Artificial Intelligence (AI)

CCS-3880 – 3rd Semester 2023

CO5: Constraint Satisfaction Problems (CSP)

Dr. Abdullah Alshanqiti

Constraint Satisfaction Problems (CSP)

CO5: Investigate constraint satisfaction for a given set of problems.

- **Defining Constraint Satisfaction Problems**
  - Example: *Map coloring*
  - Real-world CSPs
  - Constraint Graph
  - Varieties of CSPs
  - Varieties of Constraints
  - Example: *Cryptarithmetic*
- **CSP as a Standard Search Problem**
- **Backtracking Search**
  - Improving Backtracking Efficiency
  - Minimum Remaining Values (MRV) Heuristic
  - Degree Heuristic
  - Least Constraining Value
- **Forward Checking**
  - Drawback in Forward Checking
- **Local Search for CSPs**
  - Min-conflicts Heuristic

Defining Constraint Satisfaction Problems

- We use a factored representation for each state: *a set of variables, each of which has a value*.
- A problem is solved when each variable has a value that satisfies all the constraints on the variable.
- A problem described this way is called a *constraint satisfaction problem*, or CSP.

What is a CSP?

  - Finite set of variables $V_1, V_2, \ldots, V_n$
  - Non-empty domain of possible values for each variable $D_{V1}, D_{V2}, \ldots D_{Vn}$
  - Finite set of constraints $C_1, C_2, \ldots, C_m$
  - Each constraint C_i limits the values that variables can take, e.g., $V_1 \neq V_2$
- A state is an assignment of values to some or all variables.
- Consistent assignment: assignment does not violate the constraints.

CSP- Components

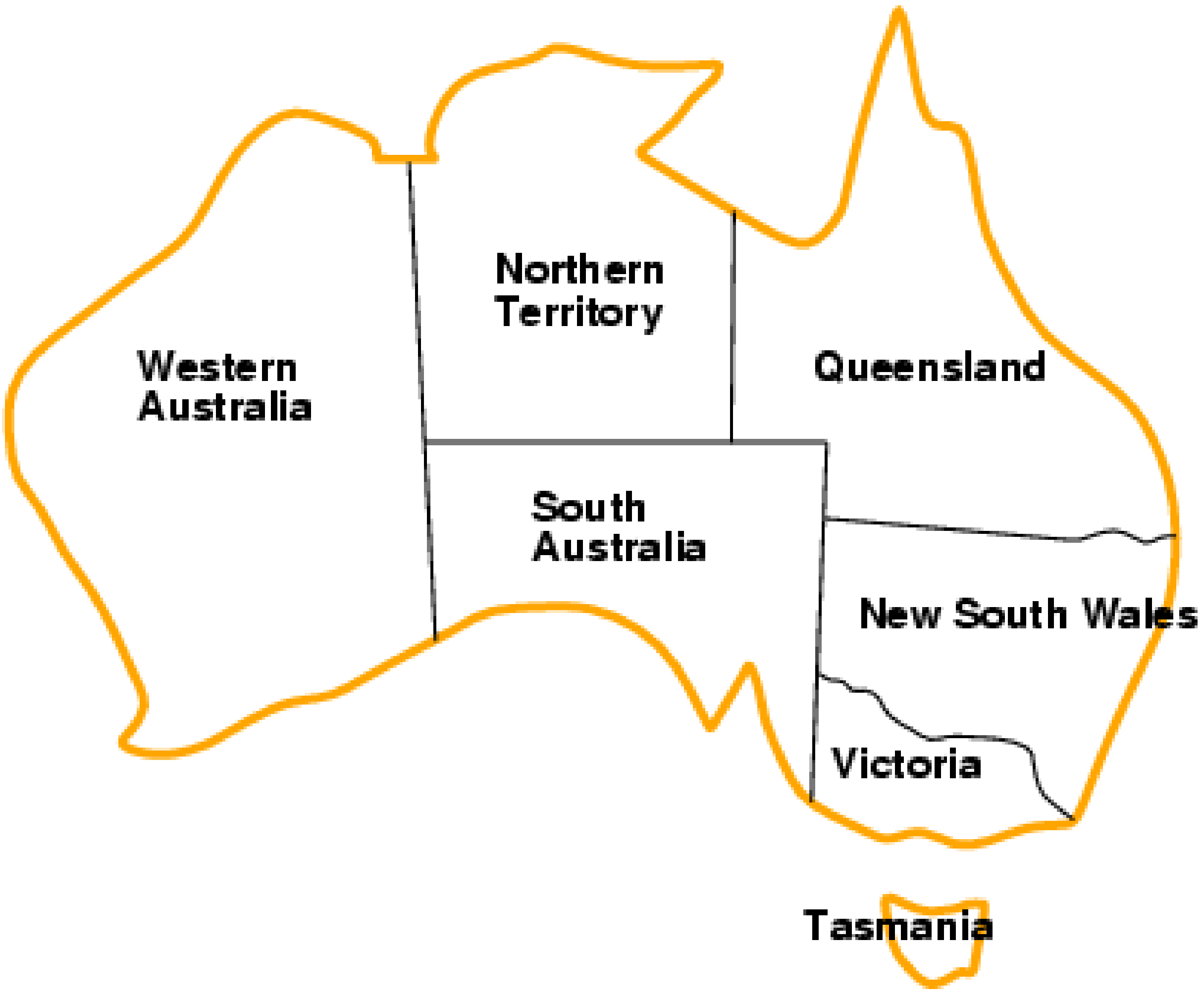
A constraint satisfaction problem consists of three components, X, D, and C :

- X is a set of variables, $\{X_1,...,X_n\}$.
- D is a set of domains, $\{D_1, . . . , D_n\}$, one for each variable.
- C is a set of constraints that specify allowable combinations of values.

- To solve a CSP, we need to define a state space and the notion of a solution.
- Each state in a CSP is defined by an assignment of values to some or all of the variables, $\{X_i = v_i, X_j = v_j , . . .\}$.
- An assignment that does not violate any constraints is called a **consistent** or legal assignment.
- A **complete assignment** is one in which every variable is assigned, and a **solution** to a CSP is a consistent, complete assignment.
- A **partial assignment** is one that assigns values to only some of the variables.

Example problem:

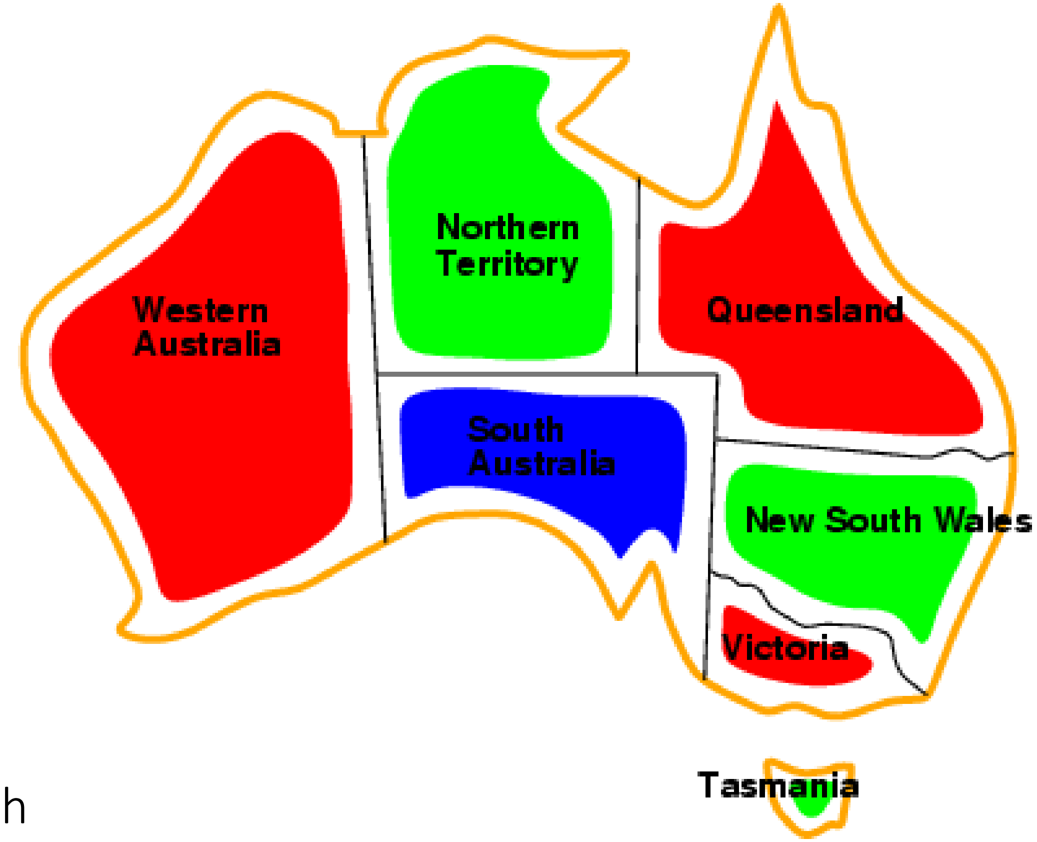
Map coloring



- Variable X = {WA, NT, Q, NSW, V, SA, T} .
- Domain D = {red , green , blue }.
- Constraint C = {neighboring regions to have distinct colors}

Example problem:

Map coloring



- For {SA = blue} in the Australia problem, none of the five neighboring variables can take on the value blue.
- Without taking advantage of constraint propagation, a search procedure would have to consider $3^5 = 243$ assignments for the five neighboring variables; with constraint propagation we never have to consider blue as a value, so we have only $2^5 = 32$ assignments to look at, a reduction of 87%.

Real-world CSPs

- **Assignment problems**
  - e.g., who teaches what class
- **Timetabling problems**
  - e.g., which class is offered when and where?
- **Job scheduling**
  - Whenever a task T_1 must occur before task T_2 , and task T_1 takes duration d_1 to complete, we add an arithmetic constraint of the form $T_1 + d_1 \leq T_2$.
- **Transportation scheduling**
- **Factory scheduling**
- **Notice that many real-world problems involve real-valued variables**

Varieties of Variables

- **Discrete variables**
  - finite domains:
    - n variables, domain size d → $O(d^n)$ complete assignments
  - infinite domains:
    - integers, strings, etc.
    - e.g., job scheduling, variables are start/end days for each job
    - need a constraint language, e.g., $StartJob_1 + 5 \leq StartJob_3$

- **Continuous variables**
  - e.g., start/end times for Hubble Space Telescope observations

Varieties of Constraints

- Unary constraints involve a single variable,
  e.g., SA ≠ “green”
- Binary constraints involve pairs of variables,
  e.g., SA ≠ WA
- Higher-order constraints involve 3 or more variables
  e.g., cryptarithmetic column constraints
- Preference (soft constraints) e.g. red is better than green can be represented by a cost for each variable assignment
  => Constrained optimization problems.

Example: Cryptarithmetic

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

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

CSP as a standard search problem

A CSP can easily be expressed as a standard search problem.

- Initial State: the empty assignment {}.
- Operators: Assign value to unassigned variable provided that there is no conflict.
- Goal test: assignment consistent and complete.
- Path cost: constant cost for every step.

Backtracking Search

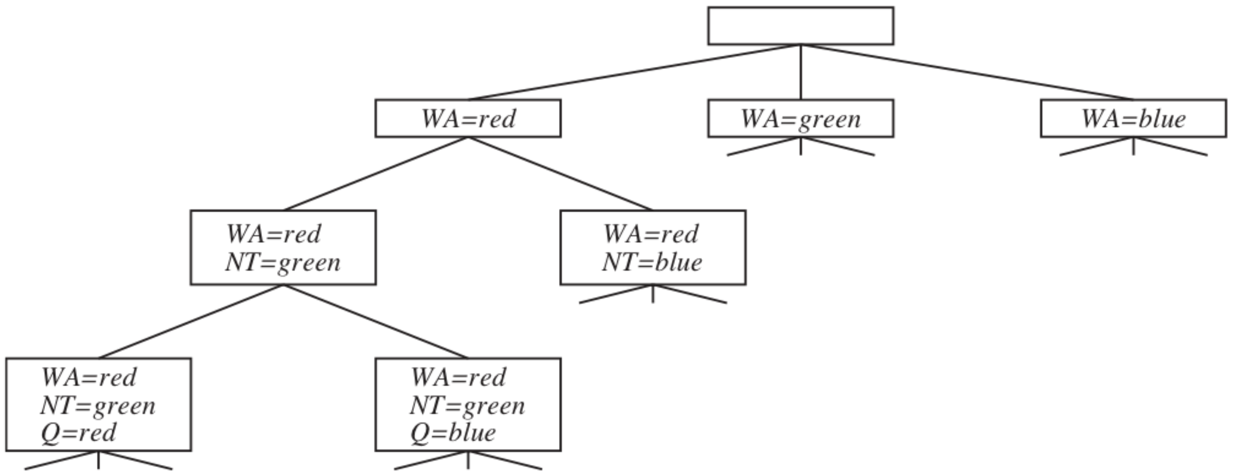
- Variable assignments are *commutative*,
  E.g *[WA = red then NT = green]* equivalent to *[NT = green then WA = red]*

- Only need to consider assignments to a single variable at each node
  → $b = d$ and there are d^n leaves

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

- Backtracking search basic uninformed algorithm for CSPs

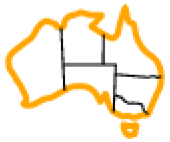
Backtracking Search



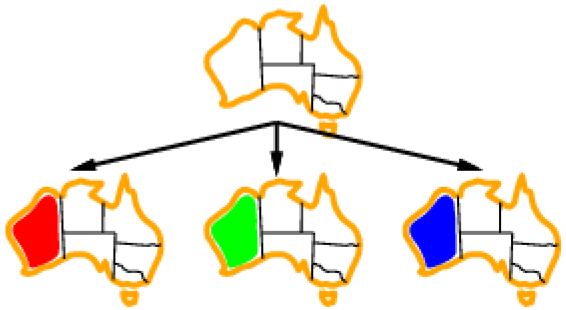

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
     if value is consistent with assignment then
        add {var = value} to assignment
        inferences ← INFERENCE(csp, var, value)
        if inferences ≠ failure then
          add inferences to assignment
          result ← BACKTRACK(assignment, csp)
          if result ≠ failure then
             return result
     remove {var = value} and inferences from assignment
  return failure
```

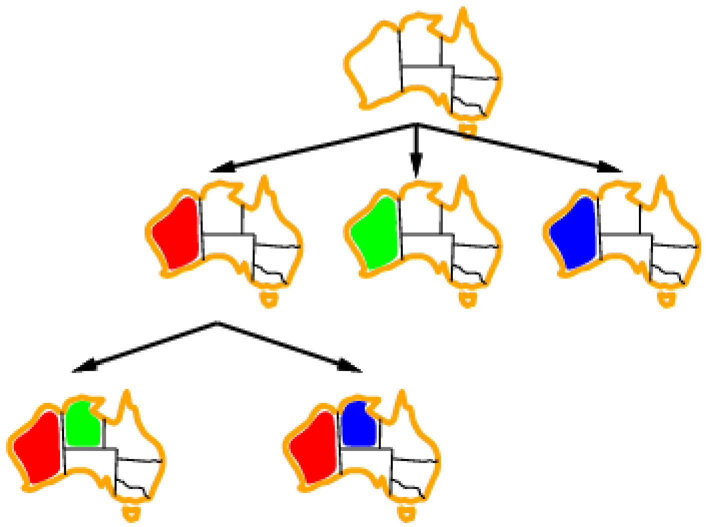
Backtracking Search



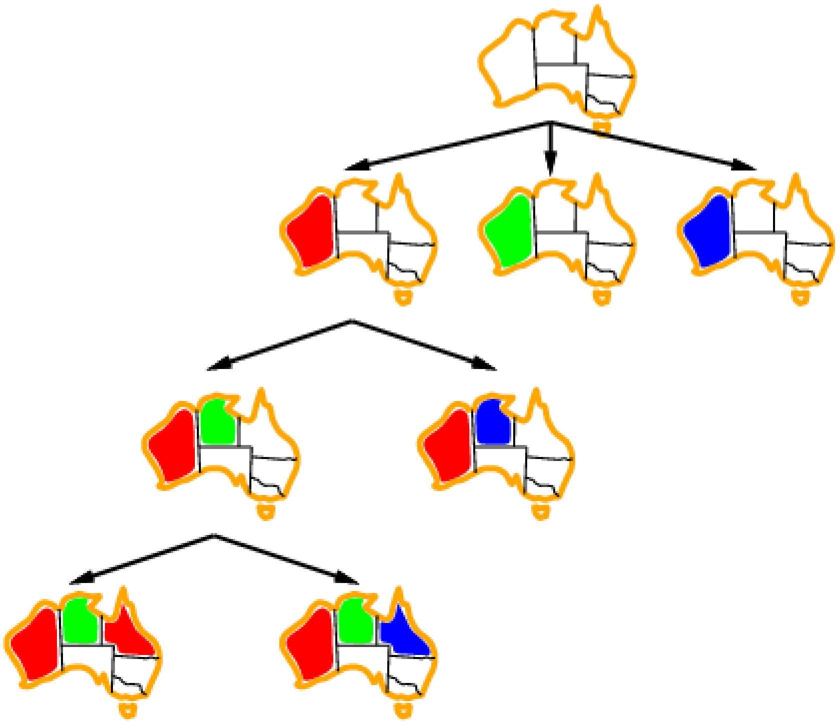
Backtracking Search



Backtracking Search



Backtracking Search



Improving Backtracking Efficiency

General-purpose methods can give huge speed gains:

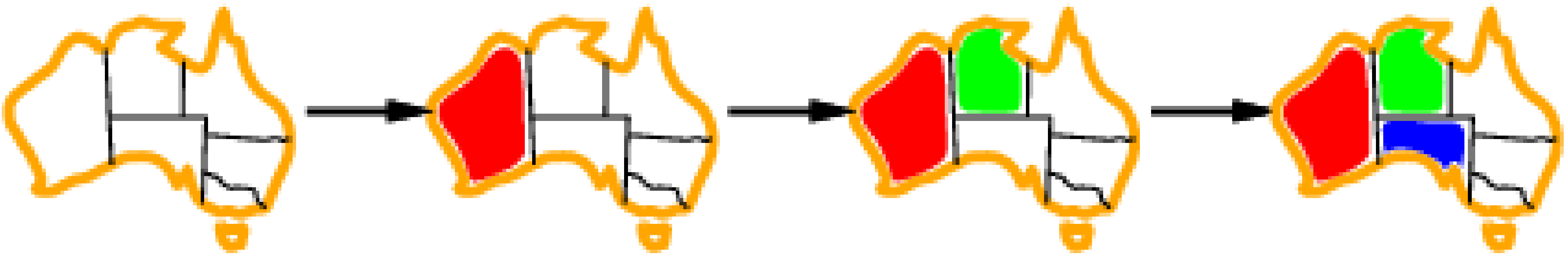
- Which variable should be assigned next? (*Select-unassigned-variable*)
- In what order should its values be tried? (*Order-domain-values*)
- Can we detect inevitable failure early? (*Inference*)
- When the search arrives at an assignment that violates a constraint,

  can the search avoid repeating this failure?

Most Constrained Variable

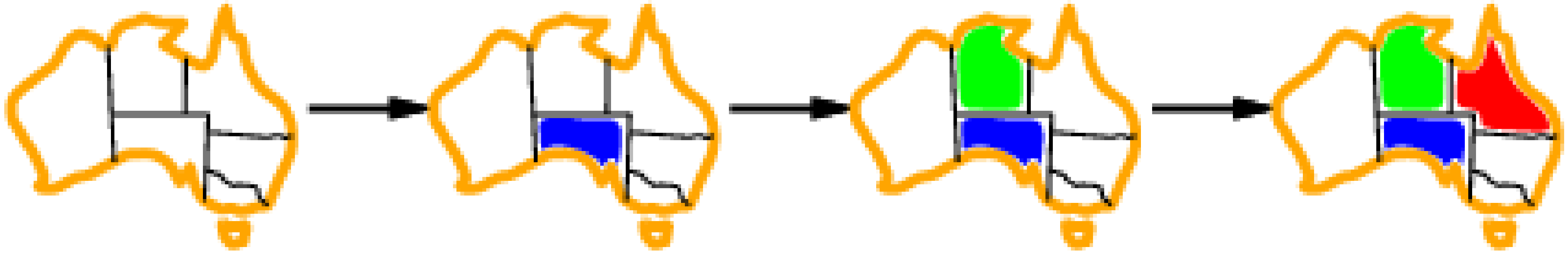
a.k.a. minimum remaining values (MRV) heuristic

Idea: *choose the variable with the fewest legal values*



- The simplest strategy for **SELECT-UNASSIGNED-VARIABLE** is to choose the next unassigned variable in order, {X1, X2, . . .}. This static variable ordering rarely results in the most efficient search.
- For example, after the assignments for WA = red and NT = green, there is only one possible value for SA, so it makes sense to assign SA = blue next rather than assigning Q.

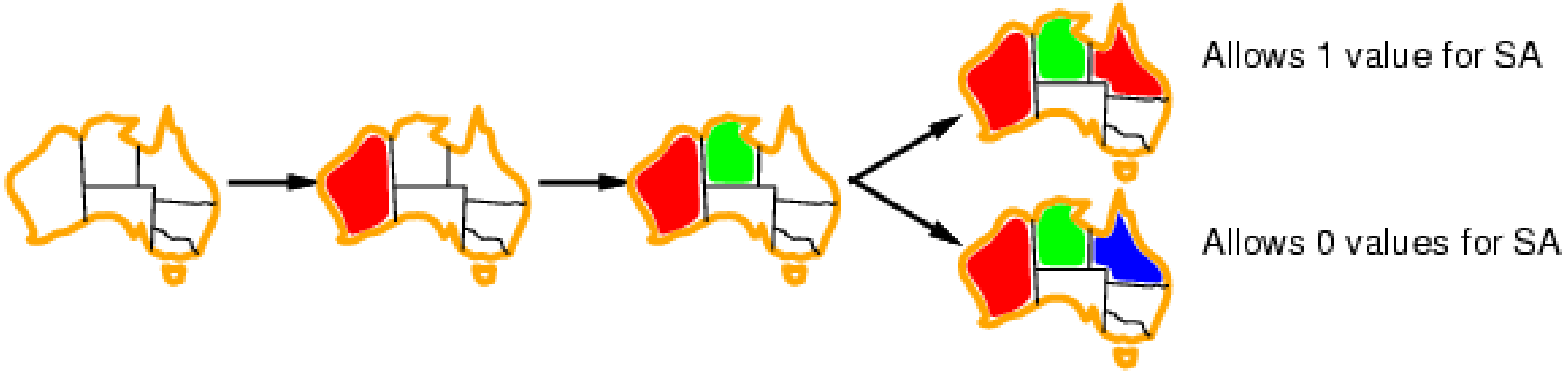
Degree Heuristic



- The MRV heuristic doesn’t help at all in choosing the first region to color in Australia, because initially every region has three legal colors.
- In this case, the **degree heuristic** comes in handy. It attempts to reduce the branching factor on future choices by selecting the variable that is involved in the largest number of constraints on other unassigned variables.
- In the figure, SA is the variable with highest degree, 5; the other variables have degree 2 or 3, except for T , which has degree 0.
- In fact, once SA is chosen, applying the degree heuristic solves the problem without any false steps—you can choose *any* consistent color at each choice point and still arrive at a solution with no backtracking.

Least Constraining Value

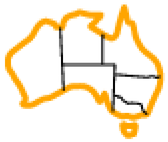
- Given a variable, choose the least constraining value.
- For example, we have generated the partial assignment with WA = red and NT = green and that our next choice is for Q. Blue would be a bad choice because it eliminates the last legal value left for Q's neighbor, SA. The least-constraining-value heuristic therefore prefers red to blue.
- In general, the heuristic is trying to leave the maximum flexibility for subsequent variable assignments.

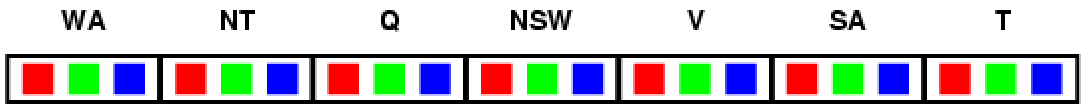


Forward Checking

- One of the simplest forms of inference is called **forward checking**.
- Idea: Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values
- Whenever a variable X is assigned, the forward-checking process establishes arc consistency for it:

    for each unassigned variable Y that is connected to X by a constraint, delete from Y 's domain any value that is inconsistent with the value chosen for X.

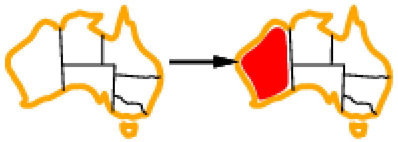


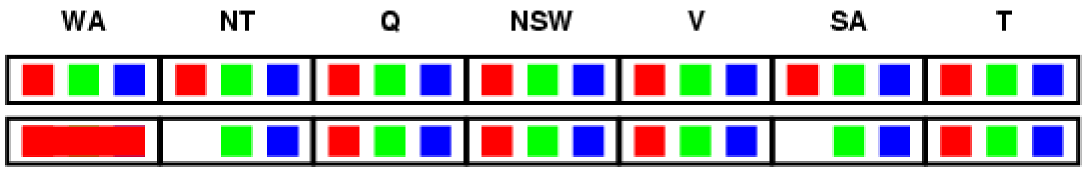


Forward Checking

- One of the simplest forms of inference is called **forward checking**.
- Idea: Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values
- Whenever a variable X is assigned, the forward-checking process establishes arc consistency for it:

  for each unassigned variable Y that is connected to X by a constraint, delete from Y 's domain any value that is inconsistent with the value chosen for X.

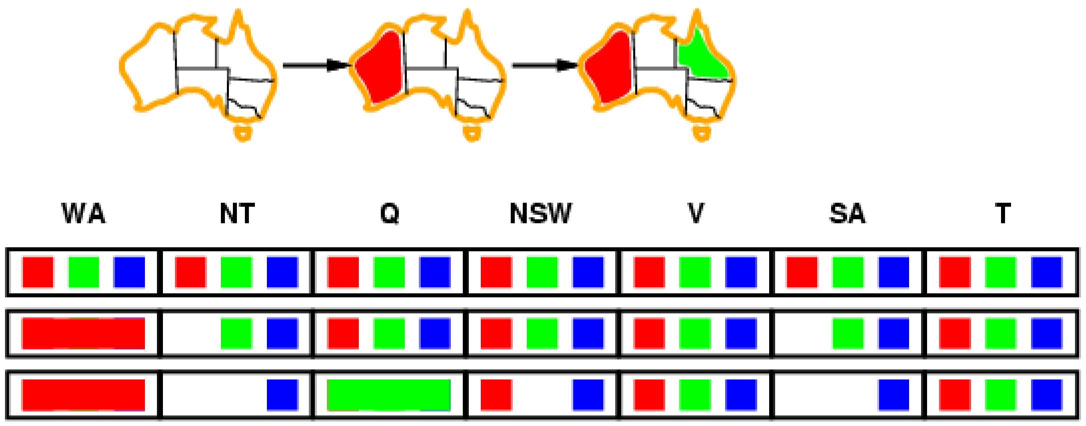




Forward Checking

- One of the simplest forms of inference is called **forward checking**.
- Idea: Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values
- Whenever a variable X is assigned, the forward-checking process establishes arc consistency for it:

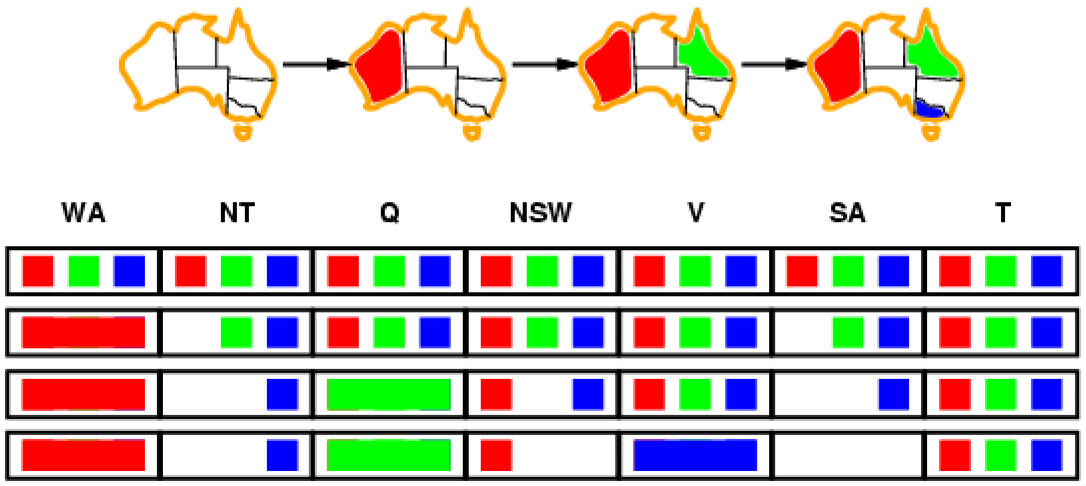
  for each unassigned variable Y that is connected to X by a constraint, delete from Y ’s domain any value that is inconsistent with the value chosen for X.



Forward Checking

- One of the simplest forms of inference is called **forward checking**.
- Idea: Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values
- Whenever a variable X is assigned, the forward-checking process establishes arc consistency for it:

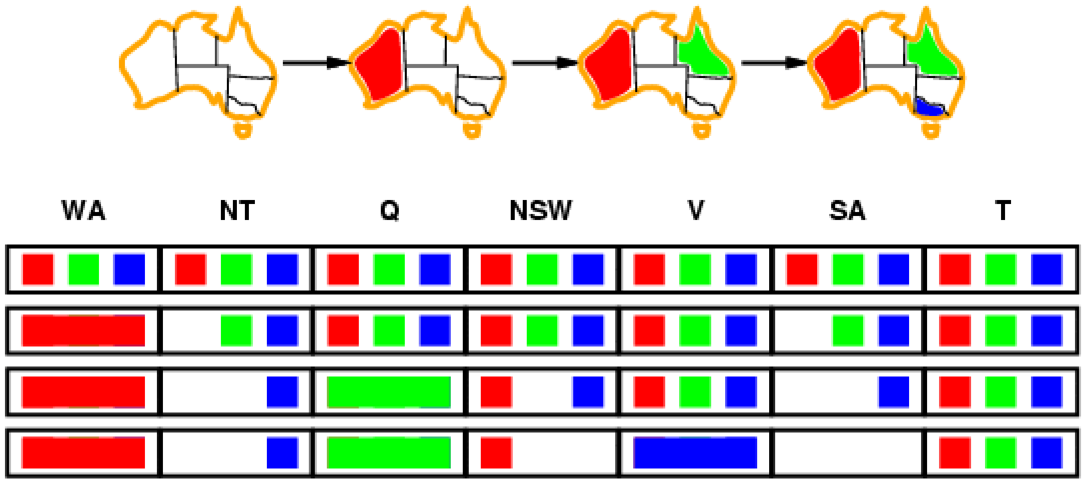
  for each unassigned variable Y that is connected to X by a constraint, delete from Y 's domain any value that is inconsistent with the value chosen for X.



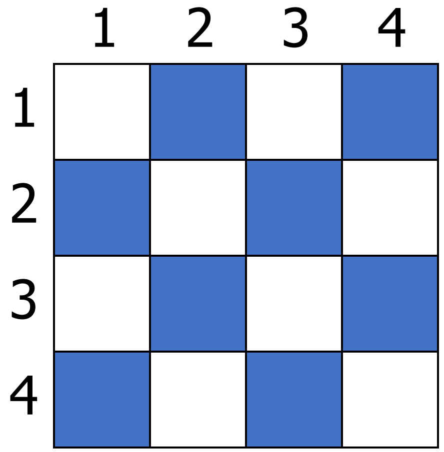
No more value for SA: backtrack

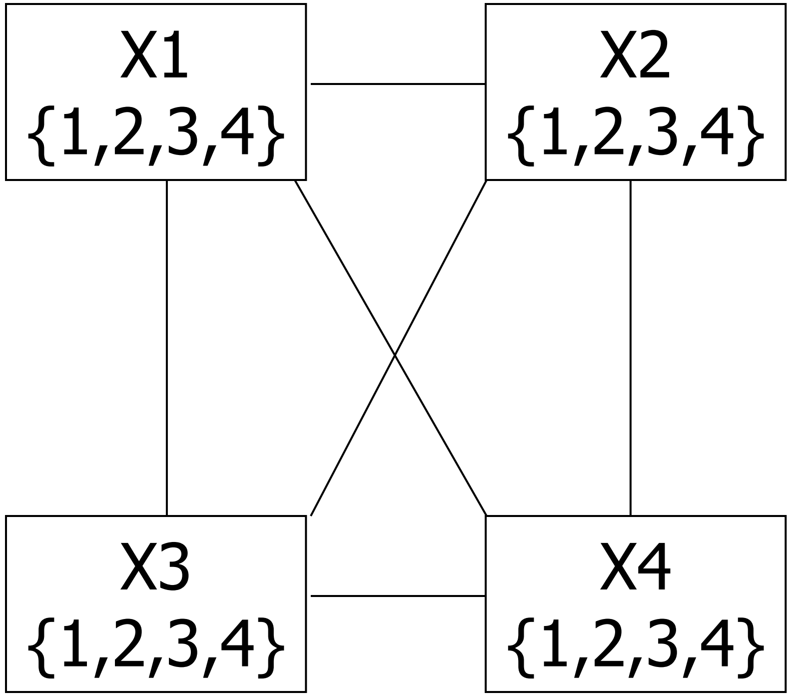
Drawback in Forward Checking

- Although forward checking detects many inconsistencies, it does not detect all of them. The problem is that it makes the current variable arc-consistent but doesn't look ahead and make all the other variables arc-consistent.
- For example, consider the Figure. It shows that when WA is red and Q is green , both NT and SA are forced to be blue.
- Forward checking does not look far enough ahead to notice that this is an inconsistency: NT and SA are adjacent (neighboring) and so cannot have the same value.



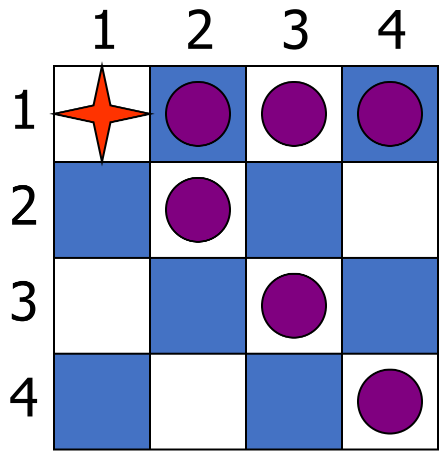
Example: 4-Queens Problem

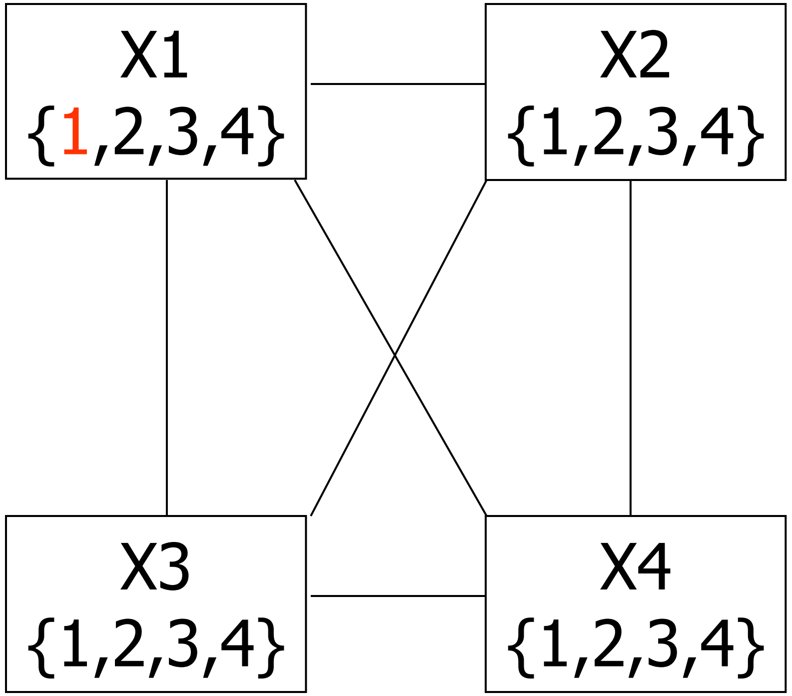




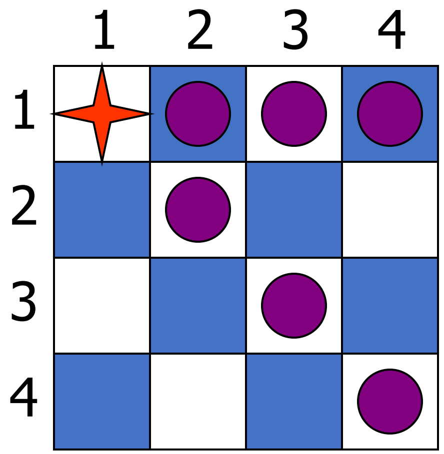
[4-Queens slides copied from B.J. Dorr]

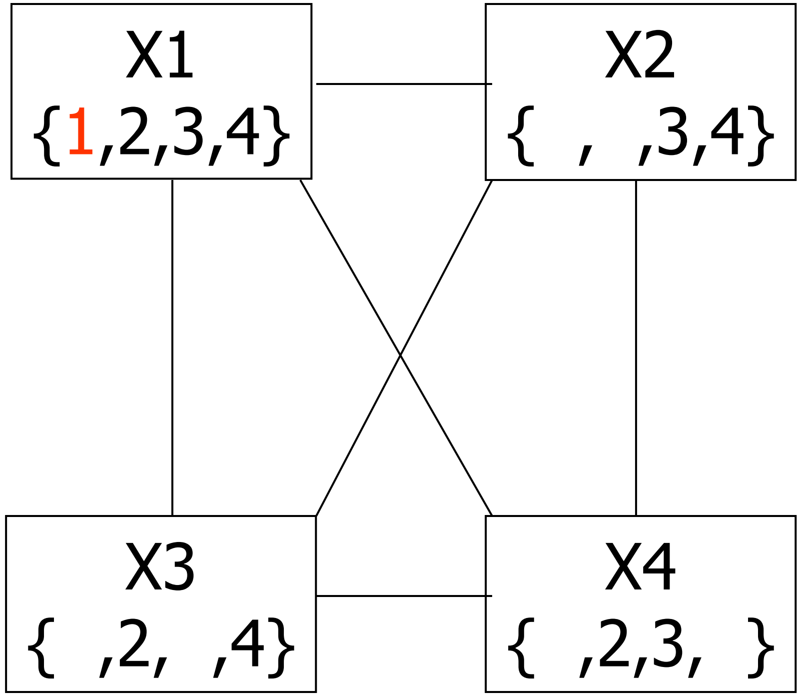
Example: 4-Queens Problem



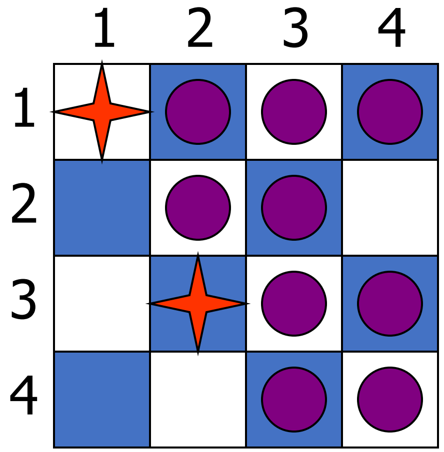


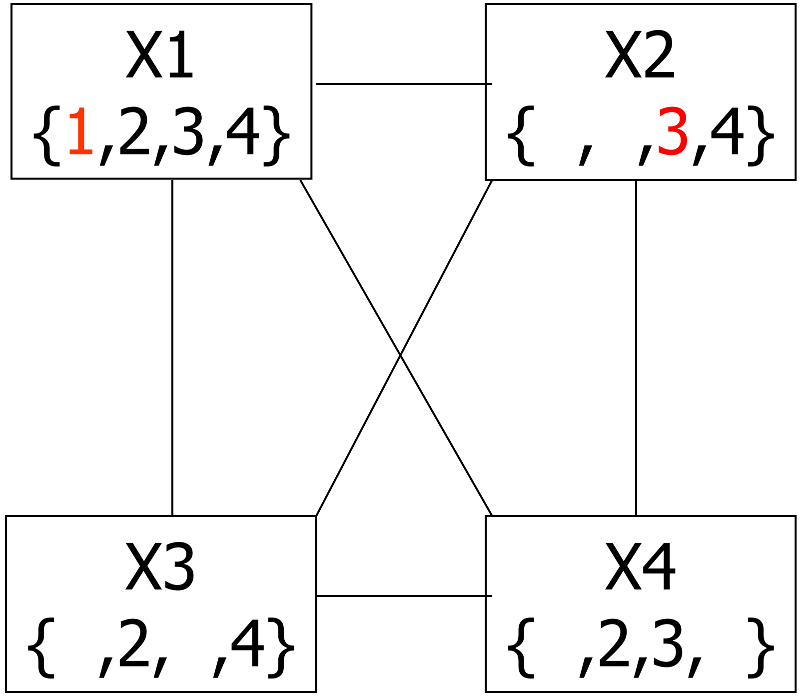
Example: 4-Queens Problem



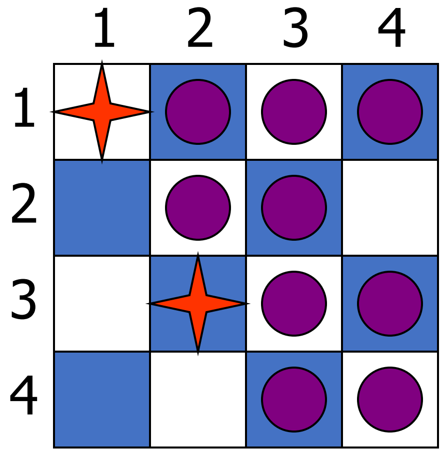


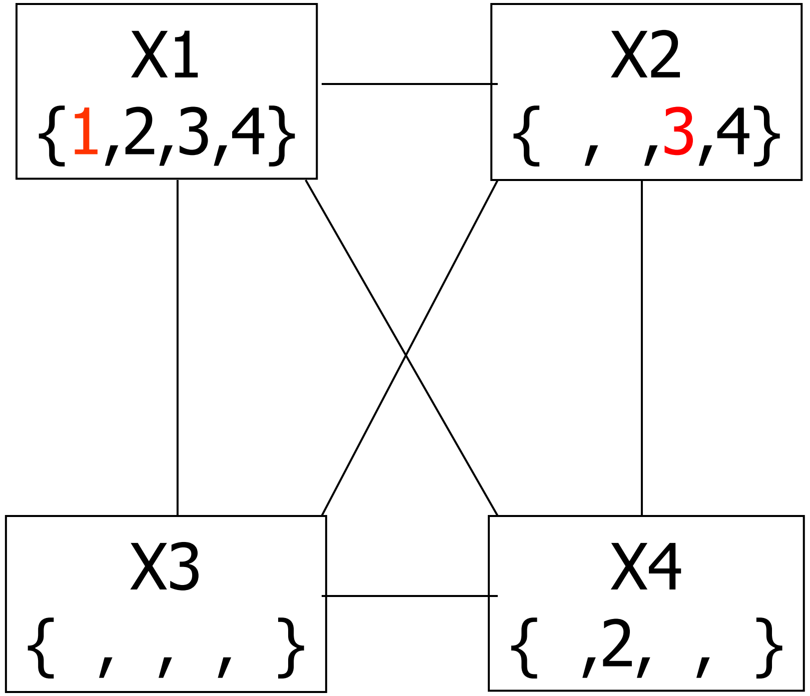
Example: 4-Queens Problem



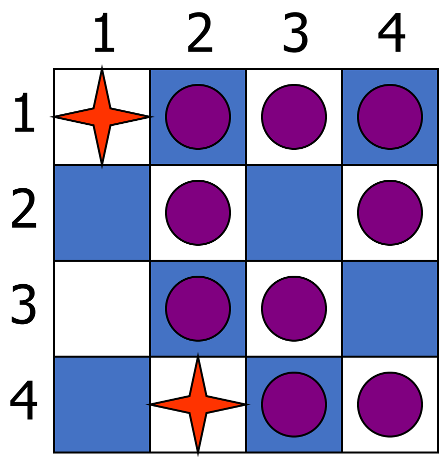


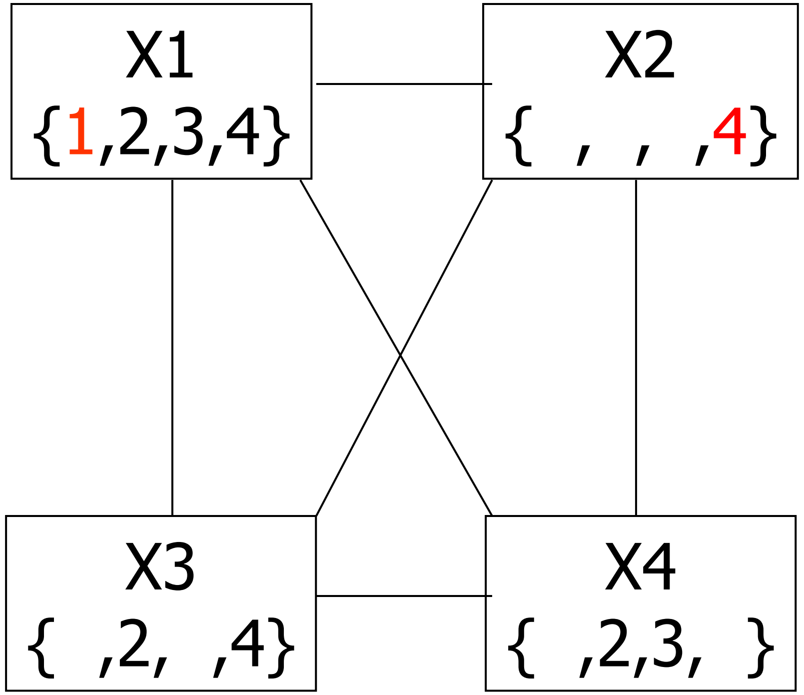
Example: 4-Queens Problem



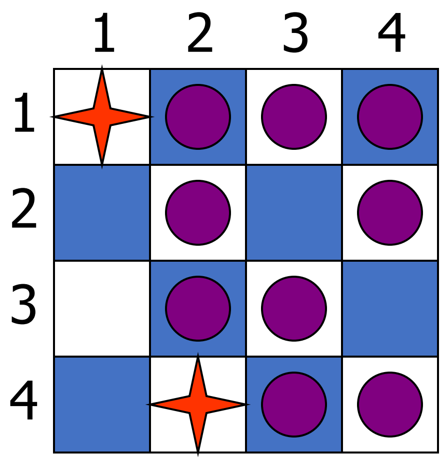


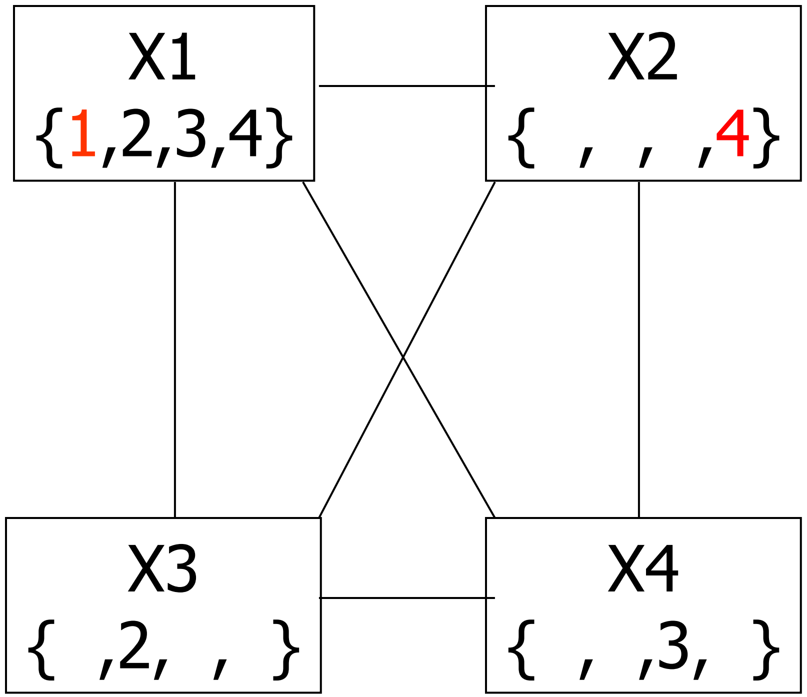
Example: 4-Queens Problem



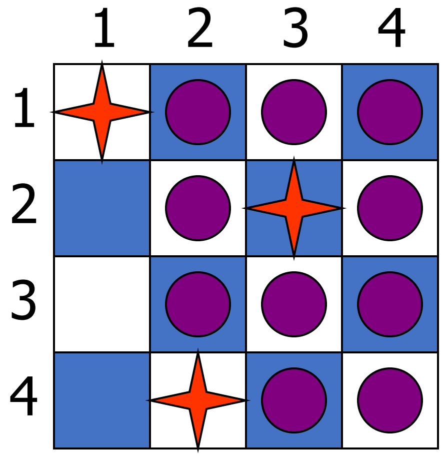


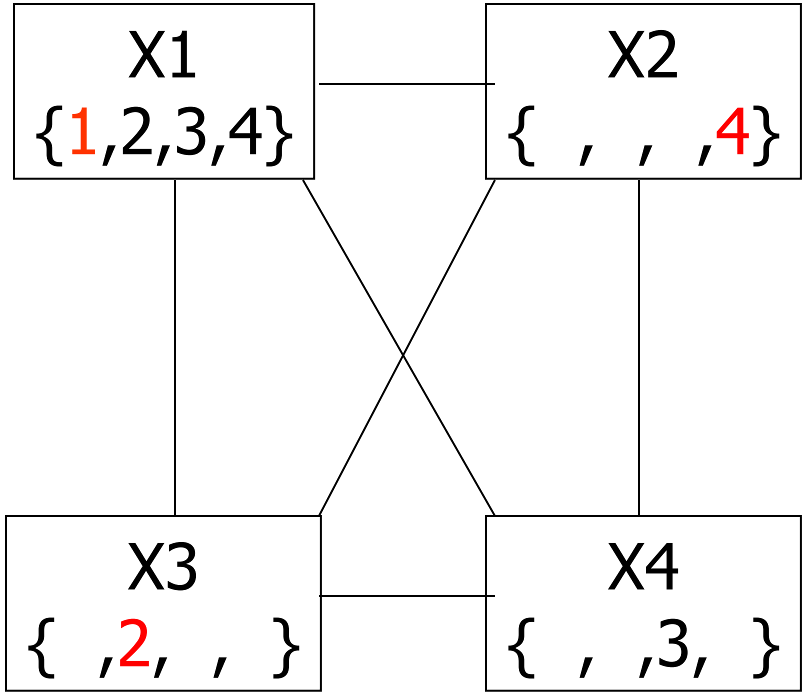
Example: 4-Queens Problem



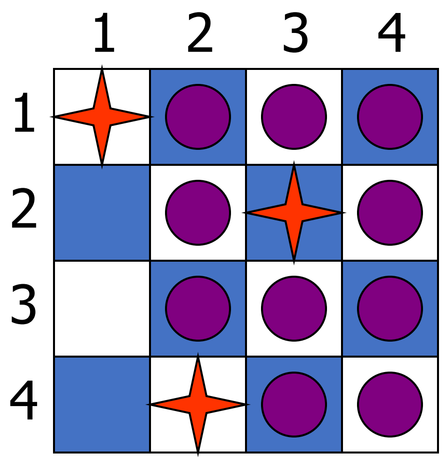


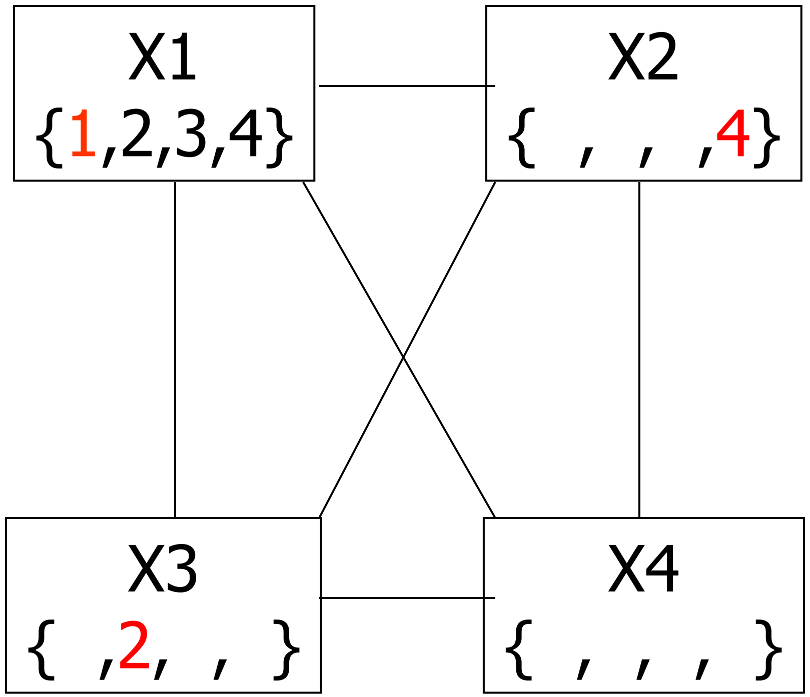
Example: 4-Queens Problem



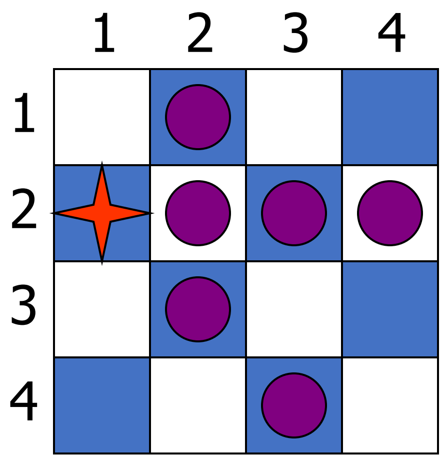


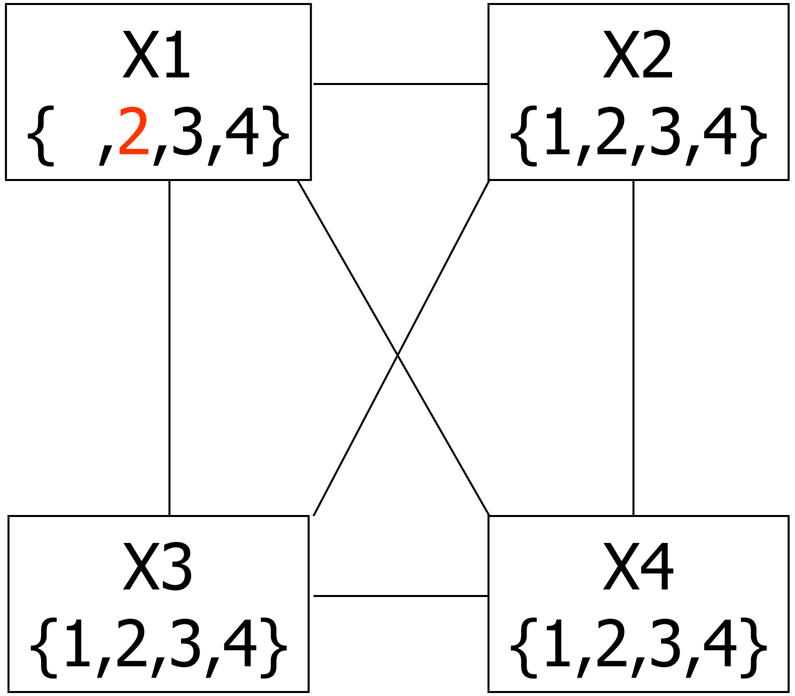
Example: 4-Queens Problem



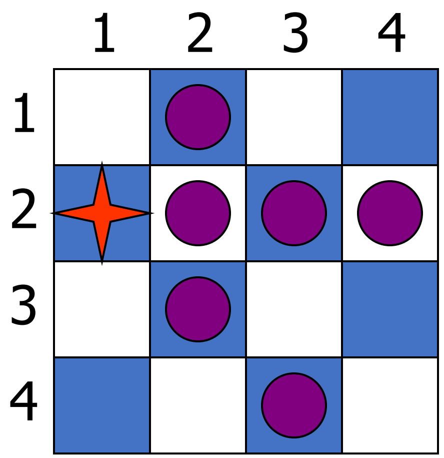


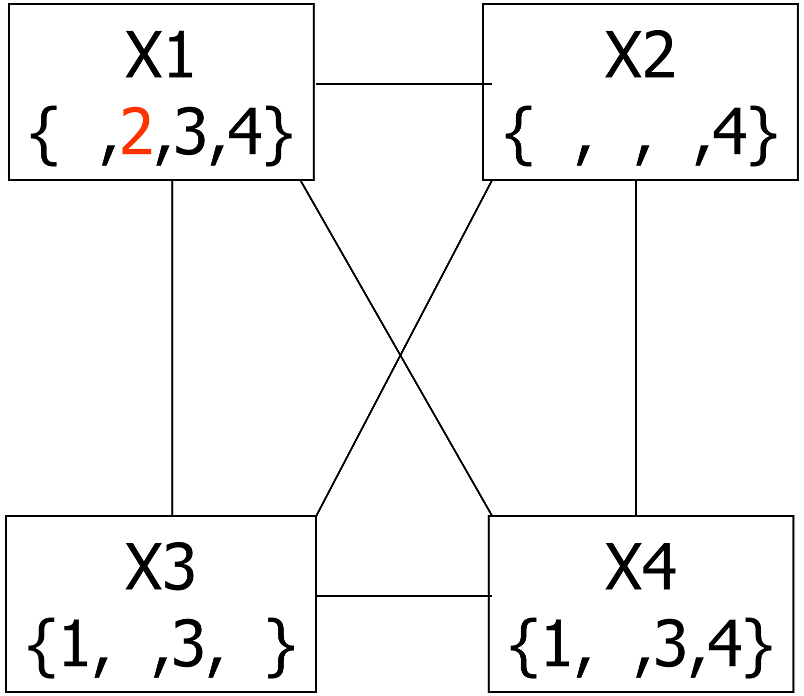
Example: 4-Queens Problem



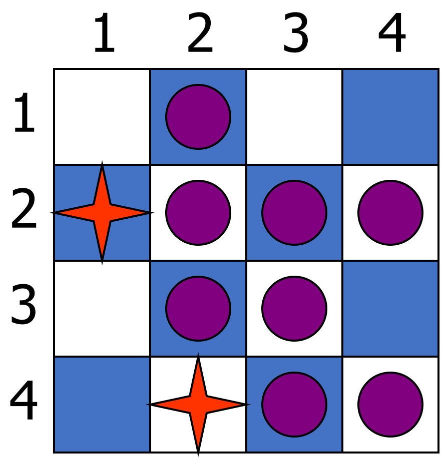


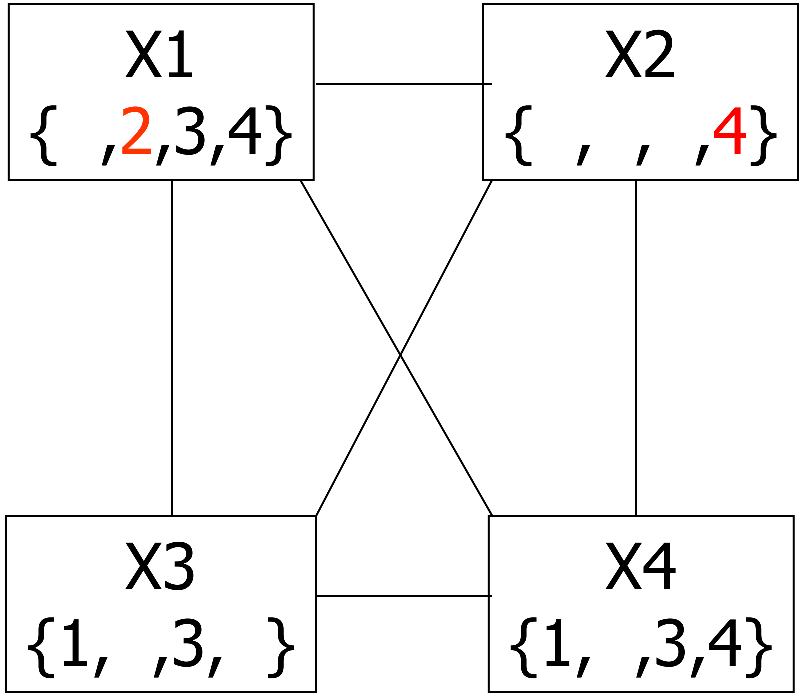
Example: 4-Queens Problem



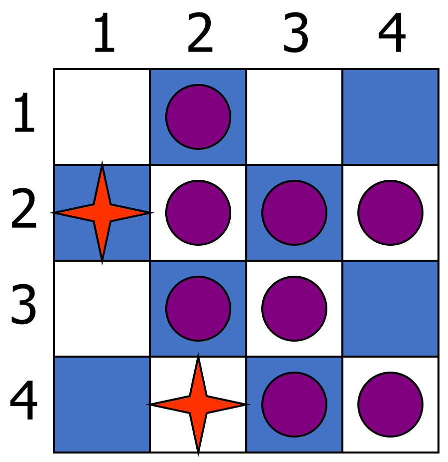


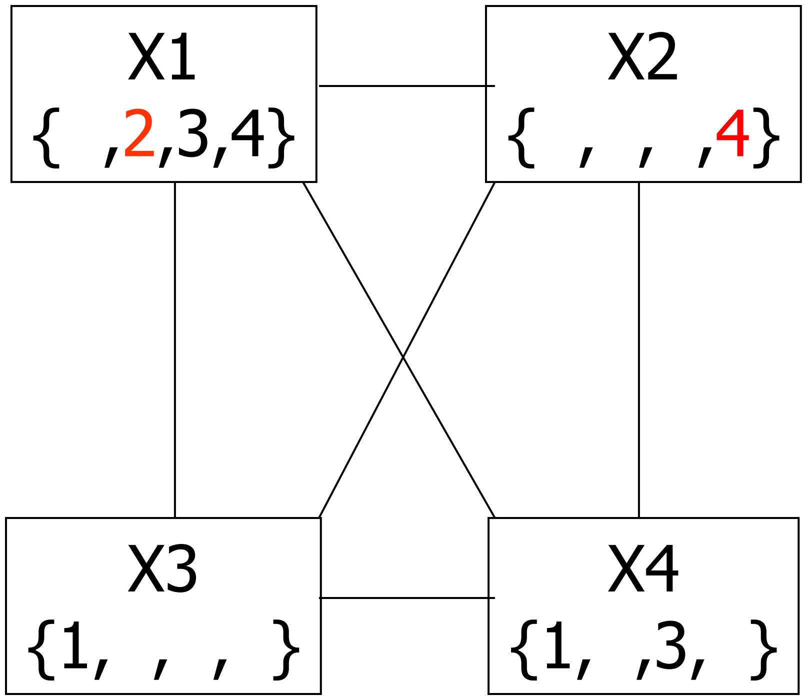
Example: 4-Queens Problem



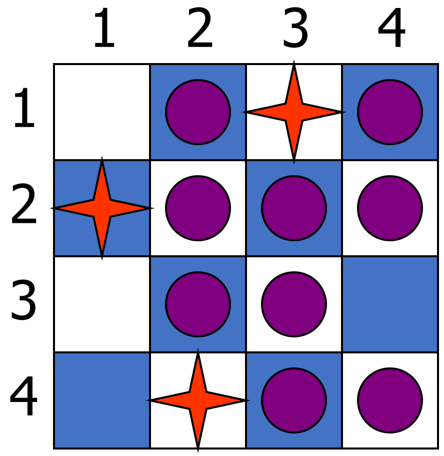


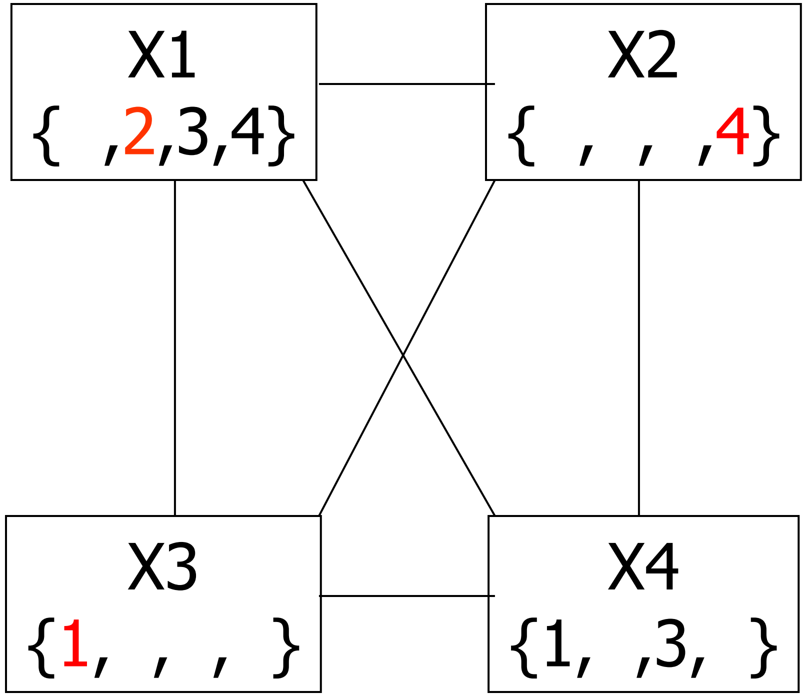
Example: 4-Queens Problem



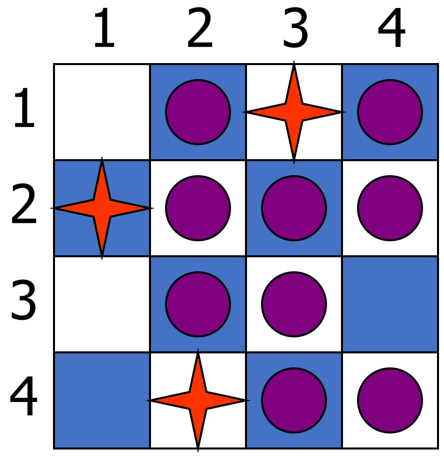


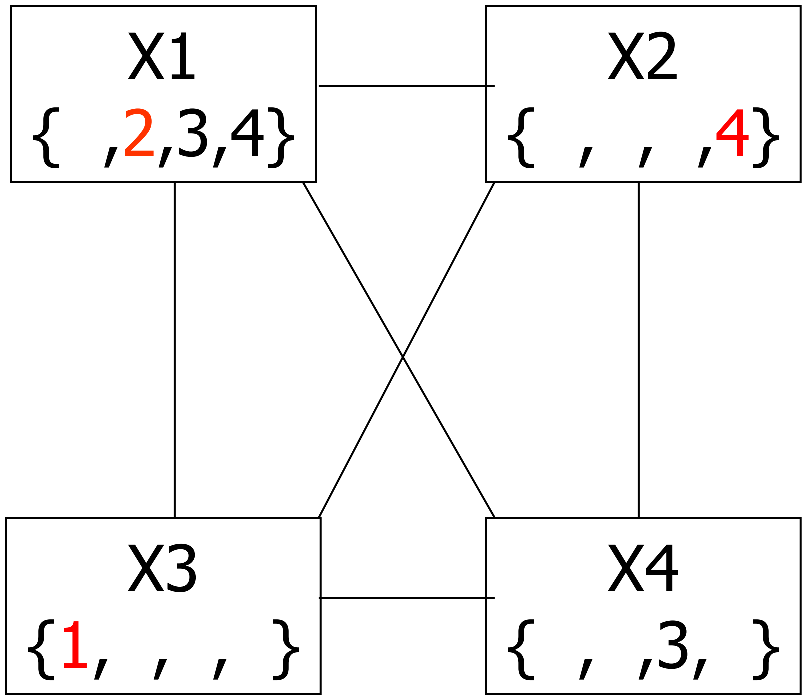
Example: 4-Queens Problem



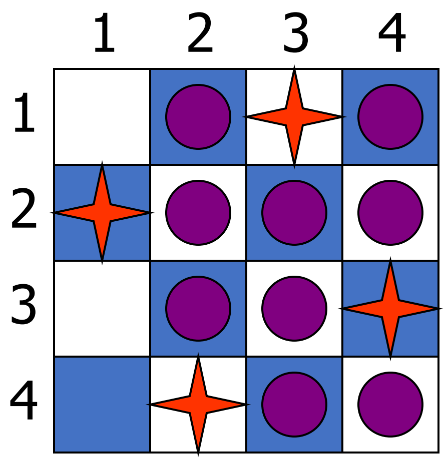


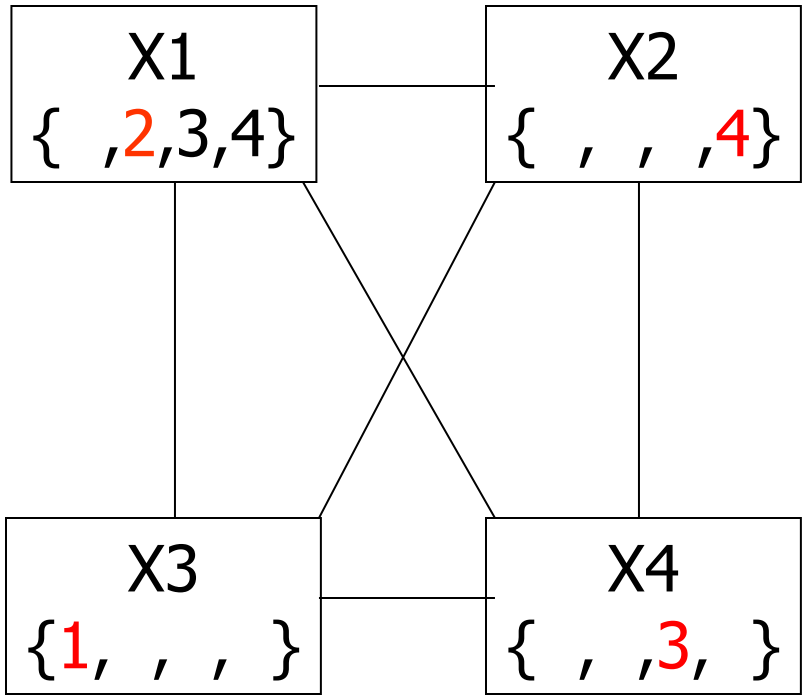
Example: 4-Queens Problem





Example: 4-Queens Problem





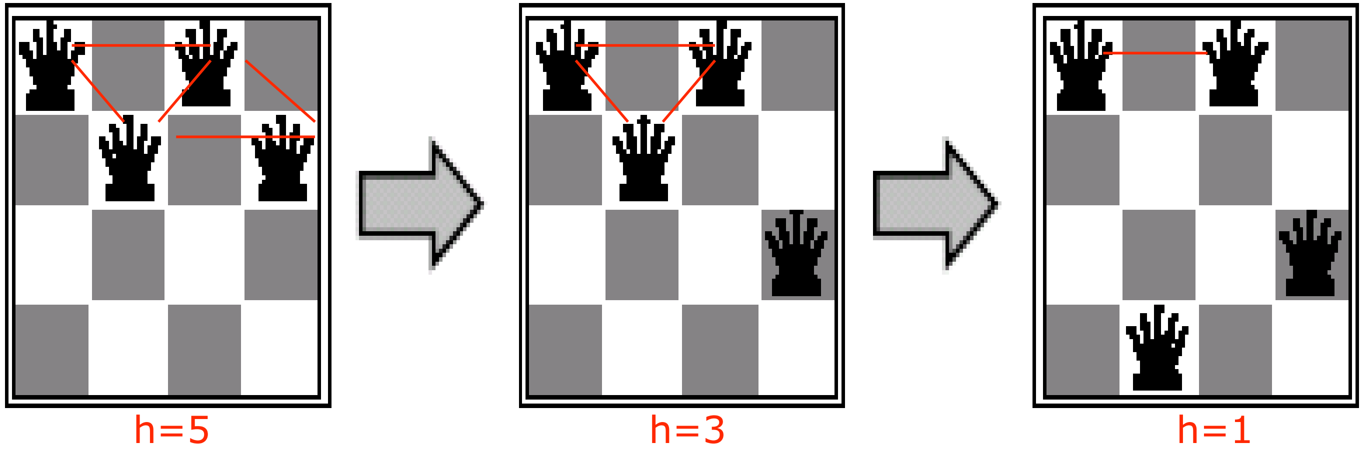
Local Search for CSPs: min-conflicts heuristic

- **Use complete-state representation**
  - Initial state = all variables assigned values
  - Successor states = change 1 (or more) values

- **For CSPs**
  - Allow states with unsatisfied constraints (unlike backtracking)
  - Operators **reassign** variable values
  - Hill-climbing with n-queens is an example

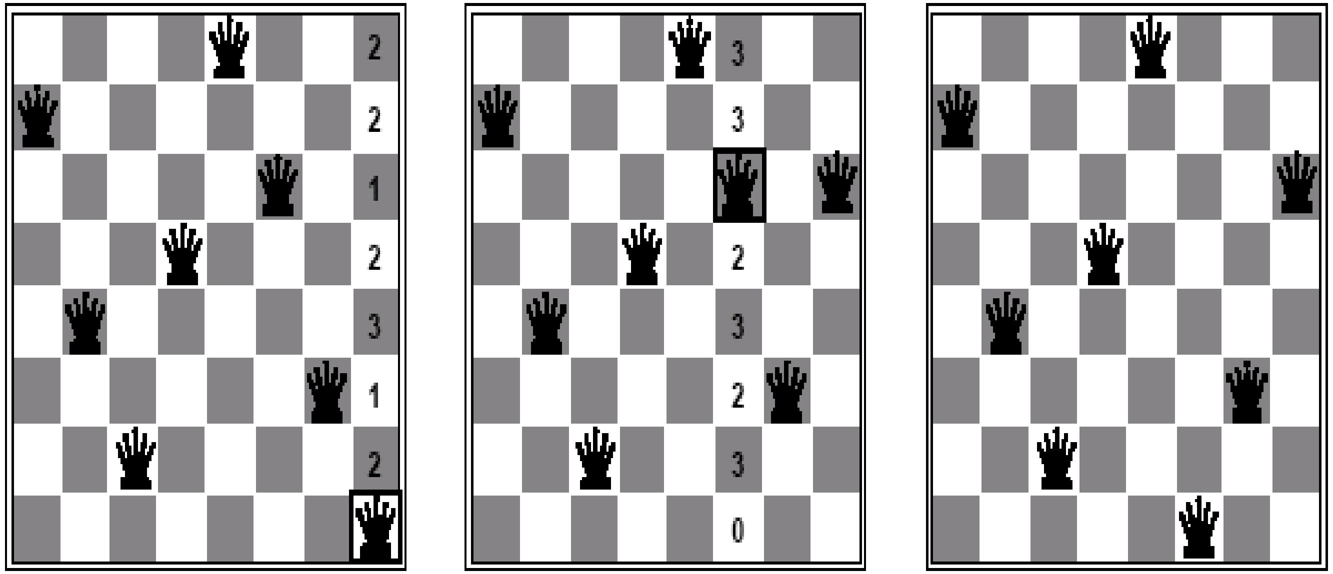
- **Variable selection**:
  - Randomly select any conflicted variable.
  - Select new value that results in a minimum number of conflicts with the other variables

Min-conflicts: example 1



Use of min-conflicts heuristic in hill-climbing.

Min-conflicts: example 2



- 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

- Local search can be particularly useful in an online setting
  - Airline schedule example
  - Much better (and faster) in practice than finding an entirely new schedule.

- The runtime of min-conflicts is roughly independent of problem size.
  - Can solve the millions-queen problem in roughly 50 steps.

Summary

- CSPs are a special kind of problem:
  - states defined by values of a fixed set of variables
  - goal test defined by constraints on variable values
- Backtracking = depth-first search with one variable assigned per node
- Variable ordering and value selection heuristics help significantly
- Forward checking prevents assignments that guarantee later failure.
- Constraint propagation (e.g., arc consistency, read it yourself) additionally constrains values and detects inconsistencies.
- Iterative min-conflicts is usually effective in practice

Contents for the next lecture

- Games as Search
- Introduction to Machine Learning
- Introduction to Deep Learning (Artificial neural network)
- Natural Language Processing (NLP)

Any questions?