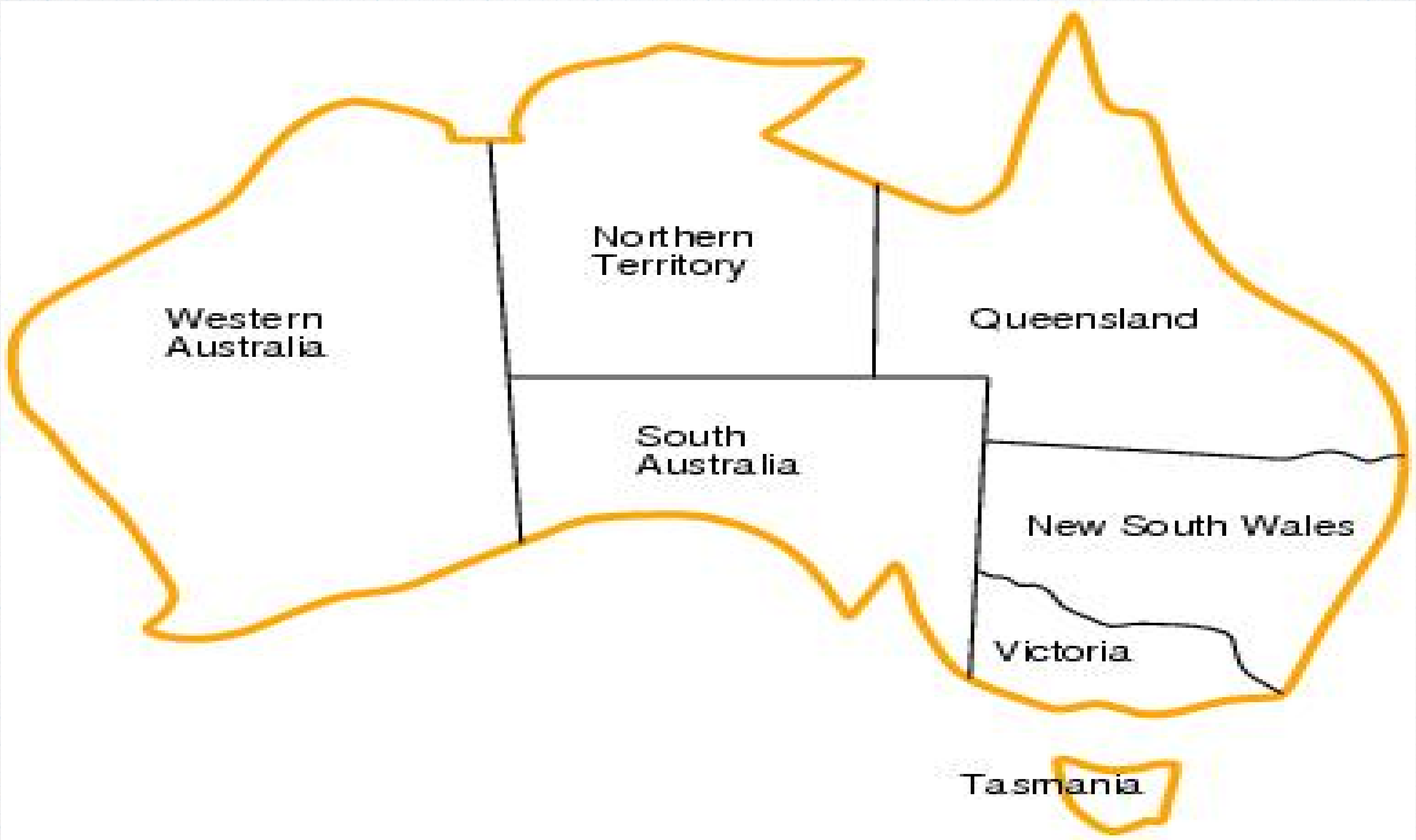
Often, the most difficult part of solving a CSP is formulating the problem.



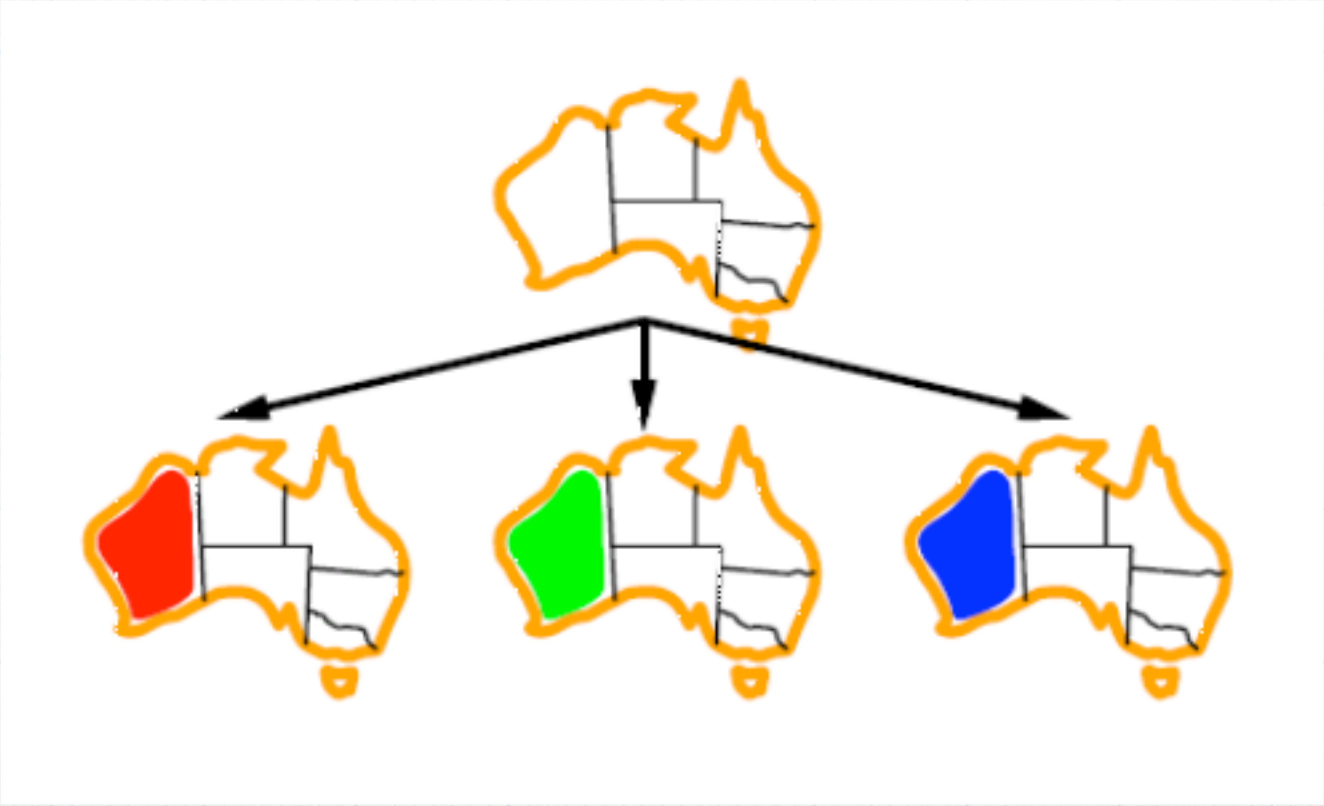
http://www.sirgalahad.org/paul/sw/winlock/img/queens.png

(n-queens soln)

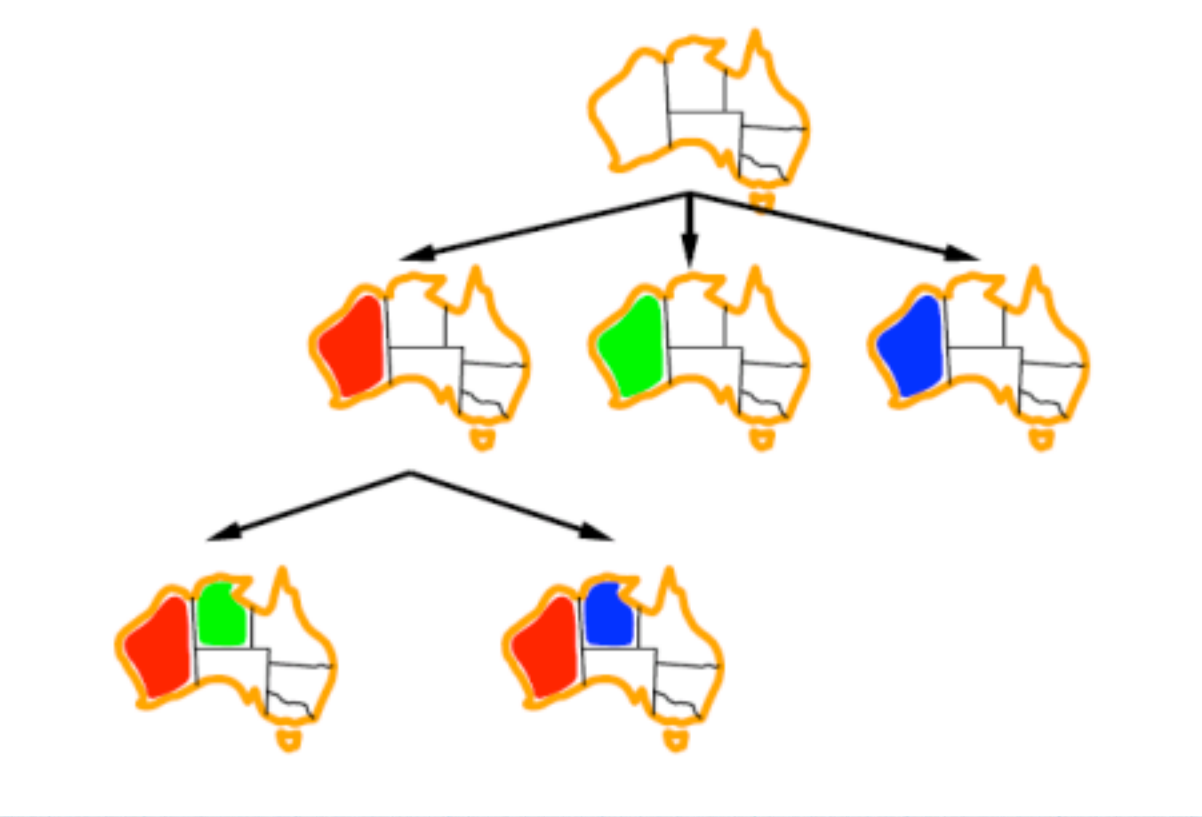
Map Colouring



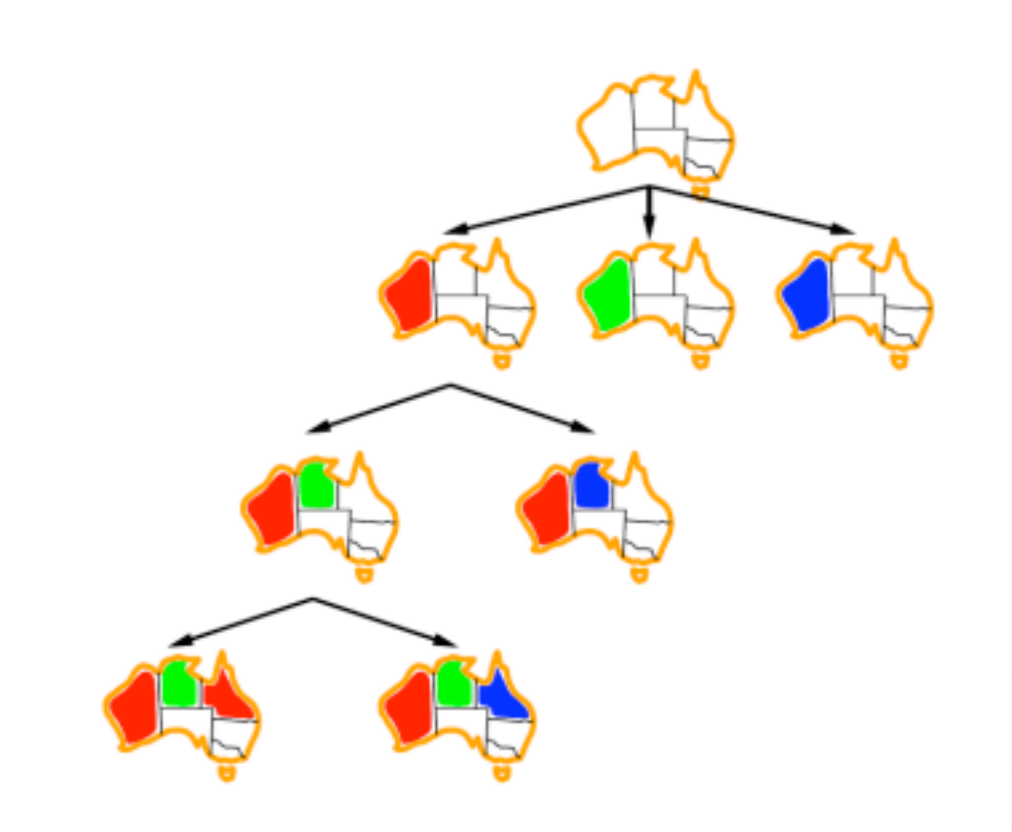
Example



Example



Example



Your Assignment

* Read a graph (map) from data file
* Color it with 4 colors
* Check online problem for particulars

input data file - usa.txt

```
al,fl,ms,tn,ga
ar,la,tx,ok,mo,tn,ms
az,ca,nv,ut,nm
ca,az,nv,or
co,wy,ut,nm,ok,ks,ne
ct,ny,ma,ri
de,md,pa,nj
fl,al,ga
⋮
```

Why Four Colors?

The four color theorem states that any plane separated into regions, such as a political map of the counties of a state, can be colored using no more than four colors in such a way that no two adjacent regions receive the same color. Two regions are called adjacent if they share a border segment, not just a point.

Conjectured in 1852 by Francis Guthrie.

http://en.wikipedia.org/wiki/Four_color_theorem

Why Four Colors?

It was not until 1976 that the four-color conjecture was proven by Kenneth Appel and Wolfgang Haken at the University of Illinois. They were assisted in some algorithmic work by J. Koch.

http://en.wikipedia.org/wiki/Four_color_theorem

J. Koch ?

Why Four Colors?

The proof reduced the infinitude of possible maps to 1,936 configurations which had to be checked one by one by computer. The work was independently double checked with different programs and computers. However, the proof was over 500 pages of hand written counter-counter-examples. The computer program ran for hundreds of hours.

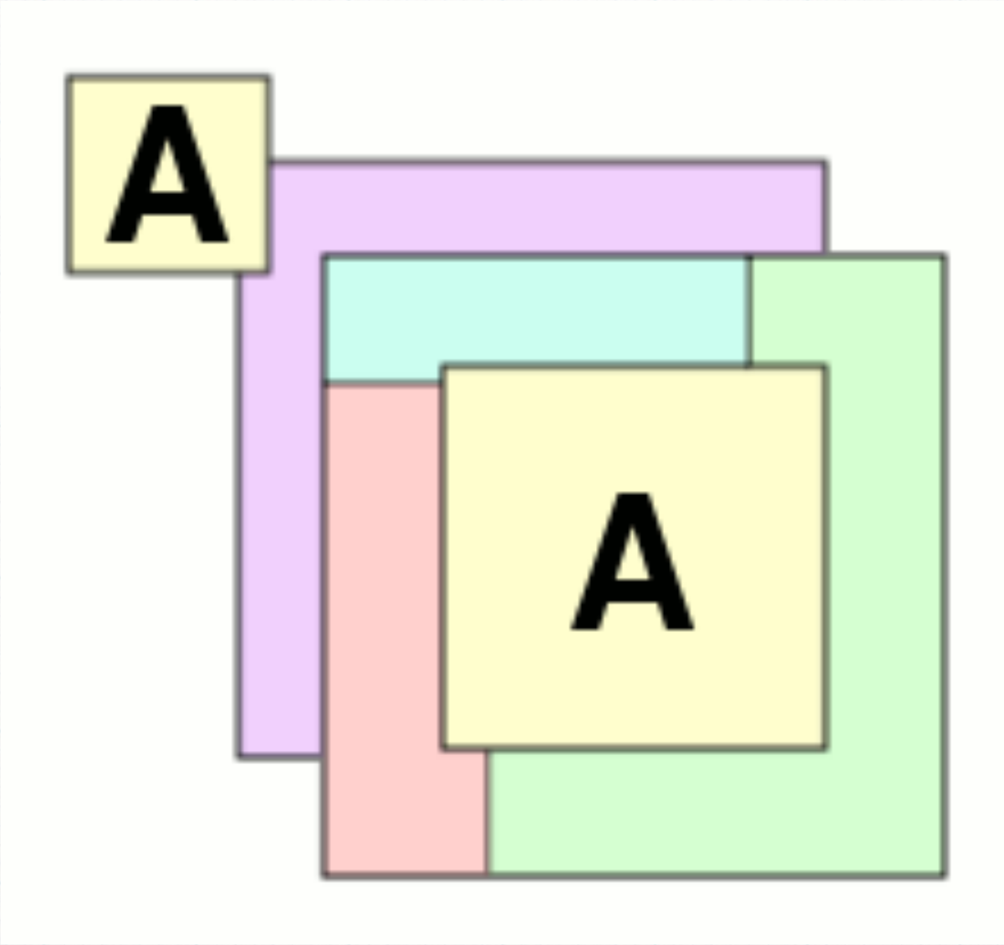
http://en.wikipedia.org/wiki/Four_color_theorem

Why Four Colors?

In 2004 Benjamin Werner and Georges Gonthier formalized a proof of the theorem inside the Coq theorem prover. This removes the need to trust the various computer programs used to verify particular cases — it is only sufficient to trust the Coq prover.

http://en.wikipedia.org/wiki/Four_color_theorem

Contiguous, Planar Maps



http://en.wikipedia.org/wiki/Four_color_theorem

http://www.barron.palo-alto.ca.us/history/graphics/crayons2.jpg

How Good is Backtracking?

* USA 4 coloring (>1,000,000)
* n-Queens 2 to 50 (>40,000,000)
* Zebra 3,859,0000

[Russel & Norvig 2003]

How Good is Backtracking?

* USA 4 coloring (>1,000,000)
* n-Queens 2 to 50 (>40,000,000)
* Zebra 3,859,0000

[Russel & Norvig 2003]

The Zebra Problem

There are five houses, each of a different color, inhabited by men of different nationalities, with different pets, drinks, and cigarettes. The Englishman lives in the red house. The Spaniard owns the dog. The ivory house is immediately to the left of the green house, where the coffee drinker lives. The milk drinker lives in the middle house. The man who smokes Old Golds also keeps snails. The Ukranian drinks tea. The Norwegian resides in the first house on the left. The Chesterfields smoker lives next door to the fox owner. The Lucky Strike smoker drinks orange juice. The Japanese man smokes Parliaments. The horse owner lives next to the Kools smoker, whose house is yellow. The Norwegian lives next to the blue house.

1. Who drinks water?
2. Who owns the Zebra?
3. Is there only a unique solution to this problem?

Often, the most difficult part of solving a CSP is formulating the problem.

(now you are convinced!)

Backtracking Search

```
function BACKTRACKING-SEARCH(csp)
   return RECURSIVE-BACKTRACKING({} , csp)

function RECURSIVE-BACKTRACKING(assignment, csp)
   if assignment is complete then return assignment
   var := SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp],assignment,csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
     if value is consistent with assignment
           according to CONSTRAINTS[csp] then
         add {var = value} to assignment
         result := RECURSIVE-BACTRACKING(assignment, csp)
         if result != failure  then return result
         remove {var = value} from assignment
   return failure
```

[Russel & Norvig 2003]

Backtracking Search

```
function BACKTRACKING-SEARCH(csp)
   return RECURSIVE-BACKTRACKING({} , csp)

function RECURSIVE-BACKTRACKING(assignment, csp)
   if assignment is complete then return assignment
   var := SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp],assignment,csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment
           according to CONSTRAINTS[csp] then
         add {var = value} to assignment
         result := RECURSIVE-BACTRACKING(assignment, csp)
         if result != failure  then return result
         remove {var = value} from assignment
   return failure
```

[Russel & Norvig 2003]

Backtracking Search

```
function BACKTRACKING-SEARCH(csp)
   return RECURSIVE-BACKTRACKING({} , csp)

function RECURSIVE-BACKTRACKING(assignment, csp)
   if assignment is complete then return assignment
   var := SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp],assignment,csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment
          according to CONSTRAINTS[csp] then
        add {var = value} to assignment
        result := RECURSIVE-BACTRACKING(assignment, csp)
        if result != failure  then return result
        remove {var = value} from assignment
   return failure
```

[Russel & Norvig 2003]

Improve efficiency

* Which variable should be assigned next?
* In what order should its values be tried?
* Can we detect failure earlier?
* Can we take advantage of structure?

Improve efficiency

* Which variable should be assigned next?
* In what order should its values be tried?
* Can we detect failure earlier?
* Can we take advantage of structure?

$\Rightarrow$ Informed Search

Study Question

Design a representation
to solve the Zebra Problem.

Now Get To Work!

http://www.randypeterman.com/g2/albums/userpics/10001/normal_abby_crayons.jpg