ARTIFICIAL INTELLIGENCE

The Very Idea

Vasant G. Honavar
Dorothy Foehr Huck and J. Lloyd Huck Chair in Biomedical Data Sciences and Artificial Intelligence
Professor of Data Sciences, Informatics, Computer Science, Bioinformatics & Genomics and Neuroscience
Director, Artificial Intelligence Research Laboratory
Director, Center for Artificial Intelligence Foundations and Scientific Applications
Associate Director, Institute for Computational and Data Sciences
Pennsylvania State University

vhonavar@psu.edu
http://faculty.ist.psu.edu/vhonavar
http://ailab.ist.psu.edu

Problem solving through problem decomposition

- Problem
  - Solving an integral
- Sub-problems
  - Easier integrals to solve
- Actions or operators
  - Rules of integral calculus and algebra
- Primitive problems
  - Problems whose solutions can be looked up or computed by executing a known procedure

Example

- Problem
  - solving an integral
- Sub-problems
  - easier integrals to solve
- Operators
  - rules of integral calculus and algebra
- Primitive problems
  - problems whose solutions can be looked up or computed by executing a known procedure

$$\int (2t + sec^2t\ sin^2t + 7\ )dt$$

AND

$$\int 2t\ dt + \int sec^2t\ sin^2t\ dt + \int 7\ dt$$

OR

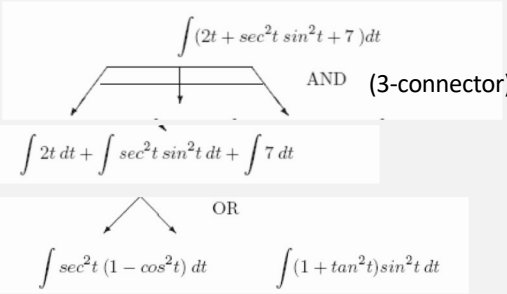
$$\int sec^2t\ (1 - cos^2t)\ dt \qquad \int (1 + tan^2t) sin^2t\ dt$$

Problem reduction representation (PRR)

- A PRR problem is specified by a 3-tuple (*G, O, P*)
  - *G* is a problem to be solved
  - *O* is a set of operators for decomposing problems into sub-problems through AND or OR decompositions
  - *P* is a set of primitive problems with known solutions
- Solution
  - An AND decomposition is solved when each of the sub-problems is solved
  - An OR decomposition is solved when at least one of the sub-problems is solved
  - A problem is unsolvable if it is neither a primitive problem nor can it be further decomposed
- PRR is a generalization of the state space representation (why?)

Problem reduction representation

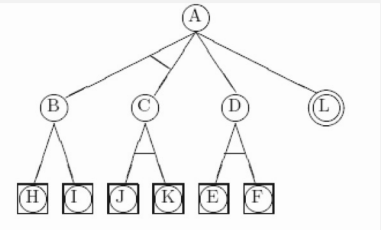
- Solving a problem in PRR reduces to searching an AND-OR graph
- Nodes correspond to problems
- Connectors correspond to actions
- Connectors correspond to AND or OR decompositions
- Connectors of arity *k* are called *k*-connectors

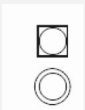


(3-connector)

Example

Problem

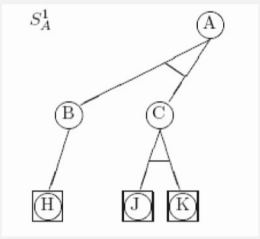


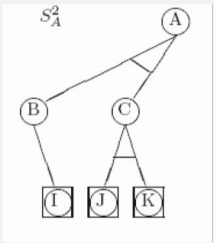


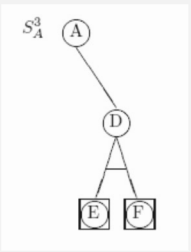
Primitive problem

Unsolvable problem

3 solutions







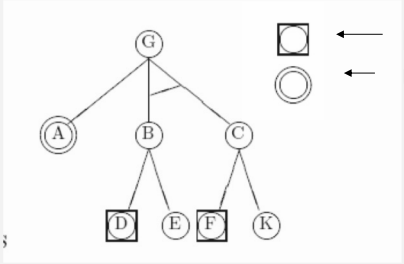
Solution to an PRR problem

- A sub-graph s_q of an AND-OR graph is said to be solution to a problem q if
  - s_q is rooted at q
  - Each non-leaf node y in s_q, has exactly one connector out of it that belongs to s_q
  - Each leaf node in s_q is a primitive problem (i.e. a member of P)
- A problem q is said to be solvable if
  - a sub-graph s_q of an AND-OR graph is a solution to q
- Solving a problem G using a PRR (G, O, P) entails finding a sub-graph S_G of the corresponding AND-OR graph that is a solution of G

Question – How can we solve an PRR problem?

- Basic idea:
  - Generalize state-space search
- How?
  - Generalize partial paths to sub graphs of the PRR AND-OR graph
  - Expanding a node must comply with the semantics of AND and OR connectors
  - Termination test must comply with the definition of a solution

Example – BFS



Primitive problem

Unsolvable problem

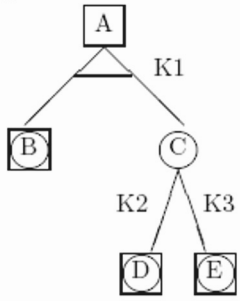
List

(G)
(A) (B & C)
(B & C)
(D & F) (E & F) (D & K) (E & K)

Exercise: Solve the same problem using DFS

Optimal (minimum cost) solution of AND-OR graphs

Example:

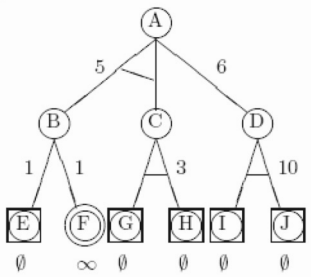


- Cost of an unsolvable primitive problem – infinity
- Cost of connectors and primitive problems are assumed to be strictly positive and bounded

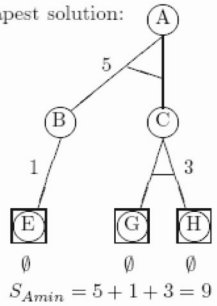
$$Cost(S_A) = Cost(k_1) + Cost(S_B) + Cost(S_C)$$

Optimal solution of an SRR problem

Example:

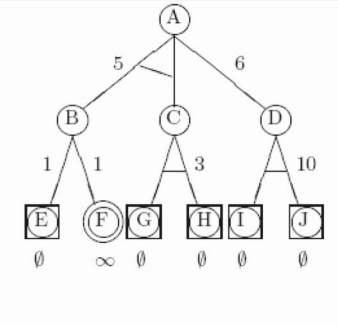


Cheapest solution:



$S_{Amin} = 5 + 1 + 3 = 9$

Branch and Bound Search for Optimal Solution



List

(A, 0)

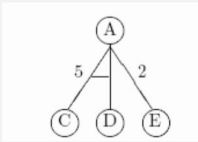
(B & C, 5) (D, 6)

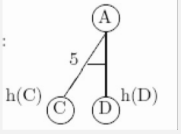
(E & C, 6)(D, 6) (F & C, ∞)

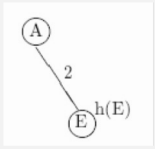
(D, 6) (E & G & H, 9) (F & C, ∞)

(E & G & H, 9) (I & J, 16) (F & C, ∞)

Using Heuristics







$$f(C\&D) = 5 + h(C) + h(D)$$

$$f(E) = 2 + h(E)$$

Admissible heuristic function

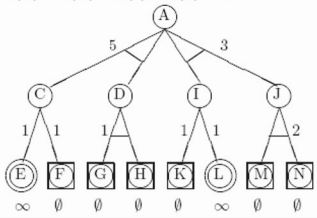
$h(n) \le h^*(n) = Cost(n)$ ← Cost of the cheapest solution of *n*

AO* - Searching AND-OR graphs

Example:

$h(A) = 0$

$h(C) = h(D) = h(I) = h(J) = 1$



(A, 0+0)

(I & J, 3+2) (C & D, 5+2)

(K & M & N,6) (C & D, 5+2) (L & M & N, ∞)

Properties of AO*

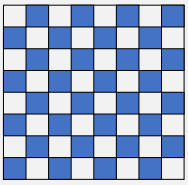
- AO* is a generalization of A* for AND-OR graphs
- AO*, like A*, is admissible if the heuristic function is admissible under the usual assumptions
  - finite branching factor
  - strictly positive action costs
- AO*, like A* is also optimal among the class of heuristic search algorithms that use an additive cost evaluation function

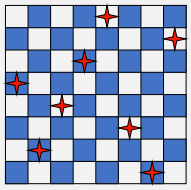
Solving Constraint Satisfaction Problems

Constraint Satisfaction Problem (CSP)

- Special class of state-space search problems
- States are defined by values assigned to some or all of the variables $\{X_1, X_2, \ldots, X_n\}$
  - Each variable X_i takes values from a domain D_i
  - In the most common setting, D_i is finite
- The assignment of values to variables is subject to a set of constraints $\{C_1, C_2, \ldots, Cp\}$
  - Each constraint relates a subset of variables by specifying the valid combinations of their values
- Solution
  - A goal state in which every variable has a value assigned to it and the assignment satisfies the specified constraints
  - Note: We don't care about the path from start state to goal state, only the assignment of variables in the goal state

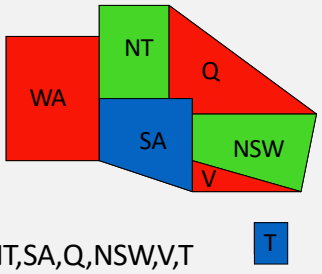
Example: Constraint satisfaction problem





- Variables: rows $Q_1 \cdots Q_8$
- Domain of each variable represents column in which a queen is placed : $\{1 \cdots 8\}$
- Constraints
  - for all $i < j$
    - $Q_i \neq Q_j$ (queens cannot share a row)
    - $|Q_i - Q_j| \neq |i - j|$ (no two queens can share a diagonal)
- Solution
  - An assignment of values to $Q_1 \cdots Q_8$ that satisfies the constraints

Example: Map Coloring Problem



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

- Variables: WA,NT,SA,Q,NSW,V,T
- Domain of each variable: {red, green, blue}
- Constraints: No two adjacent variables can have the same value:

  WA≠NT, WA≠SA, NT≠SA, NT≠Q, SA≠Q,
  SA≠NSW, SA≠V, Q≠NSW, NSW≠V

Example: Course scheduling

- Variables: Courses
- Domains of variables: Cross product of rooms, timeslots and instructors
- Constraints:
  - No two courses can share the same instructor and time slot
  - No two courses can share the same room and time slot
- Solution: An assignment of courses to rooms, timeslots and instructors that satisfies the constraints

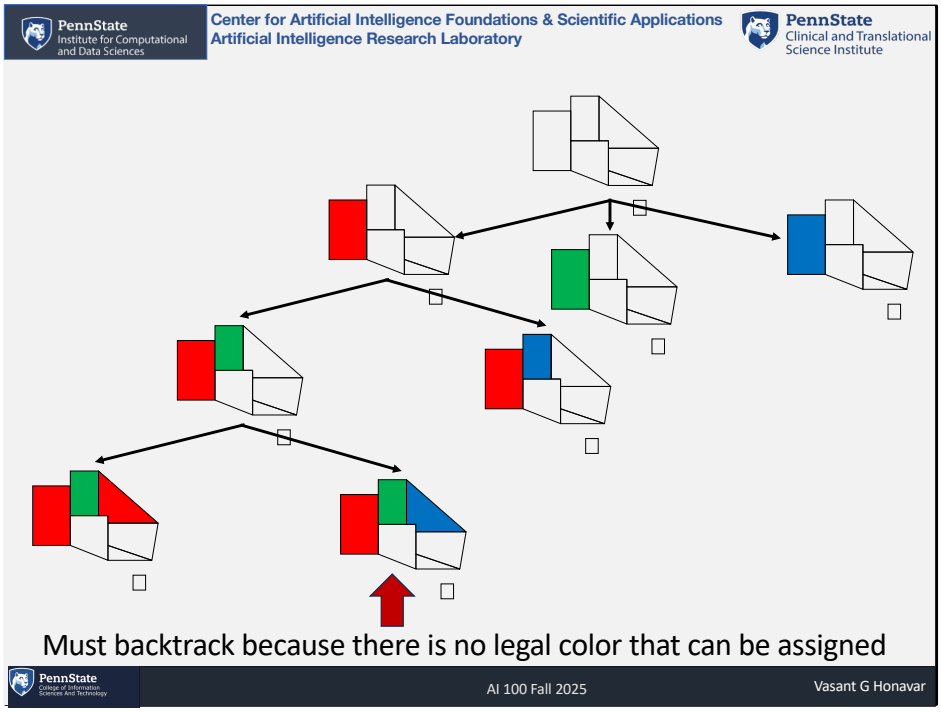
CSP as a Search Problem

- n variables $X_1, \ldots, X_n$
- Valid assignment: Assignment of values to the variables without violating any of the constraints
- Complete assignment: one where each variable has a value
- States: valid assignment (no violation of constraints)
- Initial state: empty assignment
- Successor of a state: assign a value to a variable without a value assignment
- Goal test: complete assignment (which by design is also a valid assignment)

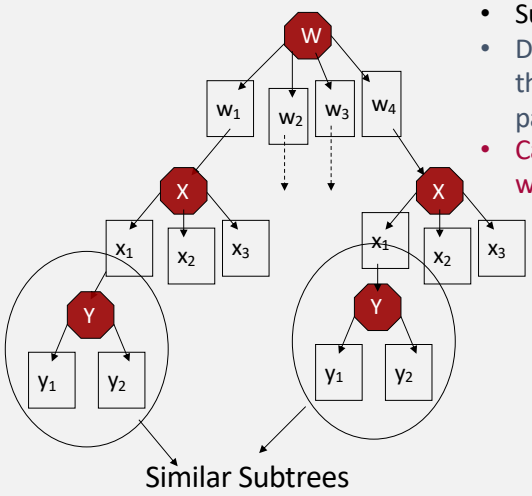
Key properties of (Discrete) CSP

- Search space is finite
- The order in which variables are assigned values has no impact on the reachable complete valid assignments
  - One can expand a node by first selecting any unassigned variable and assign it a value from its domain without violating the constraints
  - Big reduction in branching factor compared to standard search
- The solution is always of fixed depth = number of variables
- The path from start state to goal state does not matter
- All we care about is satisfying the constraints on the goal
  - We can use a simplified depth-first search with backtracking

Example: Backtracking search



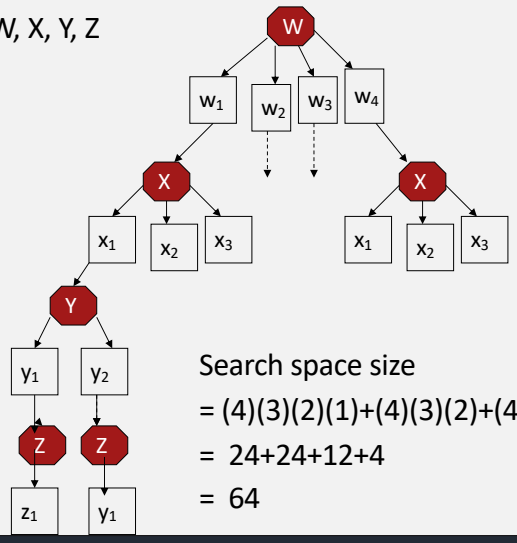
Sub-trees have similar topologies



- Suppose incompatible with $Y = y_1$
- Depth first search will rediscover this incompatibility in different parts of the state space
- Can we avoid this unnecessary work?

Variable ordering impacts size of search space

Variable ordering W, X, Y, Z



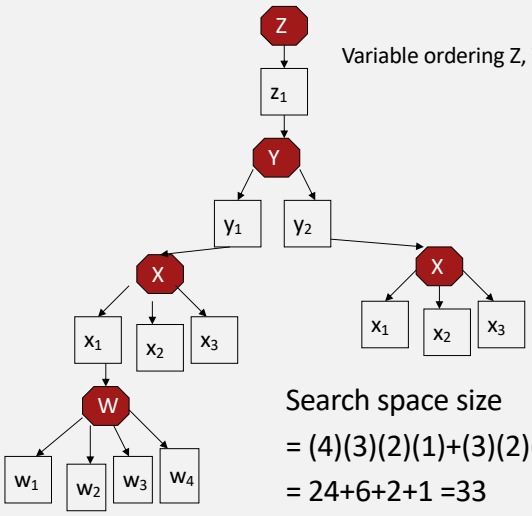
Search space size

= (4)(3)(2)(1)+(4)(3)(2)+(4)(3)+(4)

= 24+24+12+4

= 64

Variable ordering and Search space size



Variable ordering Z, Y, Z, W

Search space size

= (4)(3)(2)(1)+(3)(2)(1)+(2)(1)+1

= 24+6+2+1 =33

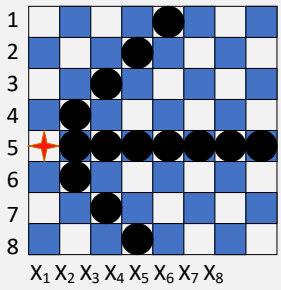
Backtracking search

- Standard backtracking fails to exploit special properties of CSP
  - Subtrees have similar topologies
    - Can we eliminate the duplicate work of rediscovering incompatible assignments?
    - Yes, if we can propagate constraints
  - Search space has minimal size under a certain ordering of variables (most constrained to least constrained)
    - Can we consider the variables and value assignments in some order that effectively minimizes the size of the search space to be considered?

Propagating constraints: Forward checking

Avoiding rediscovering incompatible assignments

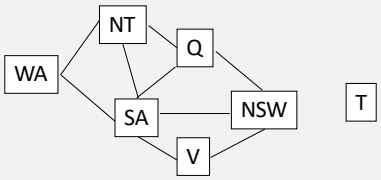
- Propagate the constraints



Assigning the value 5 to X_1 leads to removing values from the domains of $X_2, X_3, \cdots X_8$

Constraint propagation through forward checking

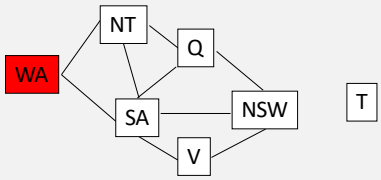
Constraint graph facilitates constraint propagation



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB

Constraint propagation through forward checking

Constraint graph facilitates constraint propagation

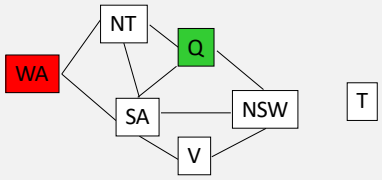


WA	NT	Q	NSW	V	SA	T
RGB	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	GB	RGB	RGB	RGB	GB	RGB

Forward checking removes the value Red of NT and of SA

Constraint propagation through forward checking

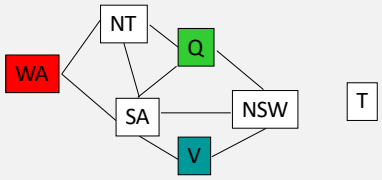
Constraint graph facilitates constraint propagation



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	~~G~~B	G	R~~G~~B	RGB	~~G~~B	RGB

Constraint propagation through forward checking

Constraint graph facilitates constraint propagation



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB
R	B	G	~~RB~~	B	~~B~~	RGB

Constraint propagation through forward checking

Constraint graph facilitates constraint propagation

Empty set: the current assignment {(WA ← R), (Q ← G), (V ← B)} does not lead to a solution

WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB
R	B	G	~~RB~~	B	~~B~~	RGB

Constraint propagation through Forward Checking

Whenever a pair $(X \leftarrow v)$ is added to assignment do:

For each variable Y not in the assignment:

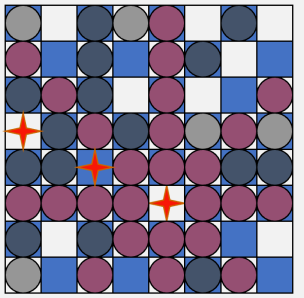
For every constraint C relating Y to the variables in the assignment:

Remove all values from Y's domain that do not satisfy C

Modified Backtracking Algorithm

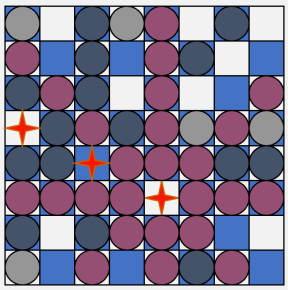
- Recall that search space is minimized by appropriate choice of the order in which variables and values are considered
- Which variable X_i should be assigned a value next?
  - Most-constrained-variable heuristic
    - Consider the variable with the fewest values in its domain
  - Most-constraining-variable heuristic
    - Consider the variable that will constrain the values of the largest number of variables
- In which order should its values be assigned?
  - Least-constraining-value heuristic
    - Assign the value that allows the greatest flexibility in assigning values to the remaining variables

8-Queens



4 3 2 3 4 ← Numbers of values for each un-assigned variable

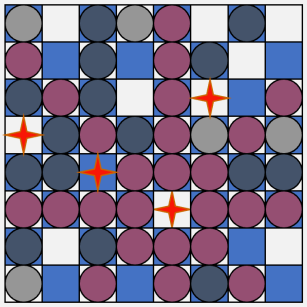
8-Queens



4 3 2 3 4 ← Numbers of values for each un-assigned variable

New assignment

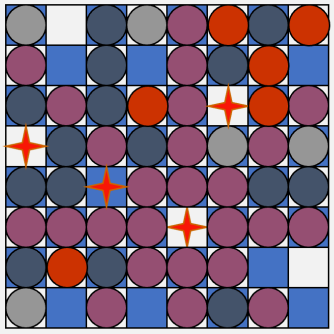
8-Queens



4 3 2 3 4 ← Numbers of values for each un-assigned variable

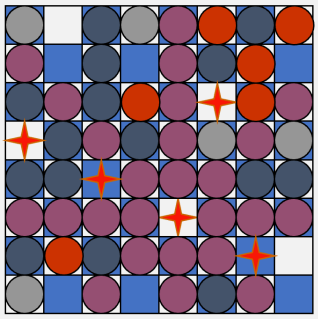
New assignment

8-Queens



3 2 1 3 ← New numbers of values for each un-assigned variable

8-Queens

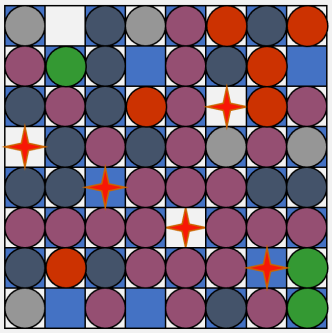


New assignment

3 2 1 3

New numbers of values for each un-assigned variable

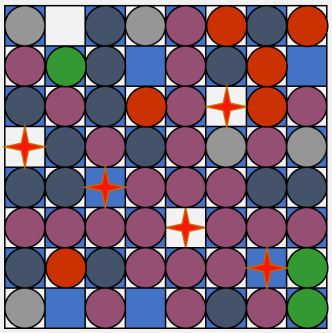
8-Queens



Forward checking

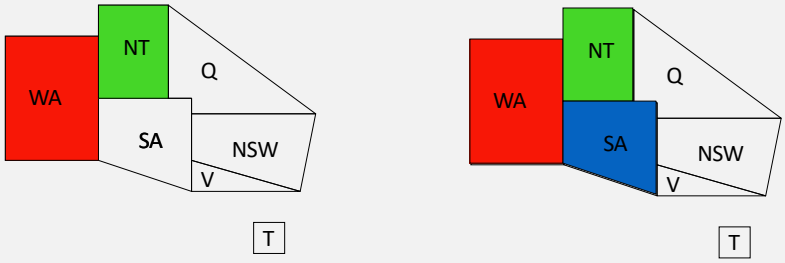
New assignment

8-Queens



--------- Next placement

Map Coloring



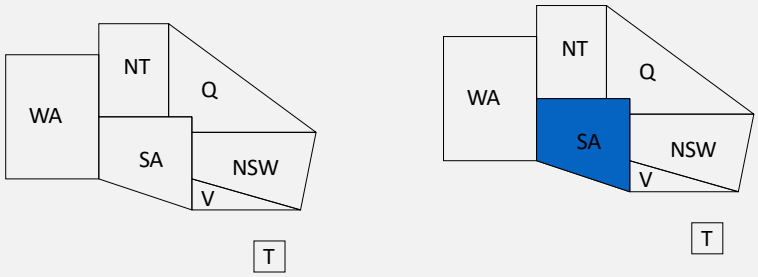
- SA's remaining domain has size 1 (value Blue remaining)
- Q's remaining domain has size 2
- NSW's, V's, and T's remaining domains have size 3

Hence, we select SA

Most-Constraining-Variable Heuristic

- Which variable *X* should be assigned a value next?
- Among the variables with the smallest remaining domains (ties with respect to the most-constrained-variable heuristic)
  - select the one that appears in the largest number of constraints on variables not in the current assignment
- Rationale: Increase future elimination of values, to reduce future branching factors

Map Coloring



- Before any value has been assigned, all variables have a domain of size 3, but SA is involved in more constraints (5) than any other variable

Hence, select SA and assign a value to it (e.g., Blue)

Least-Constraining-Value Heuristic

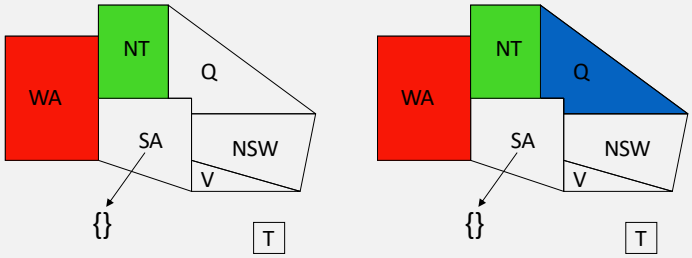
- In which order should X's values be assigned?

  Select the value of X that removes the smallest number of values from the domains of those variables which are not in the current assignment

  Rationale: Since only one value will eventually be assigned to X, pick the least-constraining value first, since it is the most likely not to lead to an invalid assignment

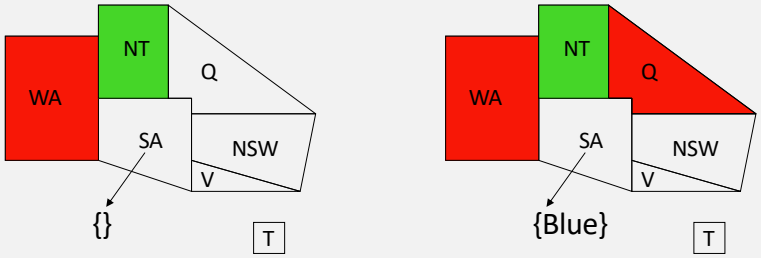
  Note: Using this heuristic requires performing a forward-checking step for every value, not just for the selected value

Map Coloring



- Q's domain has two remaining values: Blue and Red
- Assigning Blue to Q would leave 0 value for SA, while assigning Red would leave 1 value

Map Coloring



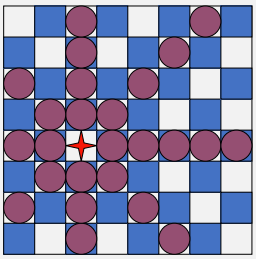
- Q's domain has two remaining values: Blue and Red
- Assigning Blue to Q would leave 0 value for SA, while assigning Red would leave 1 value

Hence, assign Red to Q

Applications of CSP

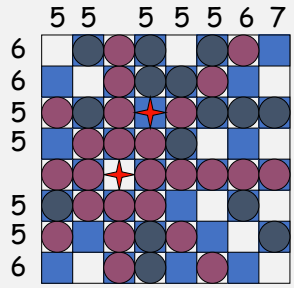
- CSP techniques are widely used
- Applications include:
  - Course scheduling
  - Crew assignments to flights
  - Management of transportation fleet
  - Flight/rail schedules
  - Job shop scheduling
  - Task scheduling in port operations
  - Design, including spatial layout design

Constraint Propagation



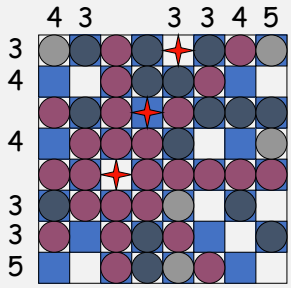
- Place a queen in a square
- Remove the attacked squares from future consideration

Constraint Propagation



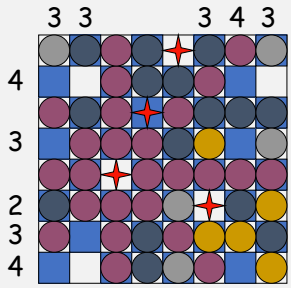
- Count the number of non-attacked squares in every row and column
- Place a queen in a row or column with minimum number
- Remove the attacked squares from future consideration

Constraint Propagation

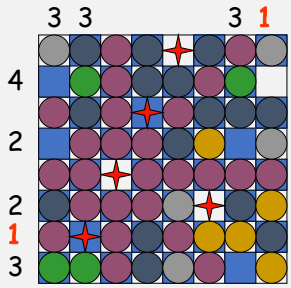


- Repeat

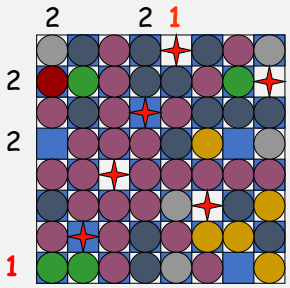
Constraint Propagation



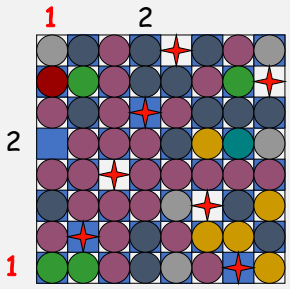
Constraint Propagation



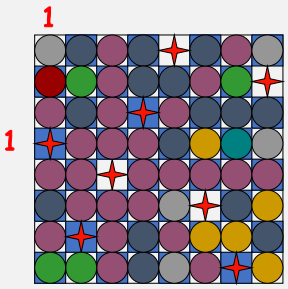
Constraint Propagation



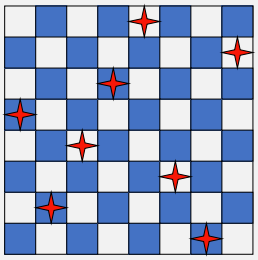
Constraint Propagation



Constraint Propagation



Constraint Propagation



Constraint propagation

Constraint propagation finds broad applications

- Scheduling
  - Surgeries
  - Jobs to machines
  - Courses
  - Rooms
- Map coloring
- Layout Problems
  - Road layout
  - VLSI layout
  - Floor plan layout
- 3D interpretation of 2D drawings
- ….