Constraint Satisfaction

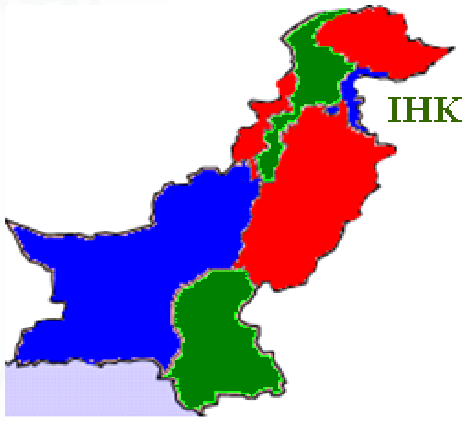
Dr. Fayyaz ul Amir Afsar Minhas
PIEAS Biomedical Informatics Research Lab
Department of Computer and Information Sciences
Pakistan Institute of Engineering & Applied Sciences
PO Nilore, Islamabad, Pakistan
http://faculty.pieas.edu.pk/fayyaz/

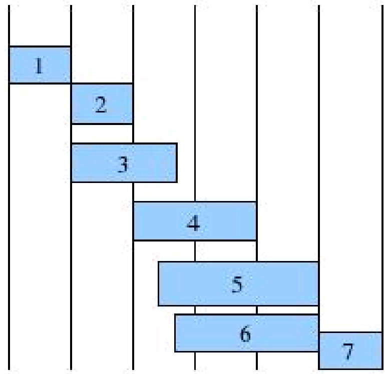
Outline

- Introduction to CSPs
- Backtracking Search for CSPs
  - Variable and Value Ordering
  - Constraint Propagation
  - Intelligent Backtracking
- Local Search Algorithms for CSPs
- Problem Structure

Constraint Satisfaction Problems

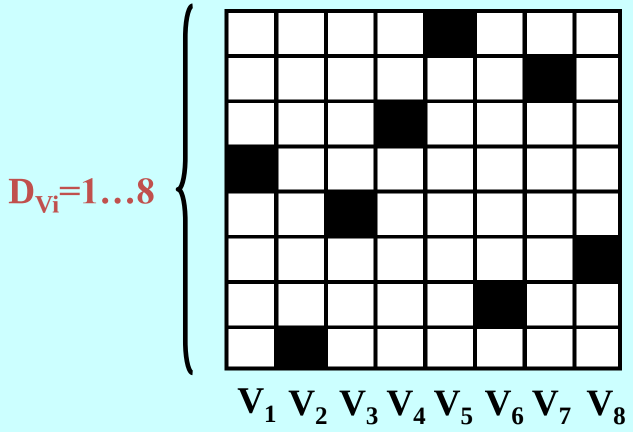
- CSPs are problems in which there are constraints on problem variables which need to be satisfied to result in a solution
  - Example
    - 8 queens puzzle
    - Map Coloring
    - Scheduling
      - Creation of a time table
      - Scheduling of Transport Facilities





Constraint Satisfaction Problems…

- Technically, a CSP comprises of the following
  - Finite set of variables: $V_1, V_2, \ldots, V_n$
  - Finite set of constraints: $C_1, C_2, \ldots, C_m$
  - Non-empty domain of possible values for each variable $D_{V1}, D_{V2}, \ldots D_{Vn}$
  - Each constraint C_i limits the values that variables can take, e.g., $V_1 \neq V_2$



$V_k \notin diag(V_i), \forall k \neq i$

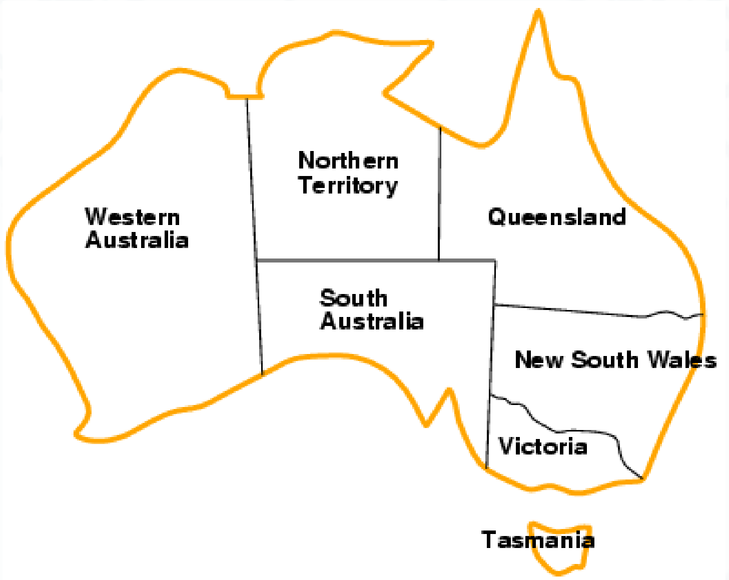
$diag(V_i)$ is the set of diagnoal points leading to V_i

$V_k \neq V_i, \forall k \neq i$

Constrained Satisfaction Problems: Terminology

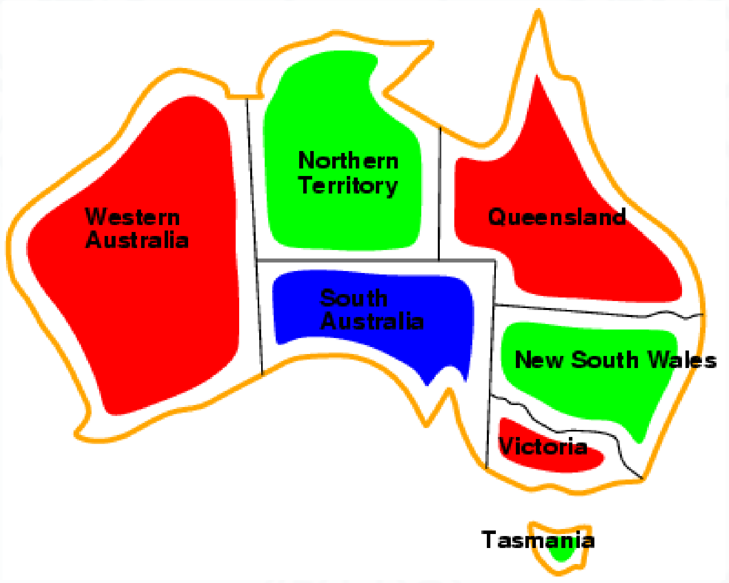
- State
  - An *assignment* of values to some or all variables
- *Consistent assignment*
  - An assignment that does not violate the constraints.
- Complete Assignment
  - An assignment is complete if all the variables in the CSP have been assigned
- A solution to a CSP is a complete and consistent assignment
- Some CSPs may require a solution that maximizes an objective function.

CSP: Example – Map Coloring



- Variables: *WA, NT, Q, NSW, V, SA, T*
- Domains: D_i = {red,green,blue}
- Constraints: adjacent regions must have different colors
  - e.g., WA ≠ NT

CSP: Example – Map Coloring…



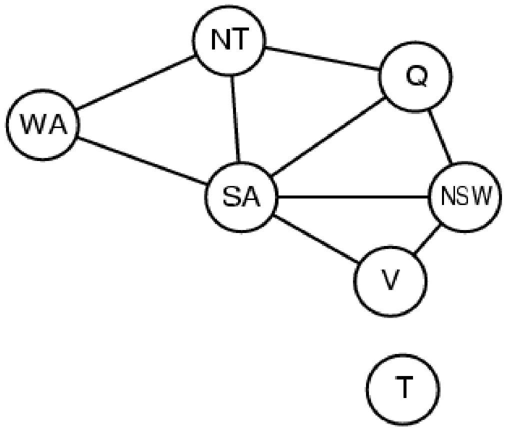
- Solutions are complete and consistent assignments,
  - WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green

Why move to CSPs?

- CSPs allow a standard representation pattern for a variety of problems
- Definition of a problem allows us to use general goal-test and successor functions
- No domain specific expertise is required in defining heuristics for a CSP

CSP: Terminology

- Binary CSP
  - Each constraint relates two variables
- Constraint Graphs
  - Shows each constraint in a CSP graphically as an arc connecting the variables (nodes) over which it is defined



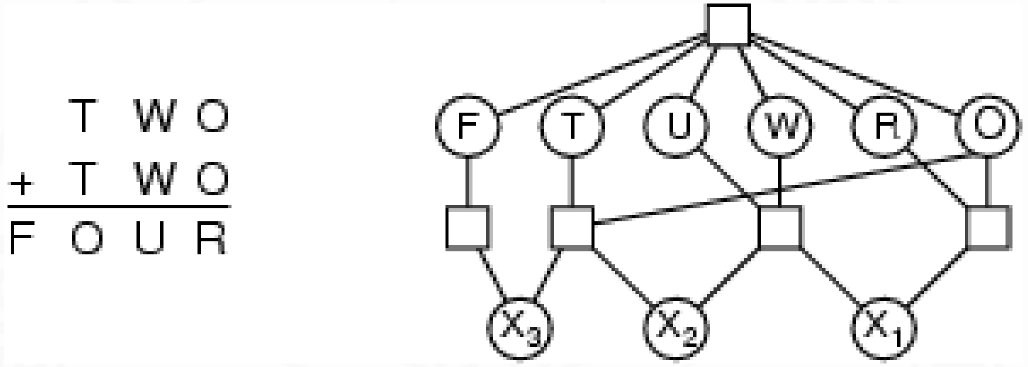
Varieties of CSPs

- Discrete variables
  - Finite domains
    - size d $\Rightarrow$ O (d^n) complete assignments
    - E.g. Boolean CSPs, include. Boolean 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. $StartJob_1 + 5 \leq StartJob_3$
    - Linear constraints: solvable, nonlinear: *un-decidable*
- Continuous variables
  - e.g. start/end times for Hubble Telescope observations
  - Linear constraints solvable in polynomial time by LP methods

Varieties of Constraints in CSPs

- Unary constraints involve a single variable
  - e.g. $SA \neq green$
- Binary constraints involve pairs of variables
  - e.g. $SA \neq WA$
- Higher-order constraints involve 3 or more variables
  - e.g. crypt-arithmetic column constraints
- Preference (soft constraints) e.g. *red* is better than *green*
  - Often represented by a cost for each variable assignment which leads to the creation of a constrained optimization problem

Example: Crypt-Arithmetic Puzzle



- 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)*
  - *$O + O = R + 10 \cdot X1$*
  - *$X1 + W + W = U + 10 \cdot X2$*
  - *$X2 + T + T = O + 10 \cdot X3$*
  - *$X3 = F$, $T \neq 0$, $F \neq 0$*

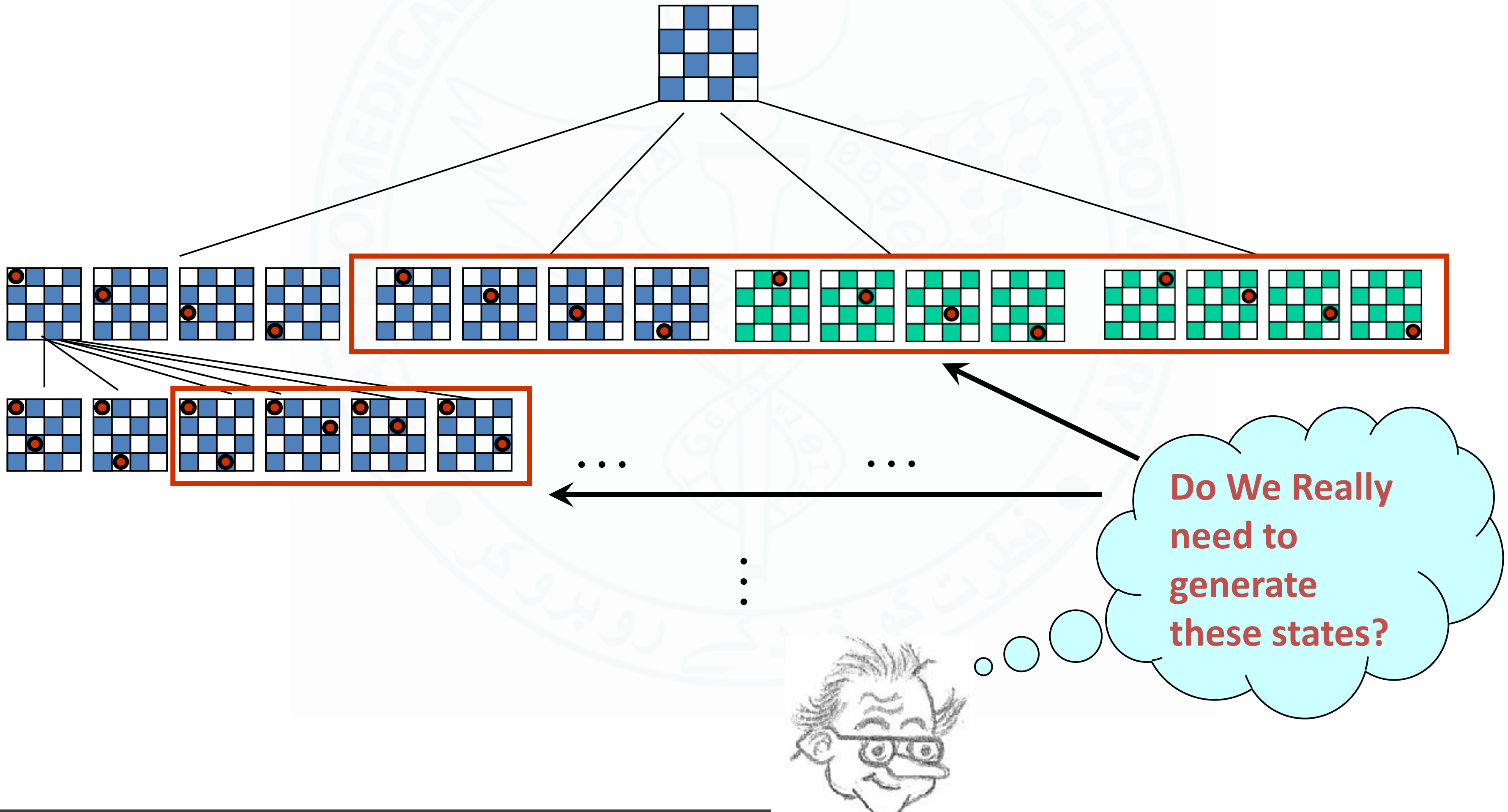
Standard search formulation

- A CSP can easily expressed as a standard search problem
- Incremental formulation
  - *Initial State*
    - The empty assignment { }
  - *Successor function*
    - Assign value to unassigned variable provided that there is no conflict
  - *Goal test*
    - The current assignment is complete
  - *Path cost*
    - As constant cost for every step

Standard search formulation…

- The solution appears at a depth n (when all the variables have assigned in an incremental manner)
  - Using DFS can offer an easier approach to the solution
- The path to the solution is irrelevant
- The branching factor, *b*, at a depth m is
  - $b = (n-m)d$
- Thus the total number of nodes being generated is: $n!d^n$
- But for CSPs the number of possible complete assignments is only d^n

Standard search formulation: Example



Standard search formulation…

- Thus standard search formulation of a CSP is not efficient
  - Does not consider the inherent commutative nature of CSPs!
    - [WA=red then NT=green] same as [NT=green then WA=red]
- Solution: Choose values for one variable at a time and backtrack when a variable has no legal values left to assign
  - This leads to Backtracking Search
    - Mother of CSP Solvers

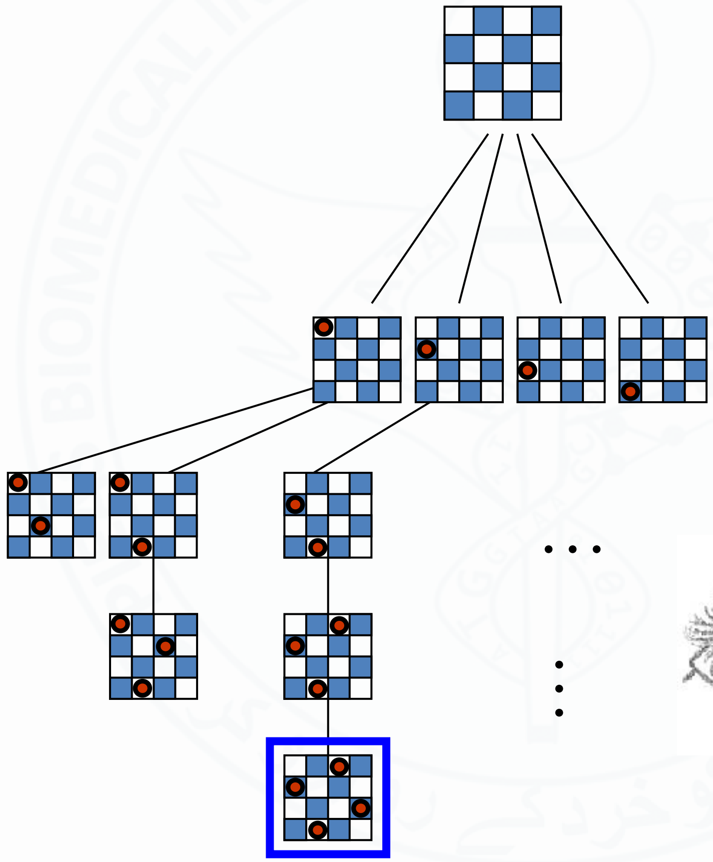
Backtracking Search

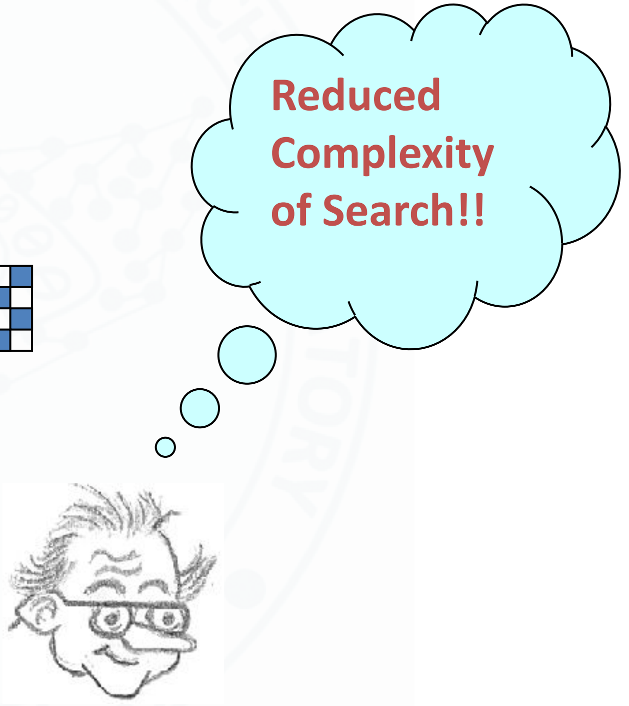
Assign V_1

Assign V_2

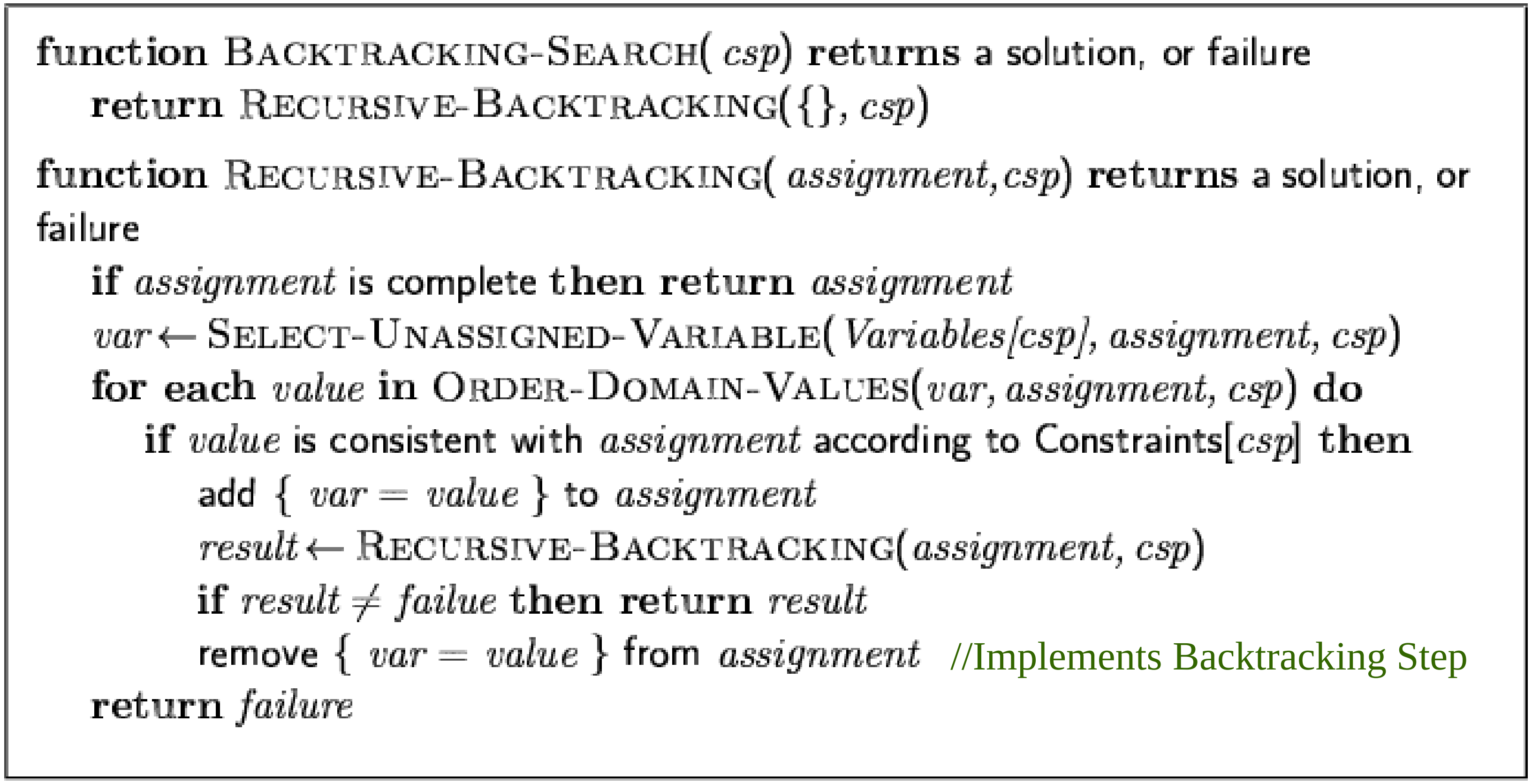
Assign V_3

Assign V_4





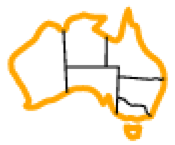
Backtracking Search



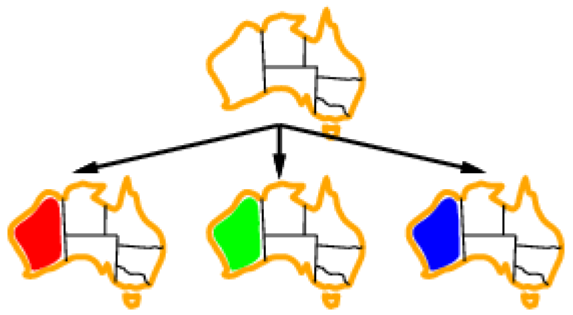
```
function BACKTRACKING-SEARCH( csp) returns a solution, or failure
    return RECURSIVE-BACKTRACKING({}, csp)

function RECURSIVE-BACKTRACKING( assignment,csp) returns a solution, or
failure
    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-BACKTRACKING(assignment, csp)
            if result ≠ failue then return result
            remove { var = value } from assignment   //Implements Backtracking Step
    return failure
```

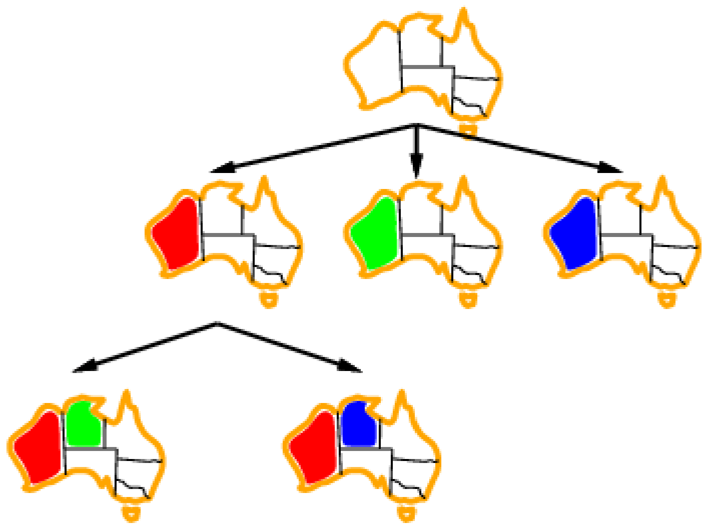
Backtracking example



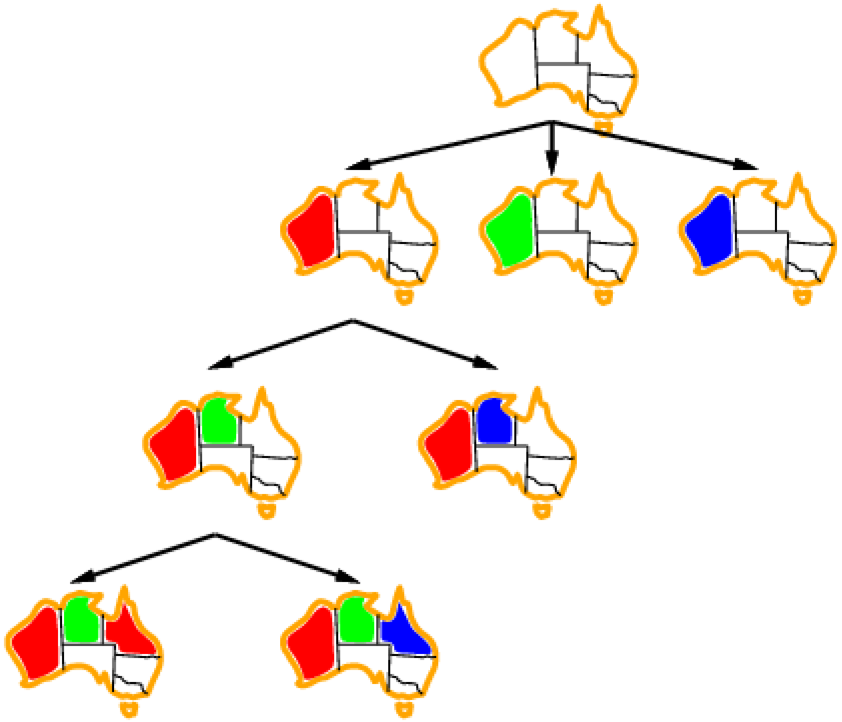
Backtracking example



Backtracking example



Backtracking example

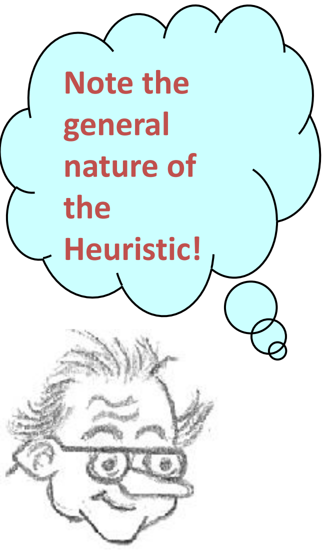


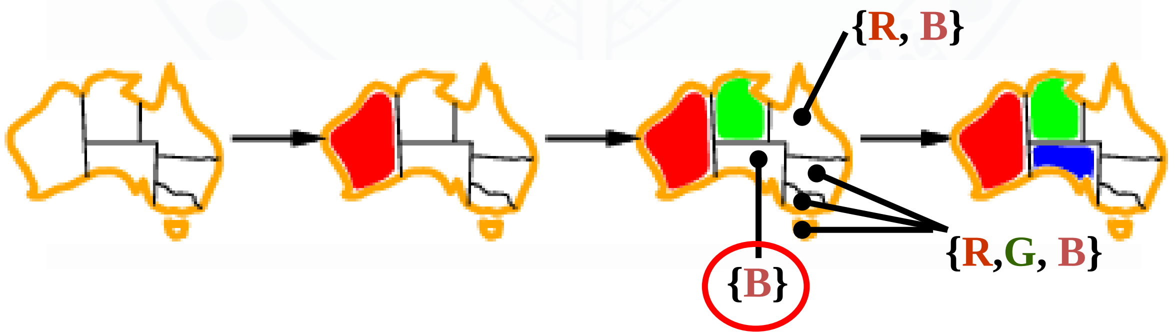
Backtracking: Improvements

- The Backtracking Algorithm given earlier is uninformed
- Can be improved by introducing some general heuristics in the following dimensions
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect inevitable failure early?
  - Can we take advantage of problem structure?

Order of Assignment of Variables

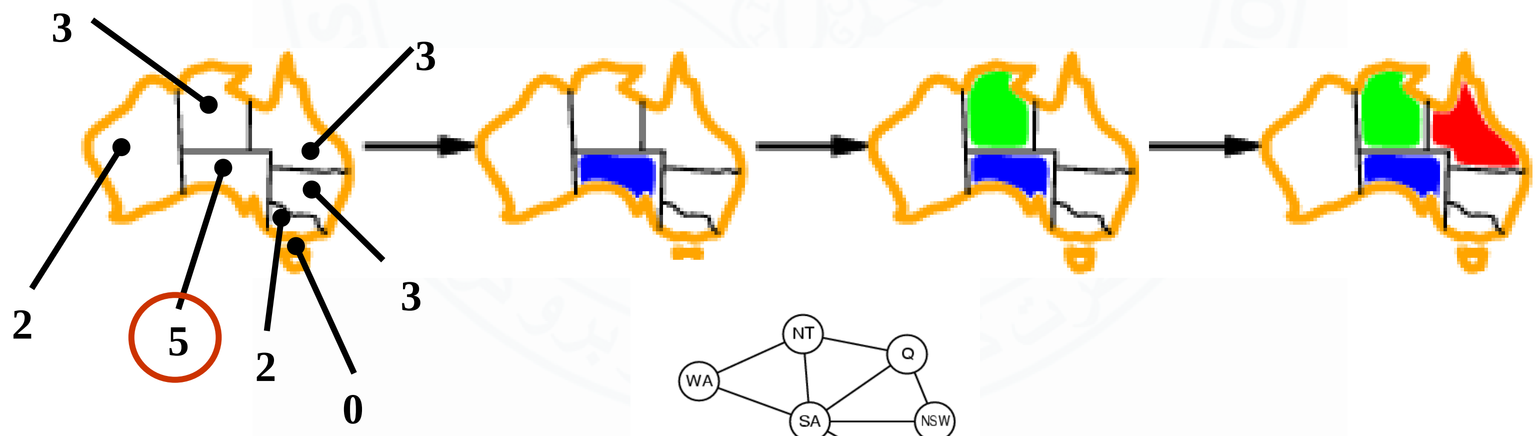
- Minimum Remaining Values (MRV)
  - Rule: Choose the variable for assignment which has the smallest number of possible moves
  - Most constrained value heuristic
  - Allows us to detect failure early and prevent un-necessary expansion of nodes

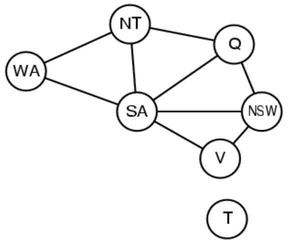




Order of Assignment of Variables…

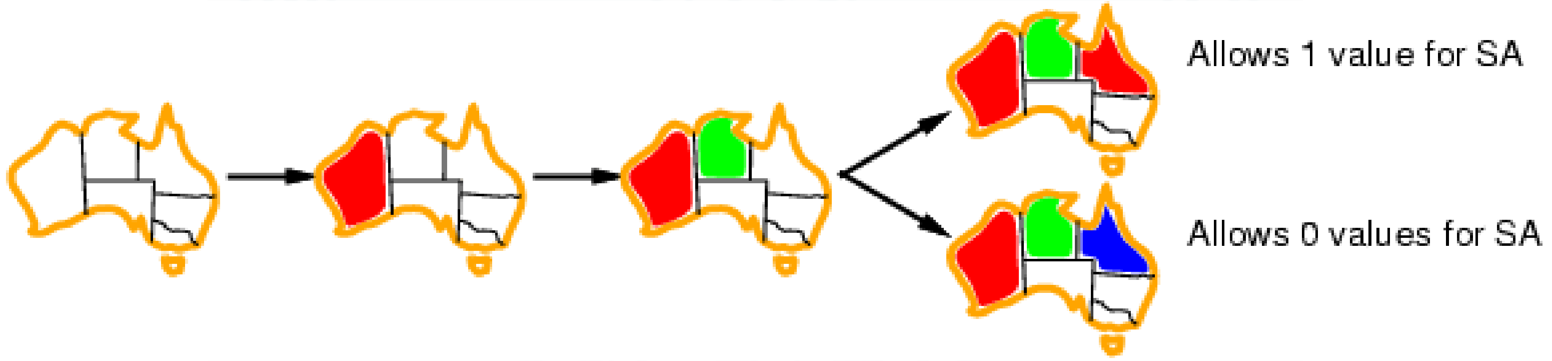
- Most Constraining Variable (MCV)
  - Rule: Choose the variable with the most constraints on remaining variables
  - Acts as a tie breaker in MRV
  - Degree Heuristic





Assignment of Values to Selected Variable

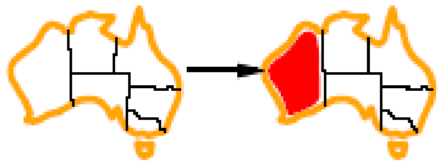
- Least constraining value heuristic
  - Rule: given a variable choose the least constraing value i.e. the one that leaves the maximum flexibility for subsequent variable assignments

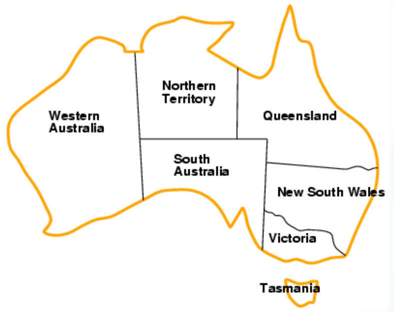


Checking Inevitable Failure Early

- Forward Checking
  - Keep track of remaining feasible values for unassigned variables
  - Terminate search when any variable has no legal values

Forward checking

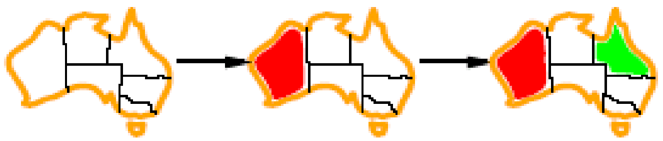


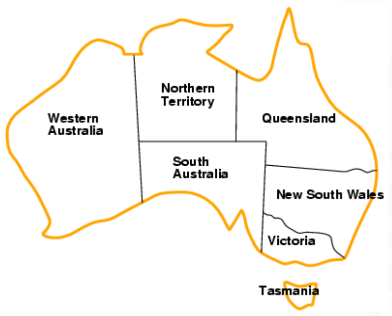


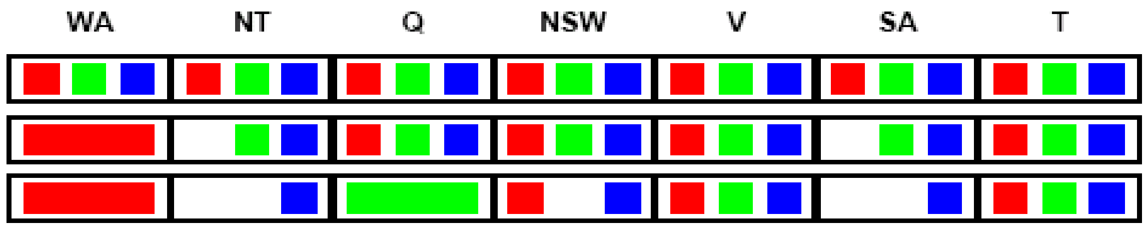
WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦

- Assign *{ WA = red }*
- Effects on other variables connected by constraints with WA
  - *NT can no longer be red*
  - *SA can no longer be red*

Forward checking



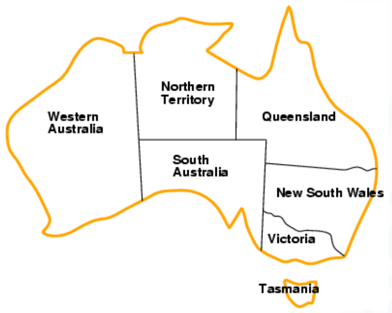


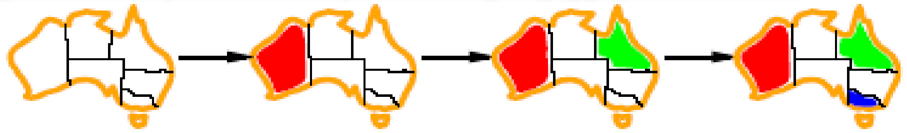


WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦

- Assign *{ Q = green }*
- Effects on other variables connected by constraints with WA
  - *NT can no longer be green*
  - *NSW can no longer be green*
  - *SA can no longer be green*
- *MRV heuristic* will automatically select NT and SA next, why?

Forward checking

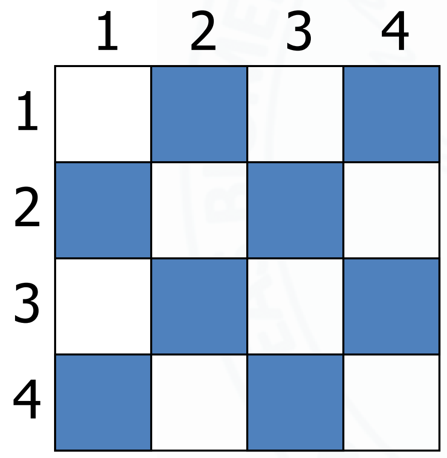


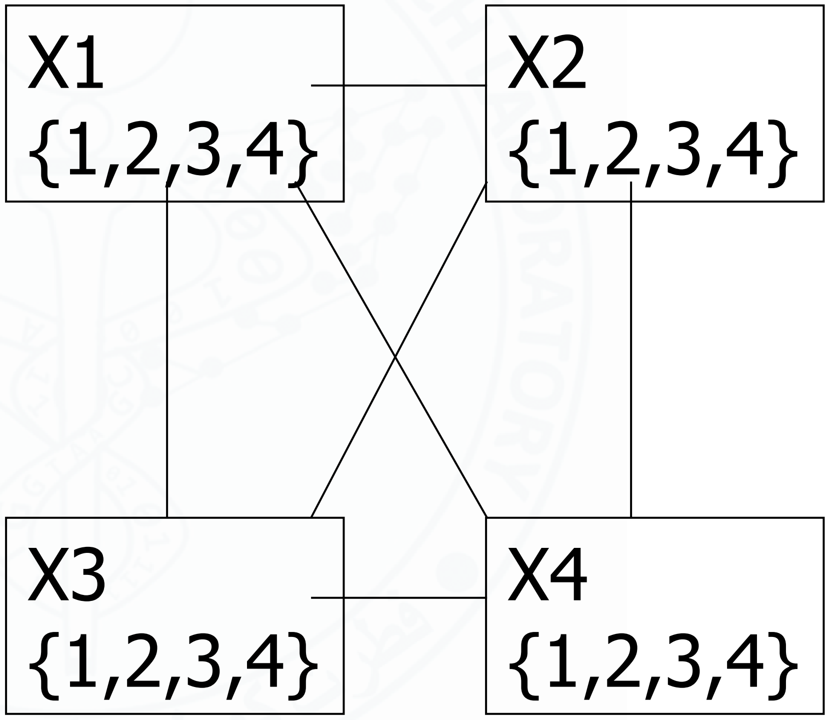


WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦
🟥	🟦	🟩	🟥	🟦		🟥🟩🟦

- If *V* is assigned *blue*
- Effects on other variables connected by constraints with WA
  - *SA is empty*
  - *NSW can no longer be blue*
- FC has detected that partial assignment is *inconsistent* with the constraints and backtracking can occur

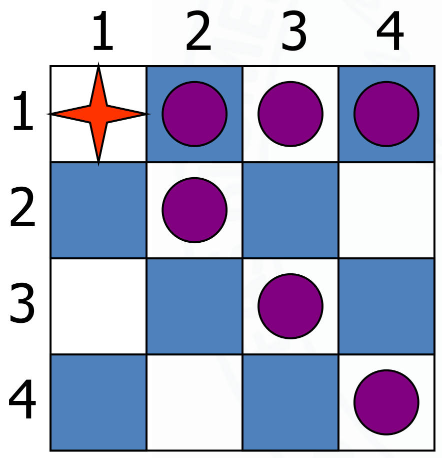
Example: 4-Queens Problem

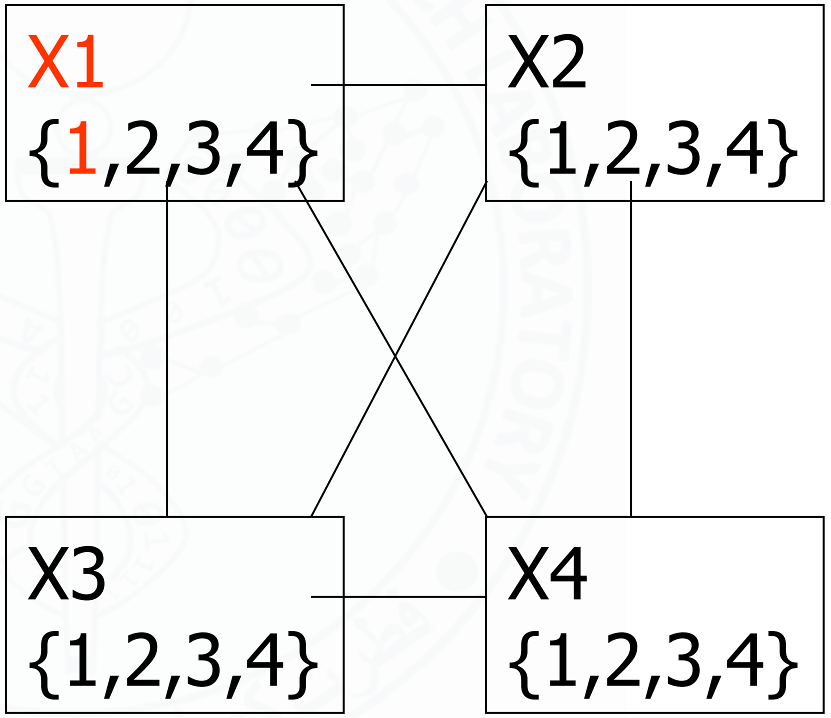




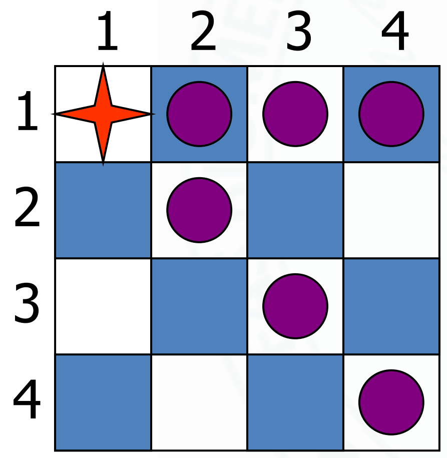
[4-Queens slides copied from B.J. Dorr CMSC 421 course on AI]

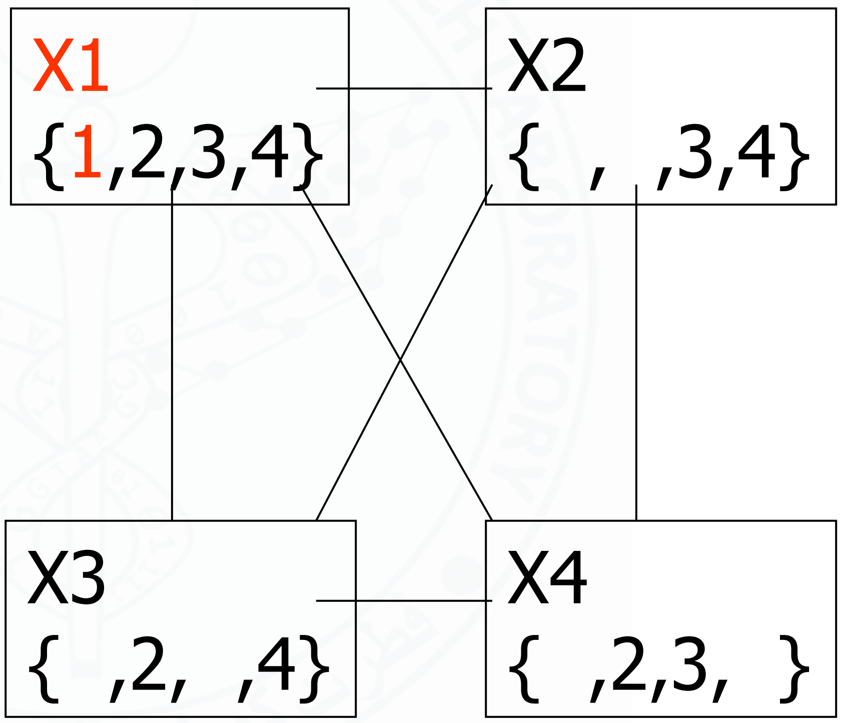
Example: 4-Queens Problem



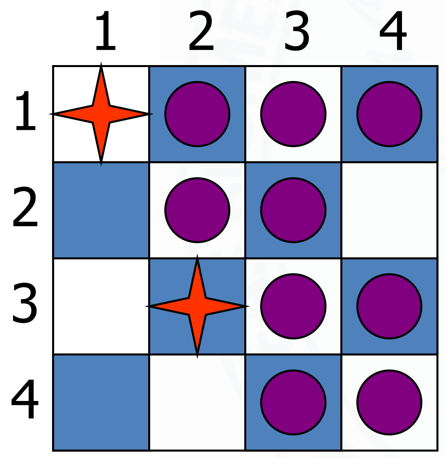


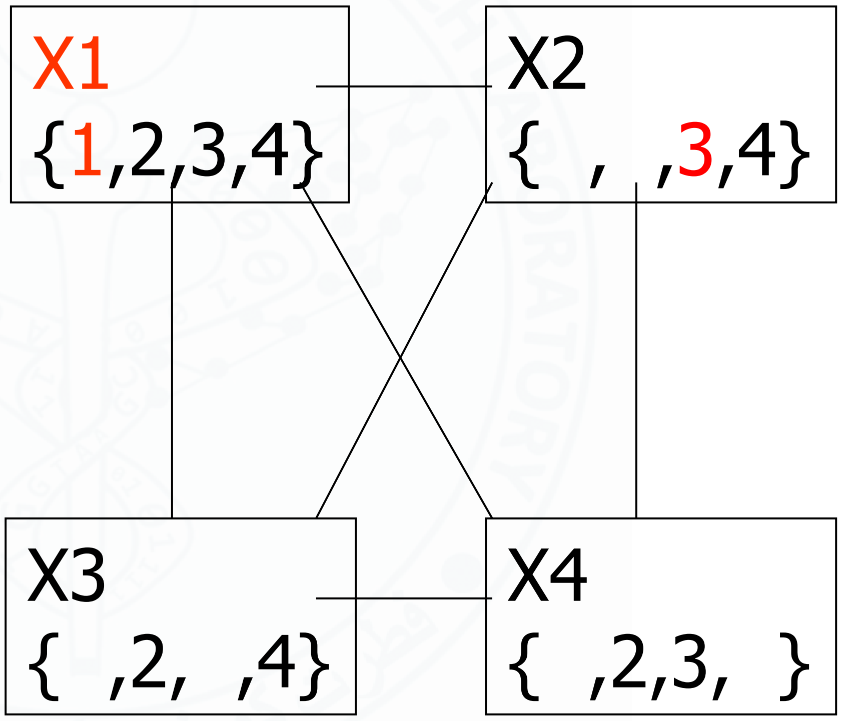
Example: 4-Queens Problem



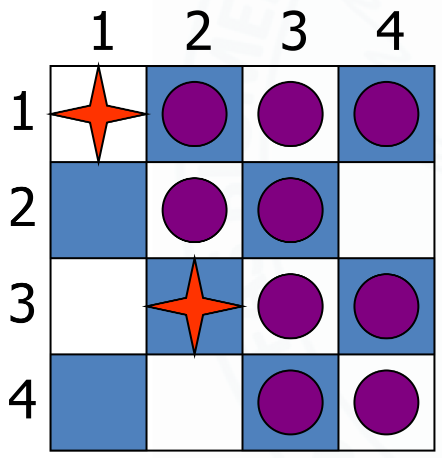


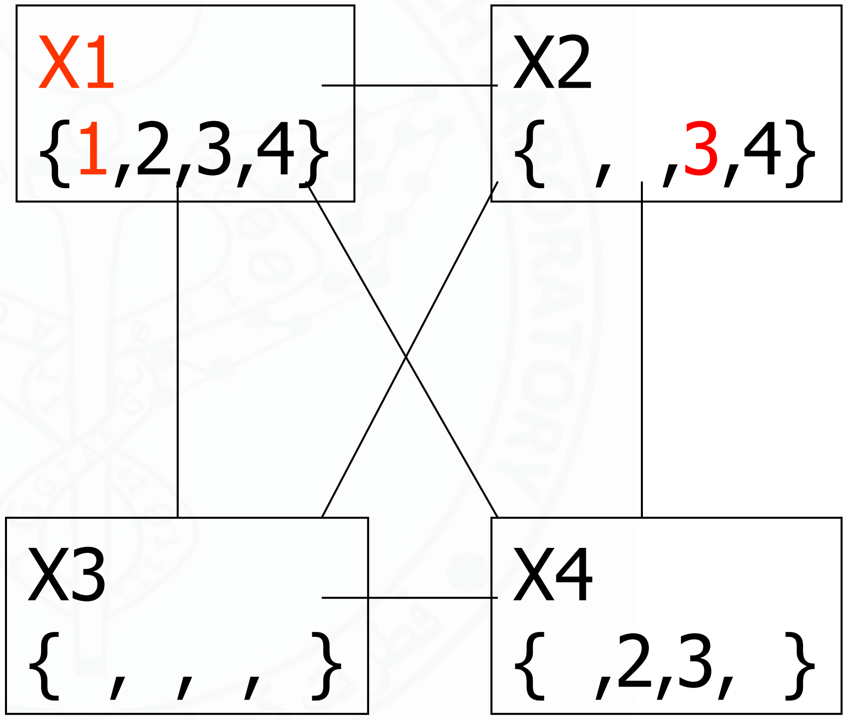
Example: 4-Queens Problem



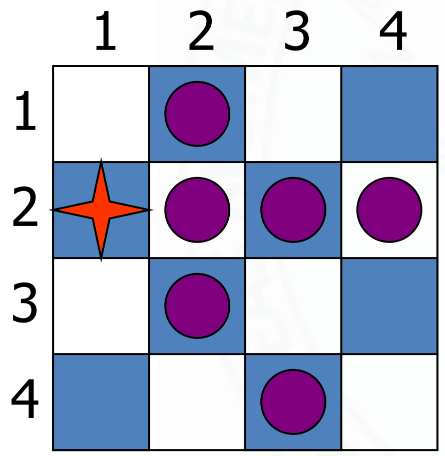


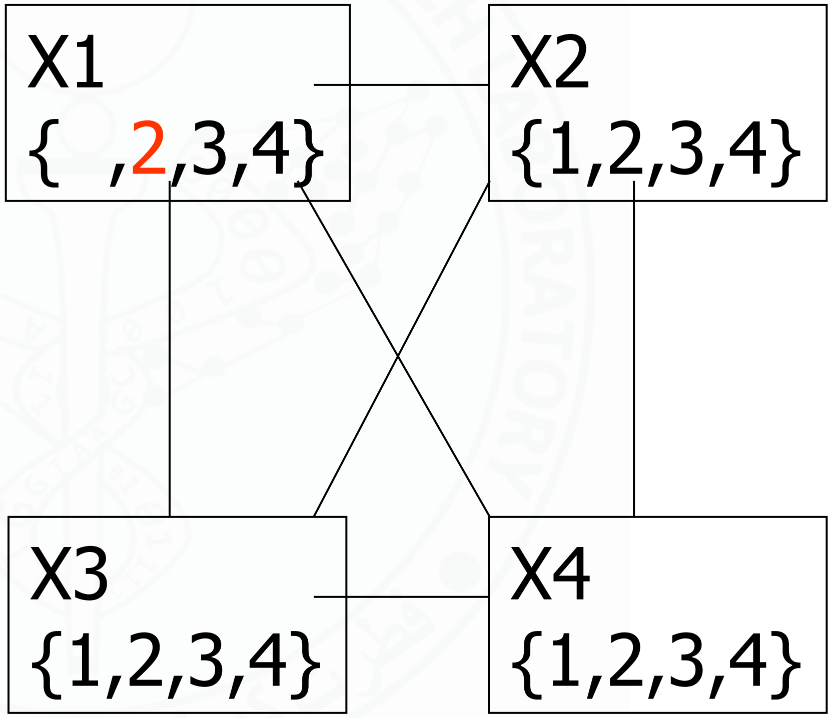
Example: 4-Queens Problem



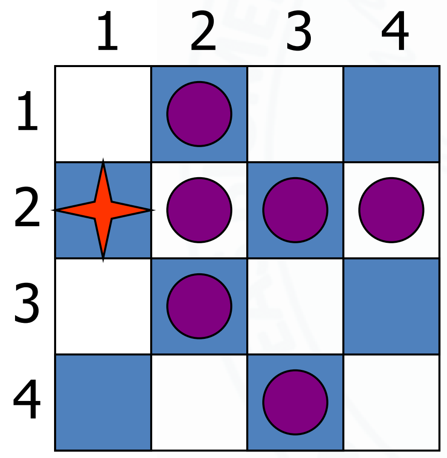


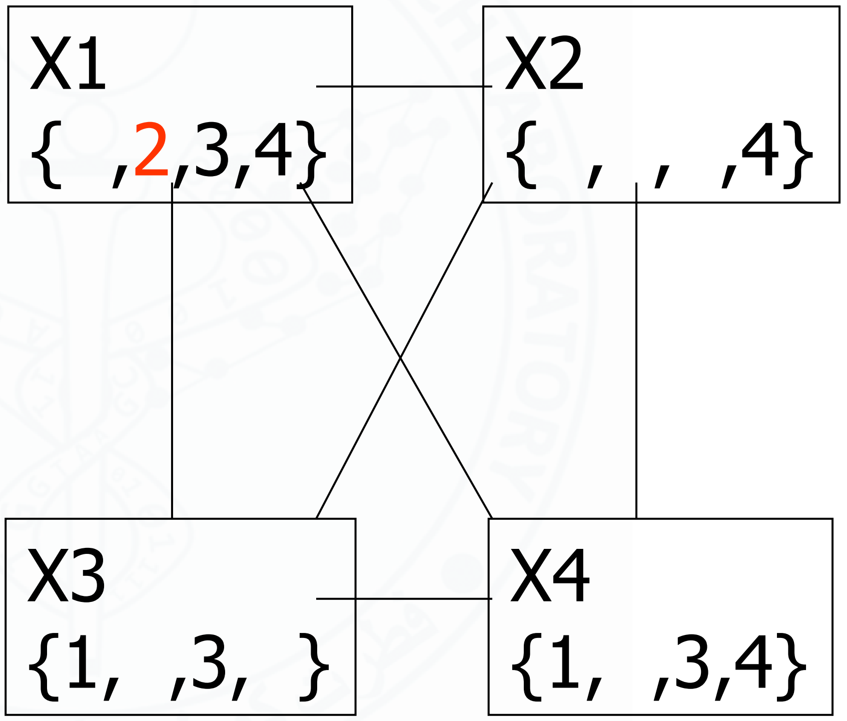
Example: 4-Queens Problem



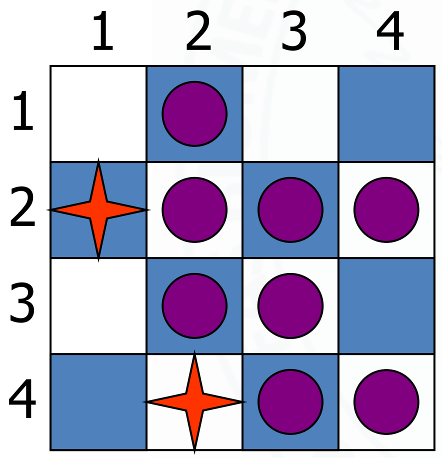


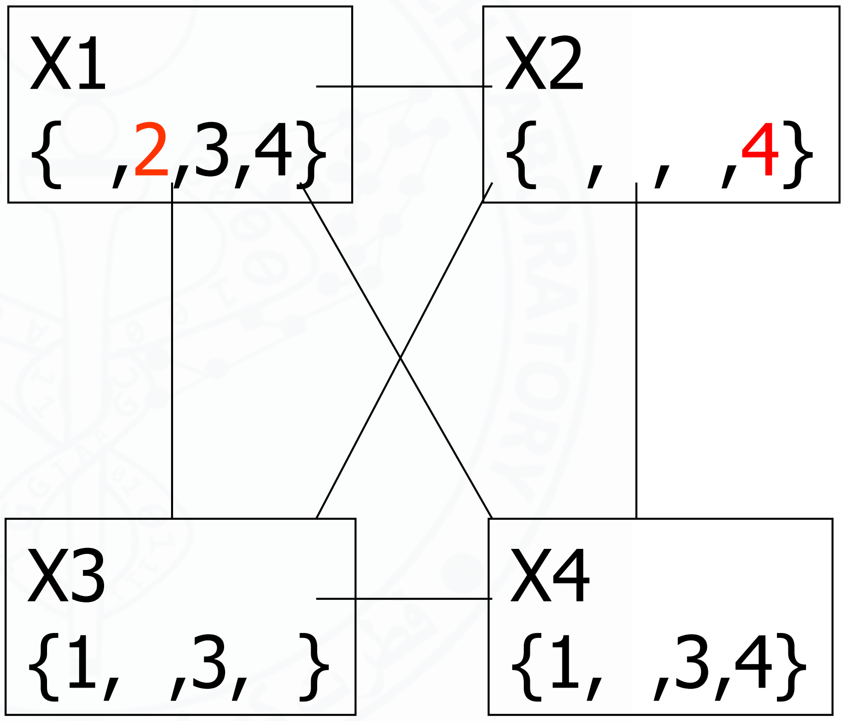
Example: 4-Queens Problem



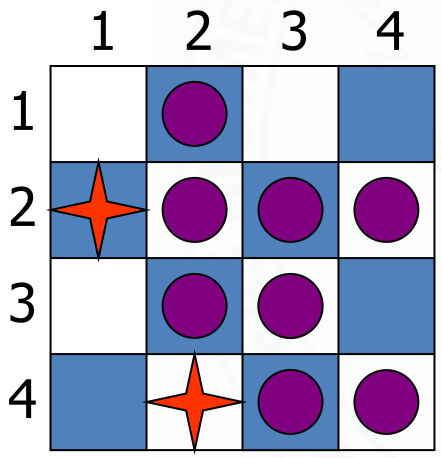


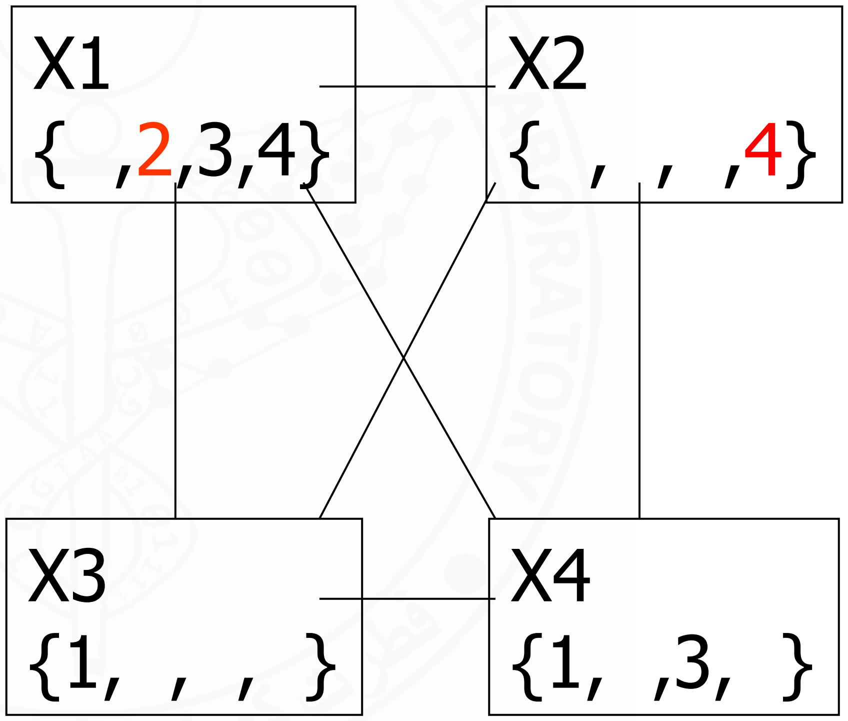
Example: 4-Queens Problem



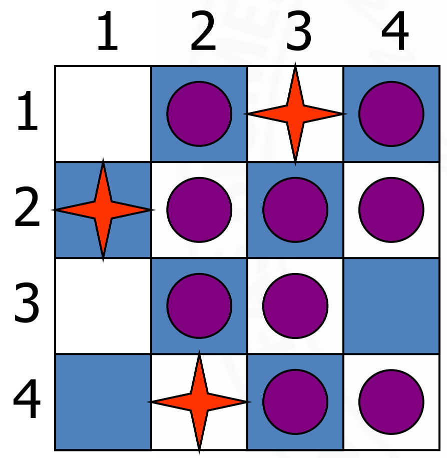


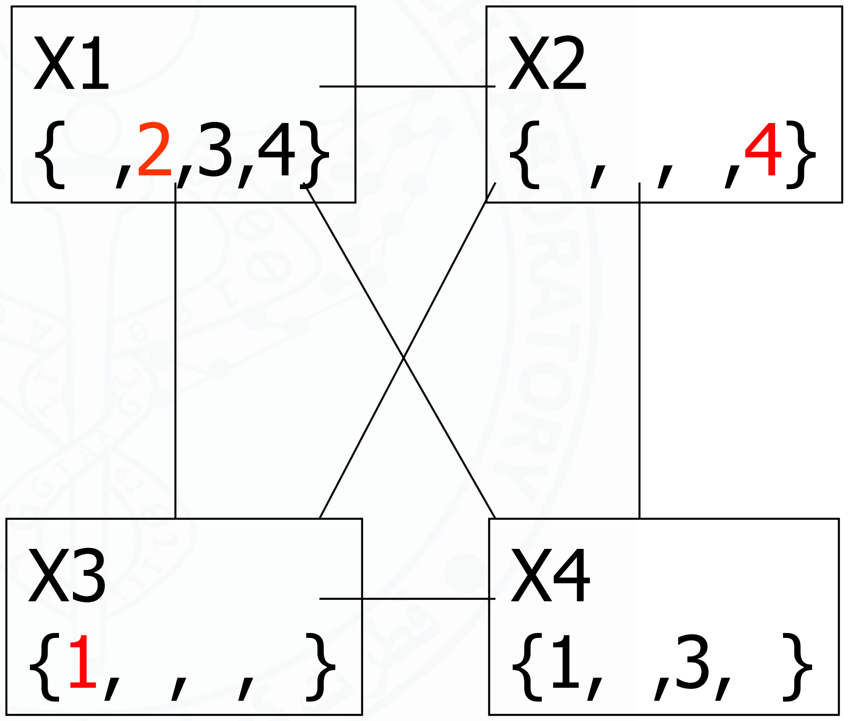
Example: 4-Queens Problem



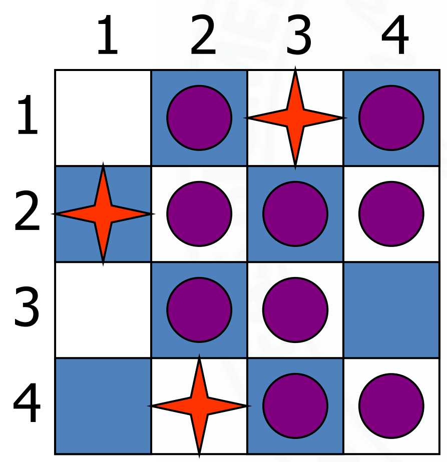


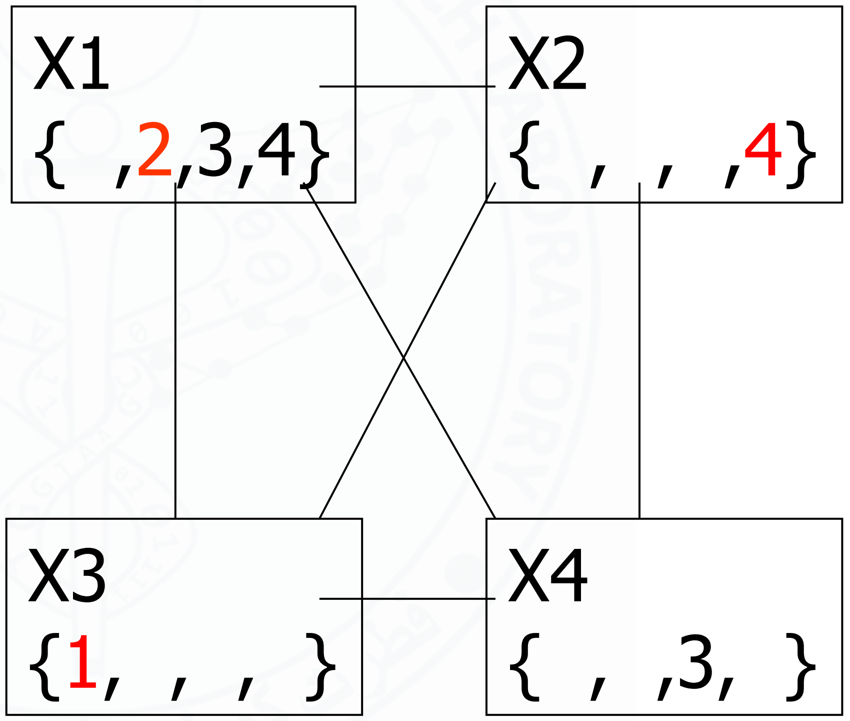
Example: 4-Queens Problem



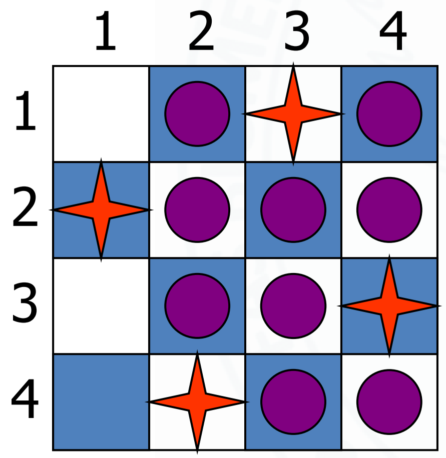


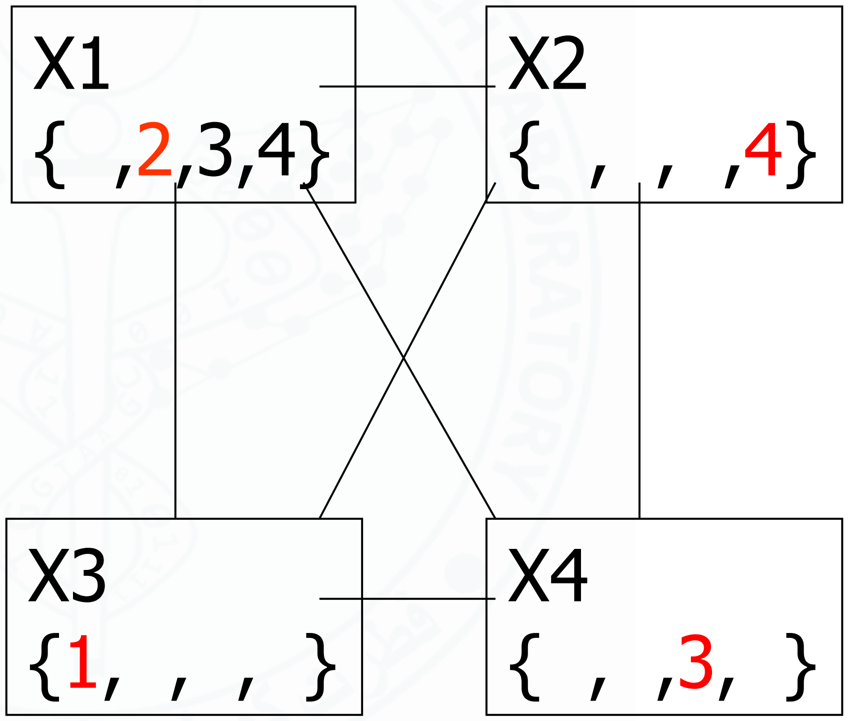
Example: 4-Queens Problem





Example: 4-Queens Problem

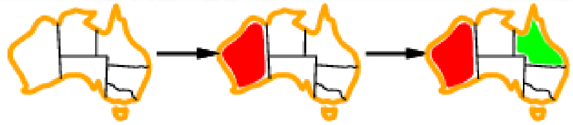




Checking Inevitable Failure Early…

- Solving CSPs with combination of heuristics plus forward checking is more efficient than either approach alone
- FC checking propagates information from assigned to unassigned variables but does not provide detection for all failures
  - NT and SA cannot be blue!

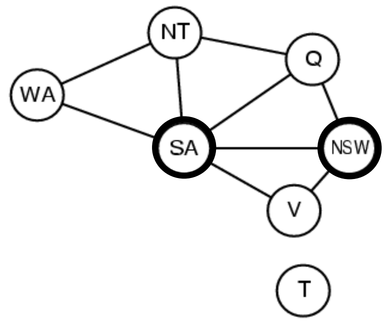
Western Australia
Northern Territory
Queensland
South Australia
New South Wales
Victoria
Tasmania

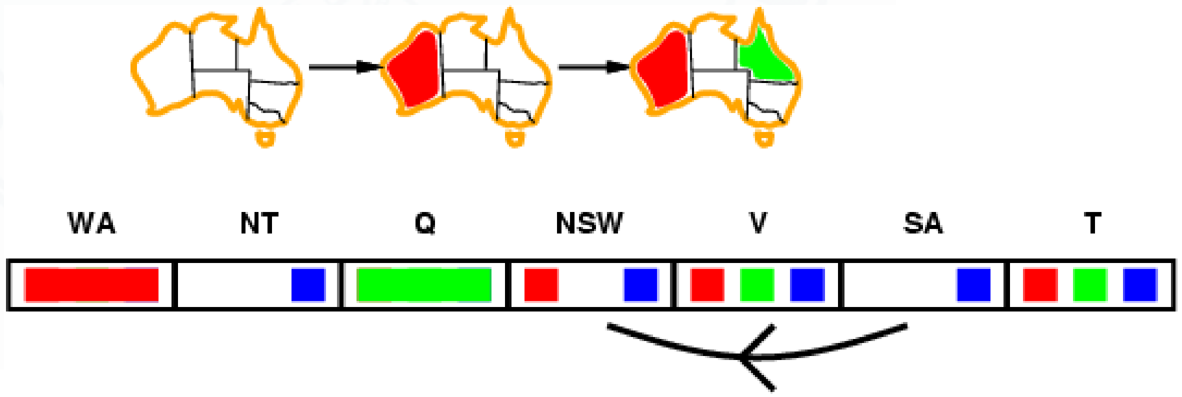


WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦

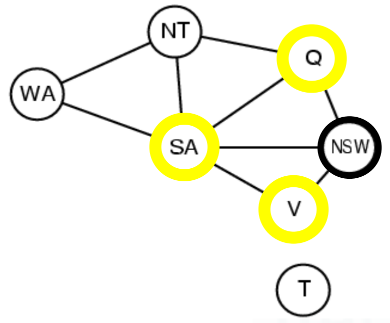
Checking Inevitable Failure Early…

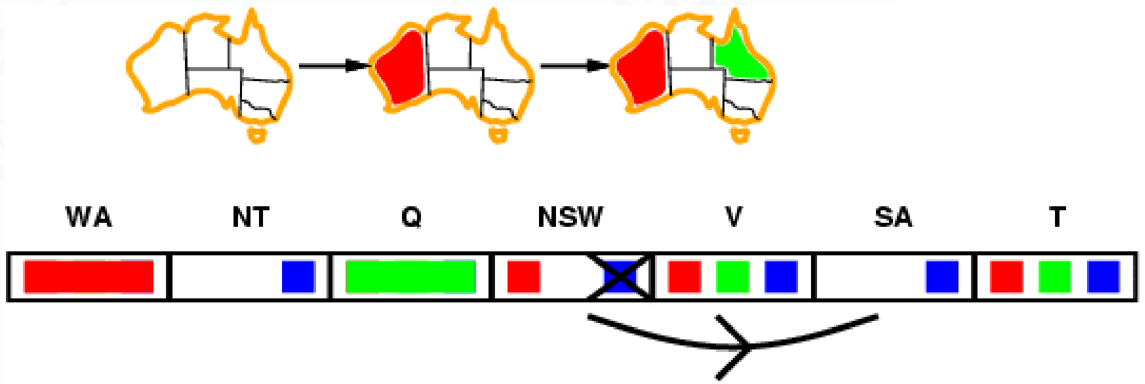
- Constraint propagation
  - repeatedly enforces constraints locally
  - Simplest form of propagation makes each arc consistent
  - *X* → *Y* is consistent iff
    - for every value *x* of *X* there is some allowed *y*



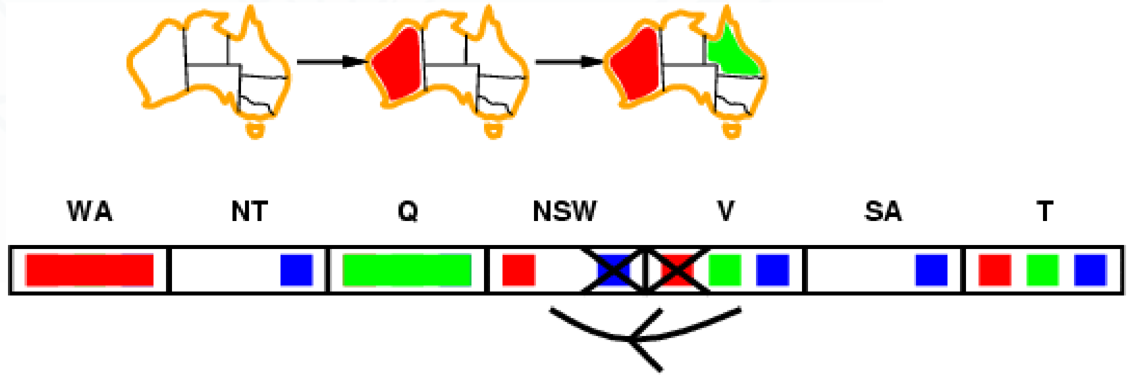


Constraint Propagation…



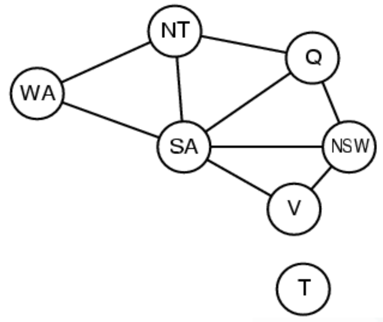


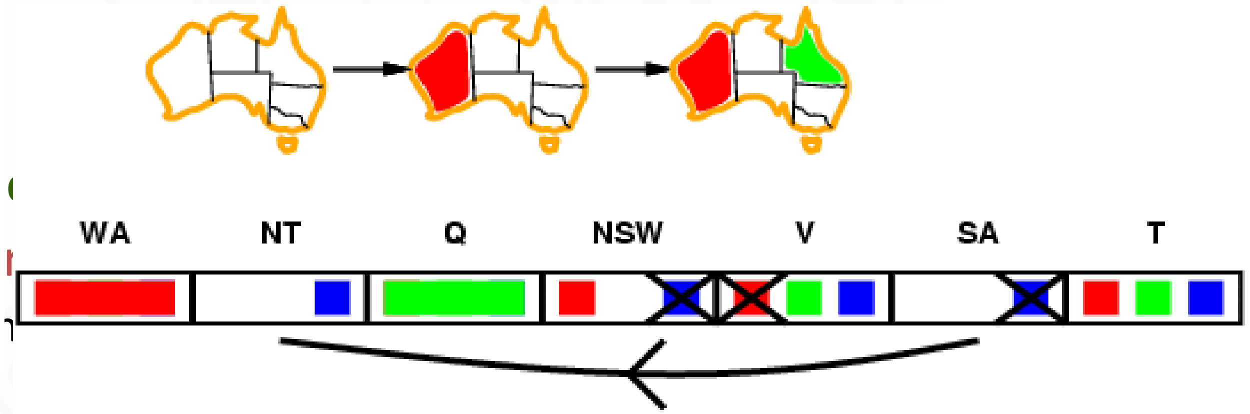
- If *X* loses a value, neighbors of *X* need to be rechecked
  - SA, V and Q are neighbors of NSW



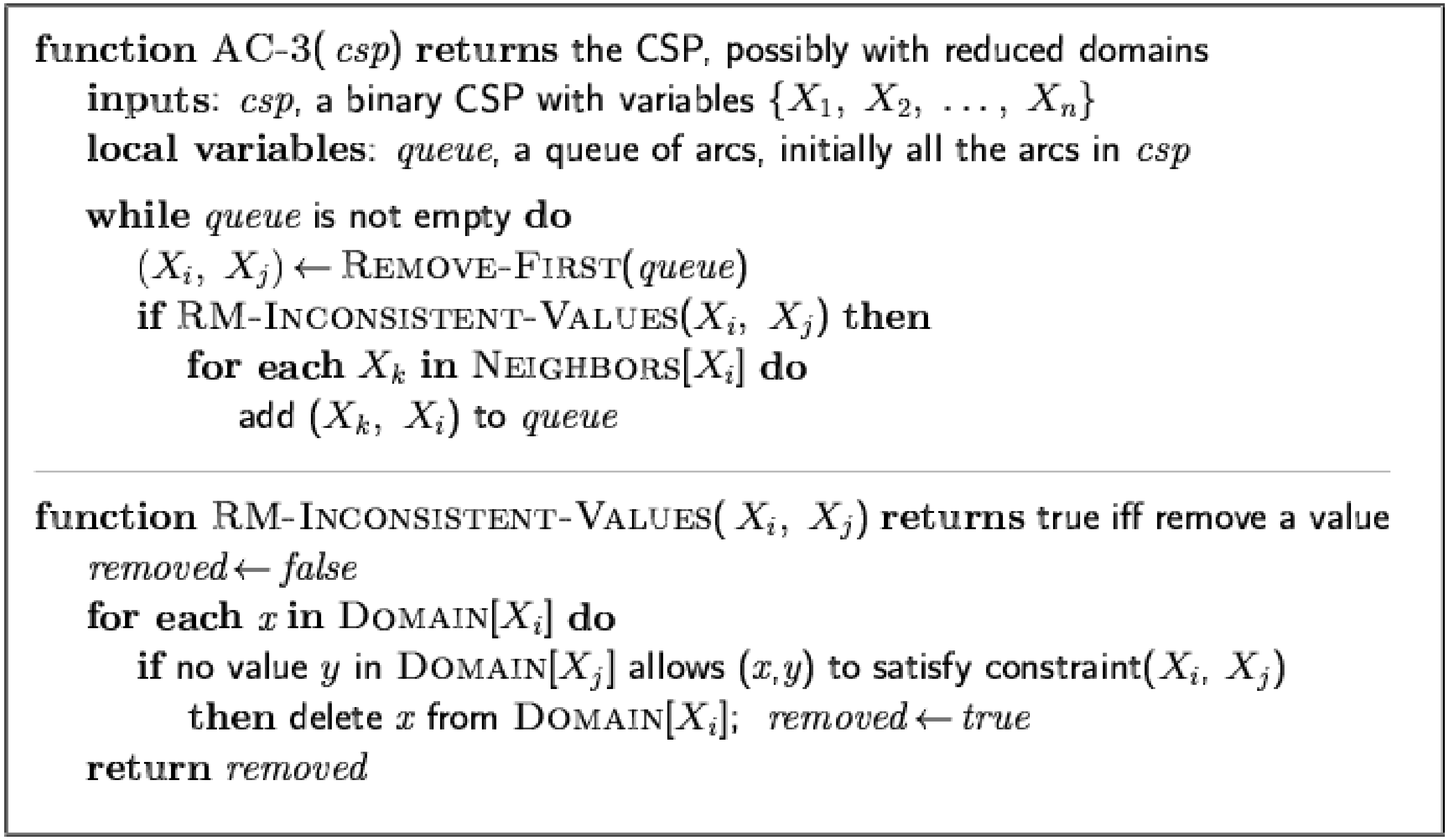
Constraint Propagation…

- CP would ultimately check the arc (SA, NT) for consistency
  - This leads to an empty feasible domain for SA indicating the inevitable failure





Constraint Propagation…



```
function AC-3( $csp$) returns the CSP, possibly with reduced domains
    inputs: $csp$, a binary CSP with variables $\{X_1, X_2, \ldots, X_n\}$
    local variables: $queue$, a queue of arcs, initially all the arcs in $csp$

    while $queue$ is not empty do
        $(X_i, X_j) \leftarrow$ REMOVE-FIRST($queue$)
        if RM-INCONSISTENT-VALUES($X_i, X_j$) then
            for each $X_k$ in NEIGHBORS[$X_i$] do
                add $(X_k, X_i)$ to $queue$
```

```
function RM-INCONSISTENT-VALUES( $X_i, X_j$) returns true iff remove a value
    $removed \leftarrow false$
    for each $x$ in DOMAIN[$X_i$] do
        if no value $y$ in DOMAIN[$X_j$] allows $(x,y)$ to satisfy constraint($X_i, X_j$)
            then delete $x$ from DOMAIN[$X_i$];   $removed \leftarrow true$
    return $removed$
```

Complexity: { $O(n^2d^3)$ }

Constraint Propagation…

- Arc consistency does not detect all inconsistencies:
  - Partial assignment *{ WA = red, NSW = red }* is inconsistent
- Stronger forms of propagation can be defined using the notion of k-consistency
- A CSP is k-consistent if for any set of k-1 variables and for any consistent assignment to those variables, a consistent value can always be assigned to any k-th variable
  - E.g. 1-consistency or node-consistency
  - E.g. 2-consistency or arc-consistency
  - E.g. 3-consistency or path-consistency

Constraint Propagation…

- A graph is strongly k-consistent if
  - It is k-consistent and
  - Is also (k-1) consistent, (k-2) consistent, … all the way down to 1-consistent
- If a CSP is n-consistent then we can solve this problem without backtracking
  - Choose a constant value of X_1
  - A consistent value of X_2 can be found (through searching across the d different values of X_2) because the graph is 2-consistent
  - We can find the values of $X_3 \dots X_n$ in a similar manner
  - Time: *O(nd)*
- YET *no free lunch*: any algorithm for establishing n-consistency must take time exponential in n, in the worst case

Handling Special Constraints

- Specialized Algorithms can be developed for handling Special Algorithms
  - All-Different
    - Rule: If there are m variables in the constraint and if the have n possible distinct values altogether and m > n, then the constraint cannot be satisfied

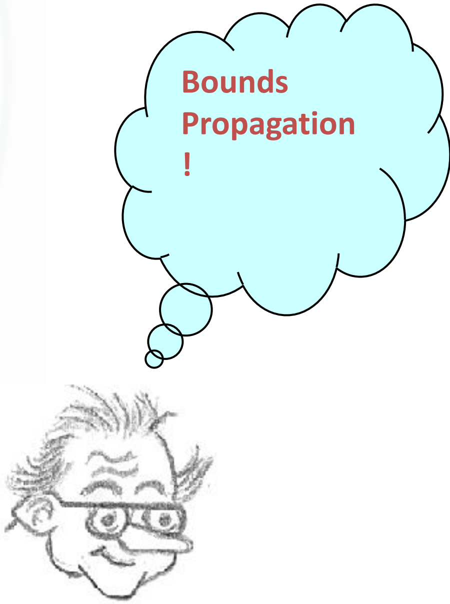
Remove any variables in the constraint that has a singleton domain
Delete that variable's value from the domain of the remaining variables
Repeat as long as there are singleton variables
If at any point an empty domain is produced or there are more variables than domain values then the constraint cannot be satisfied

Handling Special Constraints…

- At-Most Constraint
  - Example: Maximum number of persons assigned to 4 tasks should be less than 10 with 6 being the maximum number of persons available for a single task
    - Domain of Each of the 4 variables is {1…6}
  - Delete the large values of a variable from its domain which are not consistent with the minimum values of other variables
    - Delete { 5, 6} from the domain of each variable

Handling Special Constraints…

- Consider two flights PK165 and PK385 which have capacities of 165 and 385 passengers respectively
- We would like to transport a total of 420 passengers
- What will be the arrangement?
  - Domains
    - PK165 = {0 … 165}, PK385 = {0 … 385}
    - PK165+PK385 = {420,420}
  - Solution
    - PK165 = {35 … 165}, PK385 = {255 … 385}

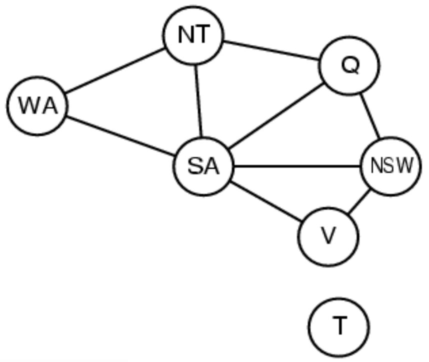


Back Jumping!

- Intelligent backtracking
  - Standard form is chronological backtracking i.e. try different value for preceding variable
  - More intelligent, backtrack to conflict set
    - Set of variables that caused the failure or set of previously assigned variables that are connected to X by constraints
    - Back jumping moves back to most recent element of the conflict set
    - Forward checking can be used to determine conflict set

Conflict Directed Backjumping

- Consider what happens when we apply simple backtracking with a fixed variable ordering Q,NSW,V,T,SA,WA,NT
- Suppose we have the partial Assignment {Q=Red, NSW=Green, V=Blue, T=red}
  - When we try the next variable SA, there are no possible values
  - So backtracking would back up and try the next color for Tasmania!
- What caused the failure?
  - It would be one of Q, NSW or V
  - This set of variables is caused a conflict set
  - So we should jump back to the most recent element of the conflict set



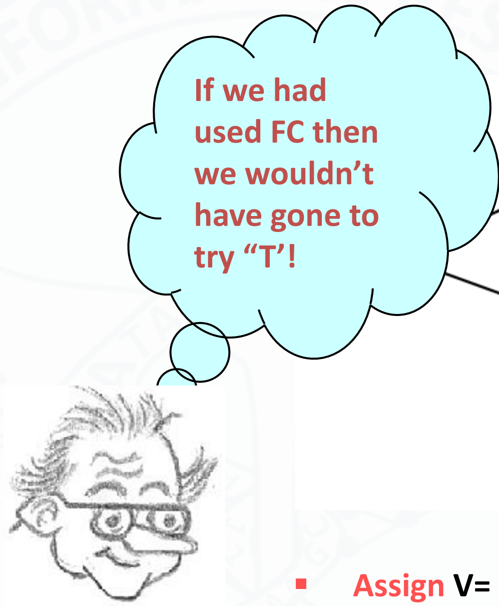
On computation of conflict sets

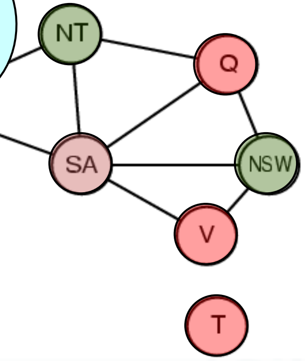
- The conflict set of a variable X is the set of previously assigned variables that are connected to X by constraints
- Forward checking can be used to compute the conflict set
  - Whenever FC based on an assignment to X deletes a value from Y’s domain
    - X should be added to the conflict set of Y
  - Every time the last value is deleted from Y’s domain, the variables in the conflict set of Y are added to the conflict set of X (minus X itself)

CDBJ Example (with FC)

- Variable ordering (fixed)
  - Q,NSW,V,T,SA,WA,NT
- C(X) indicates the conflict set of X

- Assign Q = Red
  - C(SA)={Q}, D(SA)={G,B}
  - C(NT)={Q}, D(NT)={G,B}
  - C(NSW)={Q}, D(NSW)={G,B}
- Assign NSW= Green
  - C(V)={NSW}, D(V)={R,B}
  - C(SA)={Q.NSW}, D(SA)={B}
- Assign V= Blue
  - C(SA)={Q,NSW,V}. D(SA)={} (Last value deleted)
  - C(V)={NSW} u {Q,NSW,V} – {V} = {Q,NSW}
- Assign T= Red
- Assign SA
  - No possible assignment
  - C(SA)={Q,NSW,V}
  - Backjump to V



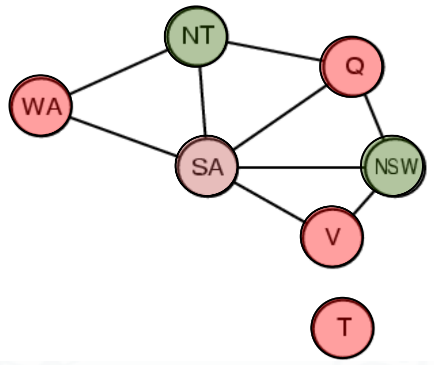


- **Assign V= Red**
  - **C(SA)={Q,NSW,V}, D(SA)= {B}**
- **Assign T = Red (Already assigned!)**
- **Assign SA = Blue**
  - **C(NT)={Q,SA}, D(NT)={G}**
  - **C(WA)={SA}. D(WA)={R,G}**
- **Assign WA=Red**
  - **C(NT)={Q,SA,WA}, D(NT)={G}**
- **Assign NT = Green**

CDBJ Example (with FC)

- Variable ordering (fixed)
  - WA,NSW,T,NT,Q,V,SA
- C(X) indicates the conflict set of X

- Assign WA = Red
  - C(SA)={WA}, D(SA)={G,B}
  - C(NT)={WA}, D(NT)={G,B}
- Assign NSW= Red
  - C(V)={NSW}, D(V)={G,B}
  - C(Q)={NSW}, D(Q)={G,B}
- Assign T = Red
- Assign NT= Green
  - C(SA)={WA,NT}. D(SA)={B}
  - C(Q)={NSW,NT}, D(Q)={B}
- Assign Q= Blue
  - C(SA)={WA,NT,Q}. D(SA)={} (Last value Deleted)
  - C(Q)={NSW,NT} u {WA,NT,Q} – {Q} = {WA,NSW,NT}
  - Backjump to NT



- **Assign NT = Blue**
  - **C(SA)={WA,NT}, D(SA)= {}**
  - **C(NT)={WA}**
  - **Back jump to WA**
  - **…**

Local search for CSP

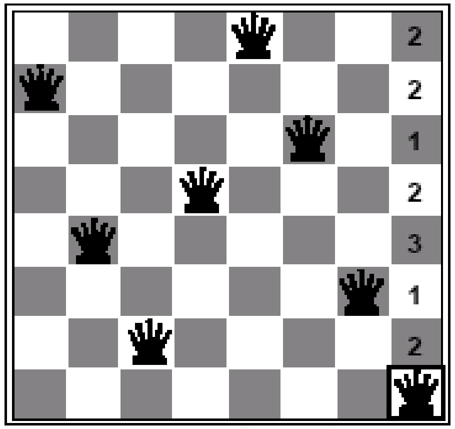
- Use complete-state representation
- For CSPs
  - allow states with unsatisfied constraints
  - operators reassign variable values
- Variable selection: randomly select any conflicted variable
- Value selection: *min-conflicts heuristic*
  - Select new value that results in a minimum number of conflicts with the other variables

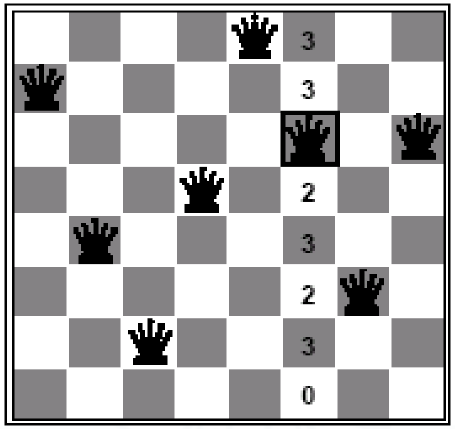
Local search for CSP

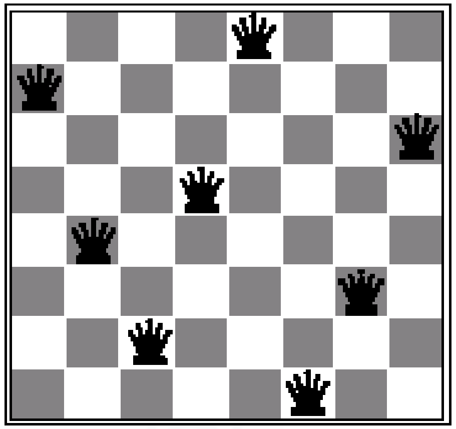
```
function MIN-CONFLICTS(csp, max_steps) return solution or failure
    inputs: csp, a constraint satisfaction problem
            max_steps, the number of steps allowed before giving up

    current ← an initial complete assignment for csp
    for i = 1 to max_steps do
            if current is a solution for csp then return current
            var ← a randomly chosen, conflicted variable from VARIABLES[csp]
            value ← the value v for var that minimize CONFLICTS(var,v,current,csp)
            set var = value in current
    return failure
```

Min-conflicts example





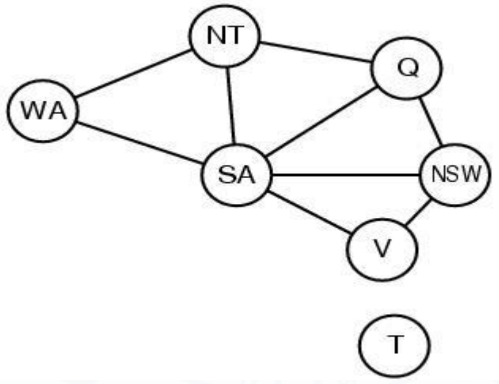


- A two-step solution for an 8-queens problem using min-conflicts heuristic.
- At each stage a queen is chosen for reassignment in its column.
- The algorithm moves the queen to the min-conflict square breaking ties randomly.

Advantages of local search

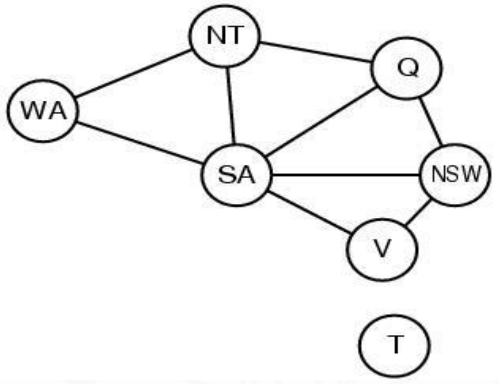
- The runtime of min-conflicts is roughly independent of problem size
  - Solving the millions-queen problem in roughly 50 steps!!
- Local search can be used in an online setting
  - Backtrack search requires more time

Problem structure



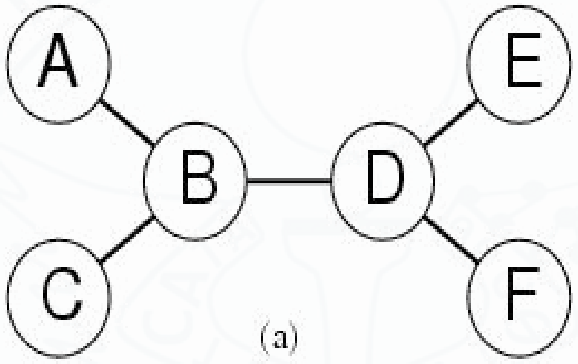
- *How can the problem structure help to find a solution quickly?*
- Subproblem identification is important:
  - Coloring Tasmania and mainland are independent subproblems
  - Identifiable as connected components of constrained graph.
- Improves performance

Problem structure



- Suppose each problem has *c* variables out of a total of *n*.
- Worst case solution cost is $O(n/c\ d^c)$, i.e. linear in *n*
  - Instead of $O(d^n)$, exponential in *n*
- E.g. *n= 80, c= 20, d=2*
  - 2^{80} = 4 billion years at 1 million nodes/sec.
  - 4 * 2^{20}= .4 second at 1 million nodes/sec

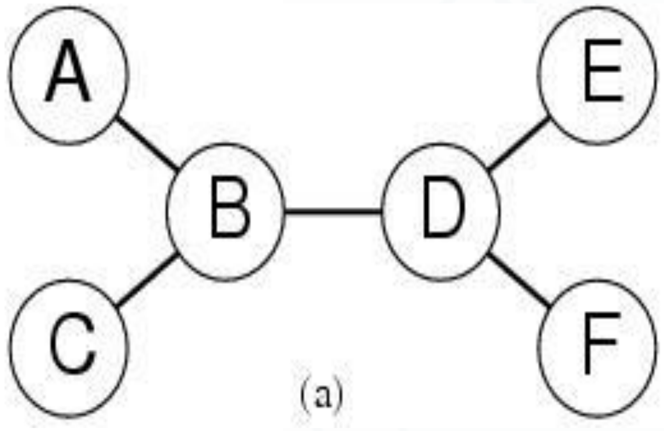
Tree-structured CSPs

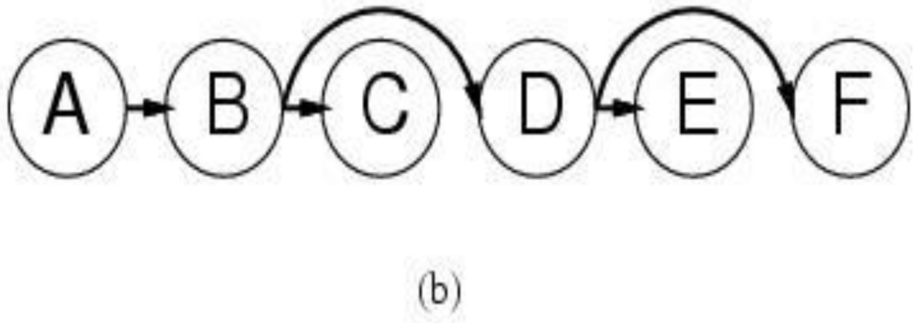


(a)

- Theorem: if the constraint graph has no loops then CSP can be solved in *O(nd* ^{2}*)* time
- Compare difference with general CSP, where worst case is *O(d* ^{n}*)*

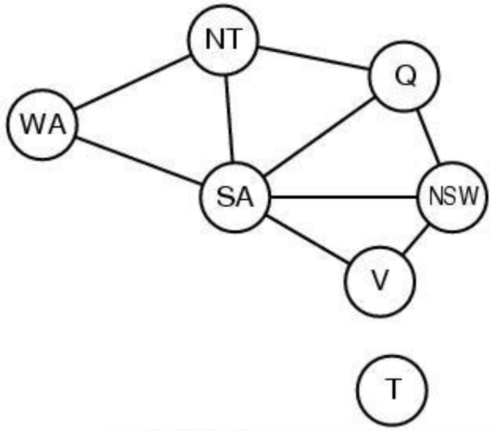
Tree-structured CSPs

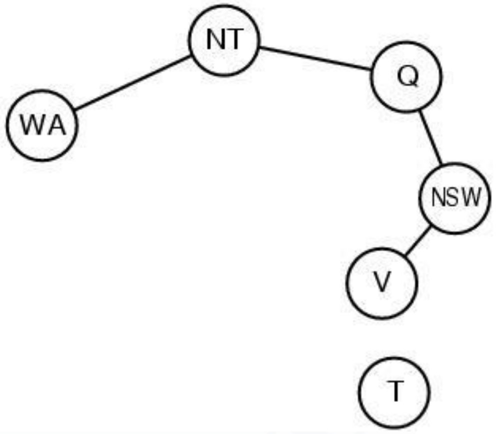




- In most cases subproblems of a CSP are connected as a tree
- Any tree-structured CSP can be solved in time linear in the number of variables.
  - Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering. (label var from X_1 to X_n)
  - For j from n down to 2, apply REMOVE-INCONSISTENT-VALUES(Parent(X_j),X_j)
  - For j from 1 to n assign X_j consistently with Parent(X_j)

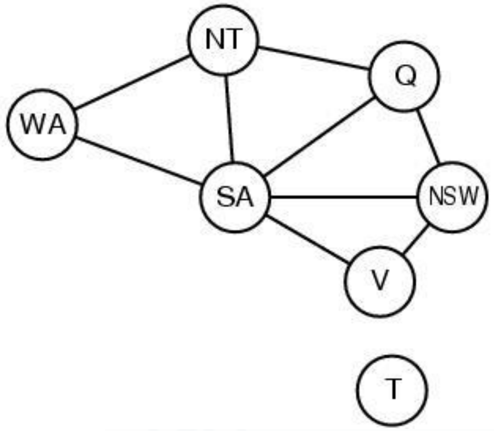
Nearly tree-structured CSPs

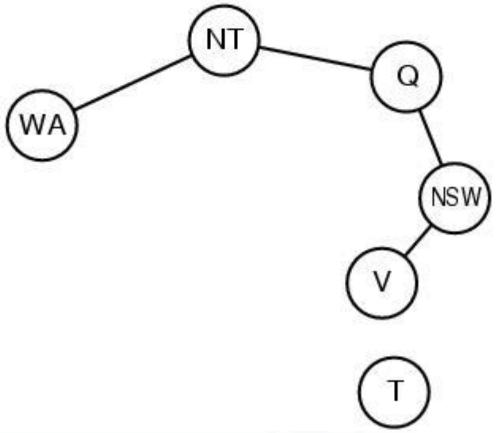




- *Can more general constraint graphs be reduced to trees?*
- Two approaches:
  - Remove certain nodes
  - Collapse certain nodes

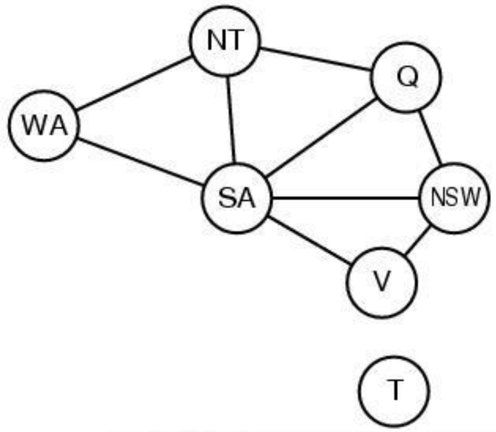
Nearly tree-structured CSPs

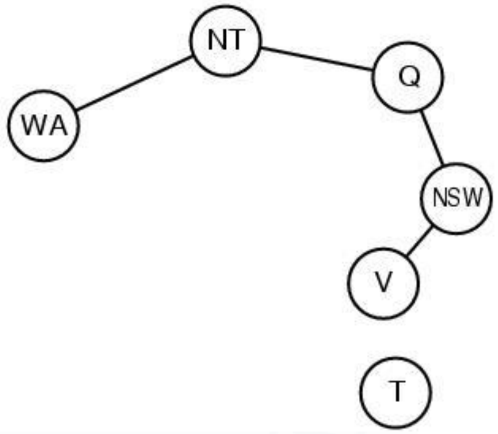




- Idea: assign values to some variables so that the remaining variables form a tree.
- Assume that we assign *{SA=x}* ← *cycle cutset*
  - And remove any values from the other variables that are inconsistent.
  - The selected value for SA could be the wrong one so we have to try all of them

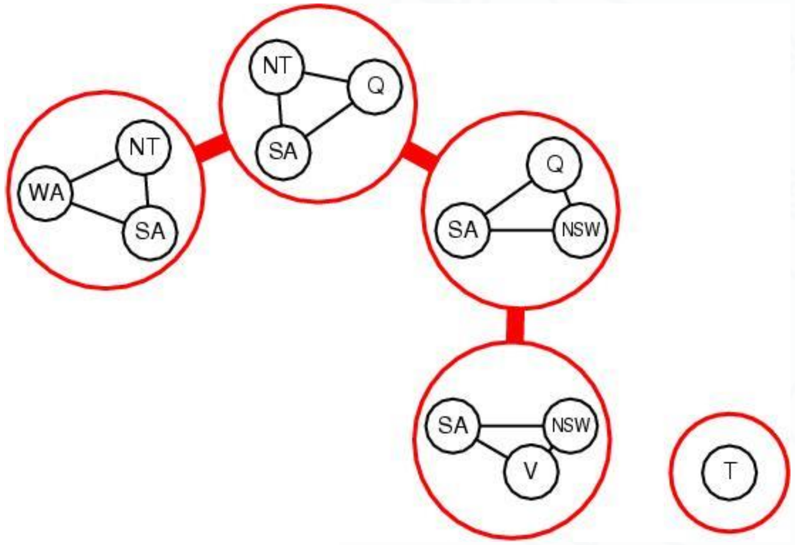
Nearly tree-structured CSPs





- This approach is worthwhile if cycle cutset is small.
- Finding the smallest cycle cutset is NP-hard
  - Approximation algorithms exist
- This approach is called *cutset conditioning*.

Nearly tree-structured CSPs



- Tree decomposition of the constraint graph in a set of connected subproblems.
- Each subproblem is solved independently
- Resulting solutions are combined.
- Necessary requirements:
  - Every variable appears in ar least one of the subproblems.
  - If two variables are connected in the original problem, they must appear together in at least one subproblem.
  - If a variable appears in two subproblems, it must appear in each node on the path.

End of Lecture

It is the true nature of mankind to learn from mistakes, not from example.

Fred Hoyle (lived 1915), British astronomer, mathematician, and writer.
Into Deepest Space (1975).