



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>

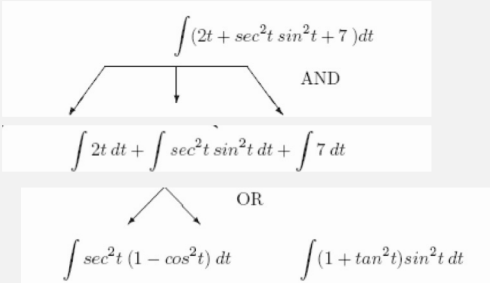
Problem solving through problem decomposition

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

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

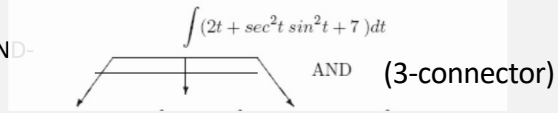


Problem reduction representation (PRR)

- 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

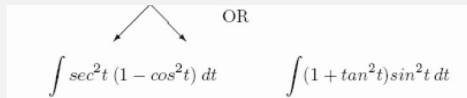
- Solving a problem in PRR reduces to searching an AND-OR graph



- Nodes correspond to problems

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

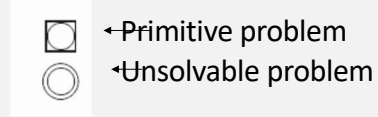
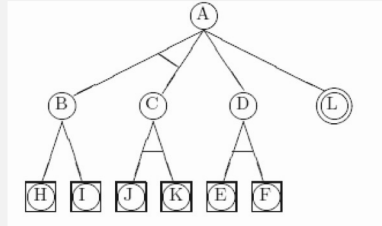
- Connectors correspond to arcs



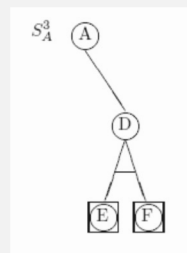
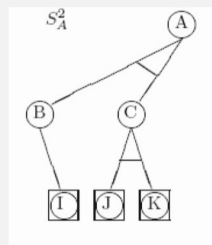
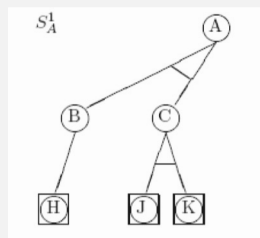
- Connectors correspond to AND or OR decompositions
- Connectors of arity k are called k -connectors

Example

Problem



3 solutions



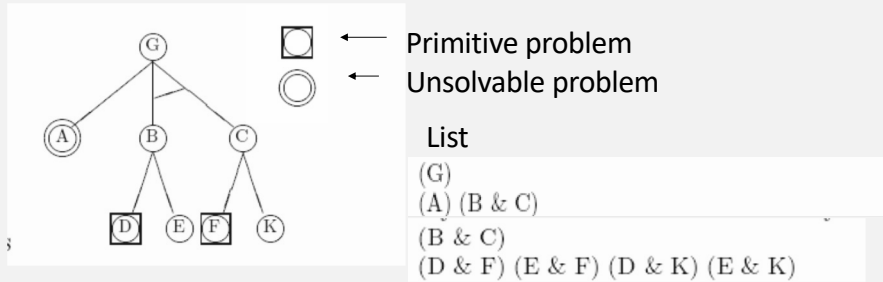
Solution to an SRR problem

- 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 SRR problem?

- Basic idea:
 - Generalize state-space search
- How?
 - partial paths → sub graphs of the SRR 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

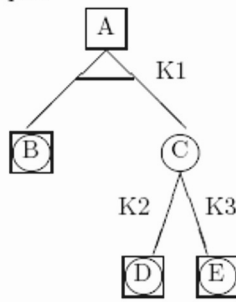
Example – BFS



Exercise: Solve the same problem using DFS

Optimal (minimum cost) solution of AND-OR graphs

Example:

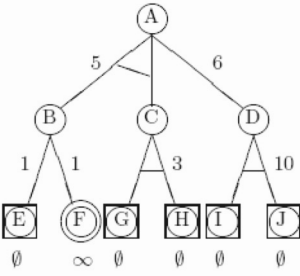


Cost of an unsolvable primitive problem – infinity
Cost of connectors and primitive problems are assumed to be strictly positive and bounded

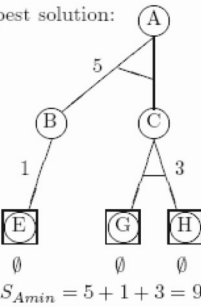
$$Cost(S_A) = Cost(k_1) + Cost(S_B) + Cost(S_C)$$

Optimal solution of an SRR problem

Example:

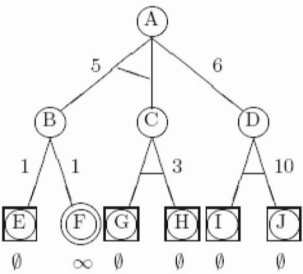


Cheapest solution:



Branch and Bound Search for Optimal Solution

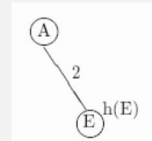
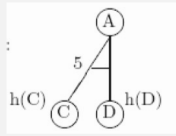
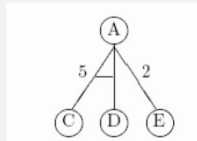
Example:



- List
- (A)
 - (B & C) (D)
 - (E & C) (D) (F & C)
 - (D) (E & G & H) (F & C)
 - (E & G & H) (I & J) (F & C)

$$Cost(A) = Cost(E \& G \& H) = 5 + 1 + 3 + 0 + 0 = 9$$

Using Heuristics



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

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

Admissible heuristic function

$$h(n) \leq h^*(n) = Cost(n) \leftarrow \text{Cost of the cheapest solution of } n$$

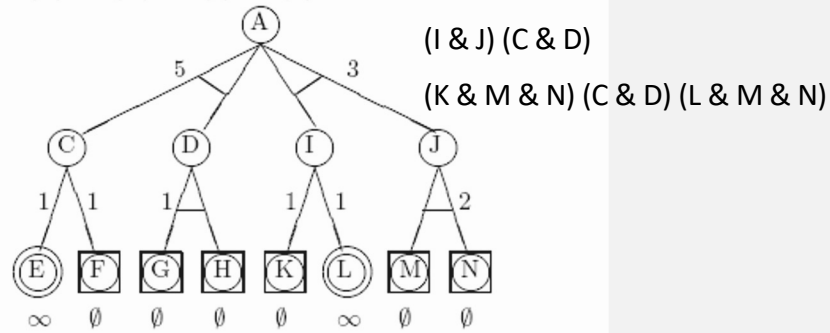
AO* - Searching AND-OR graphs

Example:

$$h(C) = h(D) = h(I) = h(J) = 1 \quad (A)$$

(I & J) (C & D)

(K & M & N) (C & D) (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 and the usual assumptions (finite branching factor etc) hold
- AO*, like A* is also optimal among the class of heuristic search algorithms that use an additive cost / evaluation function



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

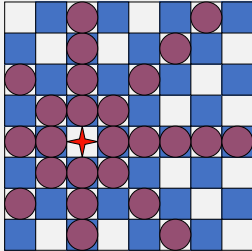
vhonavar@psu.edu
<http://faculty.ist.psu.edu/vhonavar>
<http://ailab.ist.psu.edu>

Constraint Satisfaction

Constraint satisfaction

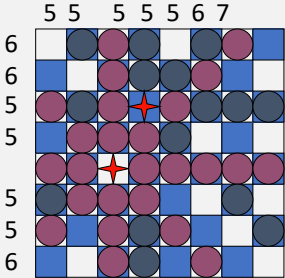
- Search techniques often consider choices in an arbitrary order
- In many problems, the same states can be reached independent of the order in which choices are made (“commutative” actions)
- Can we solve such problems more efficiently by picking the order appropriately? Can we even avoid making any choice?

Constraint Propagation



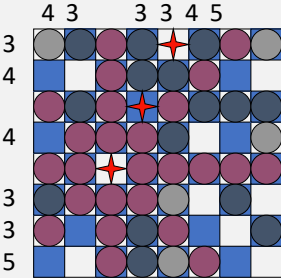
- Place a queen in a square
- Remove the attacked squares from future consideration

Constraint Propagation



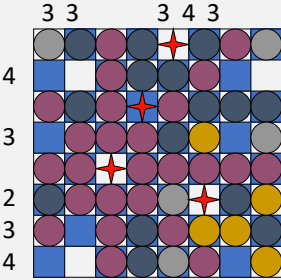
- Count the number of non-attacked squares in each row and column
- Place a queen in a row or column with minimum number
- Remove the attacked squares from future consideration

Constraint Propagation

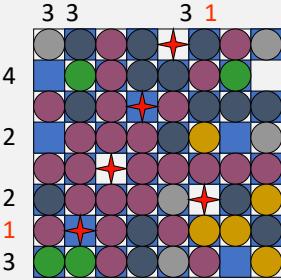


- Repeat

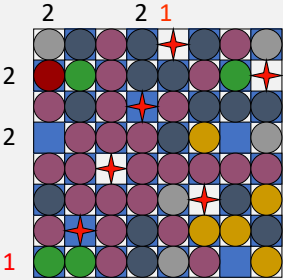
Constraint Propagation



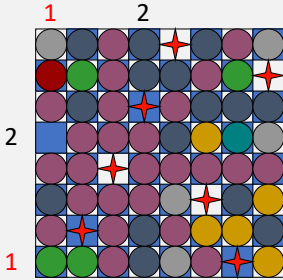
Constraint Propagation



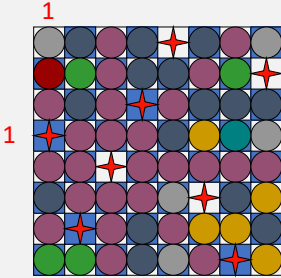
Constraint Propagation



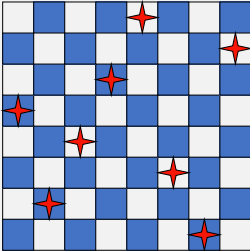
Constraint Propagation



Constraint Propagation



Constraint Propagation



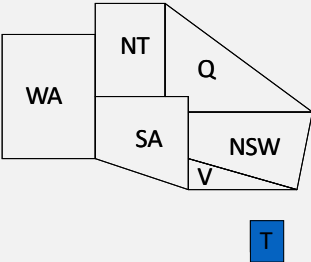
What do we need?

- More than just a state transition function and a goal test
 - We also need:
 - A means to **propagate the constraints** imposed by one queen's position on the positions of the other queens
 - An early **failure test**
- **Explicit representation of constraints**
- **Constraint propagation algorithms**

Constraint Satisfaction Problem (CSP)

- Set of **variables** $\{X_1, X_2, \dots, X_n\}$
- Each variable X_i has a **domain** D_i of possible values. Usually, D_i is finite
- Set of **constraints** $\{C_1, C_2, \dots, C_p\}$
- Each constraint relates a subset of variables by specifying the valid combinations of their values
- **Goal: Assign a value to every variable such that all constraints are satisfied**

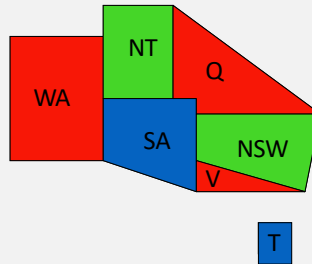
Map Coloring



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

- Use 3 colors – red, green, blue to color the map
- such that no two adjacent states have the same color

Map Coloring



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

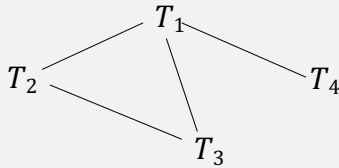
- 7 variables {WA,NT,SA,Q,NSW,V,T}
- Each variable has the same domain:
{red, green, blue}
- No two adjacent variables have the same value:

$WA \neq NT$, $WA \neq SA$, $NT \neq SA$, $NT \neq Q$, $SA \neq Q$,
 $SA \neq NSW$, $SA \neq V$, $Q \neq NSW$, $NSW \neq V$

8-Queens Problem

- 8 variables $X_1 \cdots X_8$
- Each variable takes one of 8 values: 1, 2, ..., 8
- Constraints are of the forms:
 - $X_i = k \rightarrow X_j \neq k$ for all $j = 1$ to $8, j \neq i$
 - Similar constraints for diagonals

Task Scheduling



Four tasks $T_1, T_2, T_3,$ and T_4 are related by time constraints:

- T_1 must be done during T_3
 - T_2 must be achieved before T_1 starts
 - T_2 must overlap with T_3
 - T_4 must start after T_1 is complete
- Are the constraints compatible?
 - What are the possible time relations between two tasks?
 - What if the tasks use resources in limited supply?

How to formulate this problem as a CSP?

CSP as a search problem


- Set of **variables** $\{X_1, X_2, \dots, X_n\}$
- Each variable X_i has a **domain** D_i of possible values. Usually, D_i is finite
- Set of **constraints** $\{C_1, C_2, \dots, C_p\}$
- Each constraint relates a subset of variables by specifying the valid combinations of their values
- **Goal: Assign a value to every variable such that all constraints are satisfied**

CSP as a Search Problem


- n variables X_1, \dots, 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 CSP

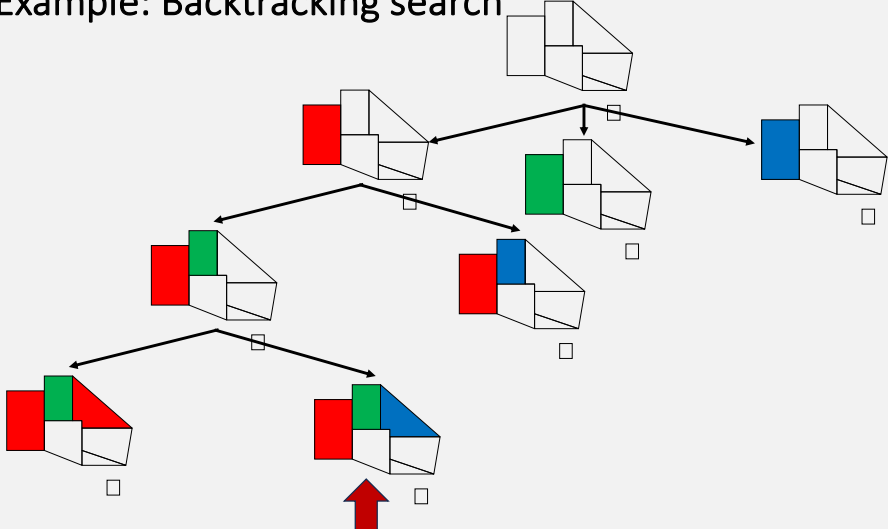
- The order in which variables are assigned values has no impact on the reachable complete valid assignments
 - One can expand a node by first selecting any unassigned variable and assign it a value from its domain without violating the constraints
 - Big reduction in branching factor to standard search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

 PennState
 Institute for Computational and Data Sciences


Center for Artificial Intelligence Foundations & Scientific Applications
 Artificial Intelligence Research Laboratory


 PennState
 Clinical and Translational Science Institute

Example: Backtracking search



Must backtrack because there is no legal color for SA

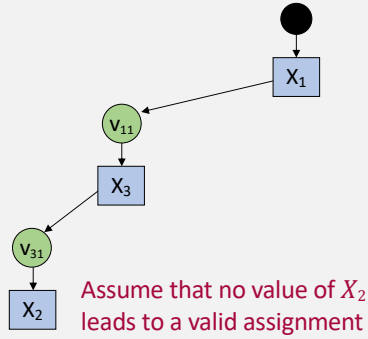
 PennState
 Center for Artificial Intelligence Foundations & Scientific Applications
 Artificial Intelligence Research Laboratory

 PennState
 Clinical and Translational Science Institute

AI 100 Fall 2024

Vasant G Honavar

Backtracking Search

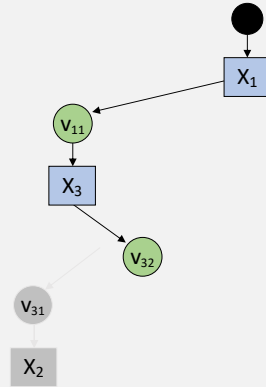


Now the search algorithm
backtracks to the previous variable
(X_3) and tries another value

Assignment = $\{(X_1, v_{11}), (X_3, v_{31})\}$

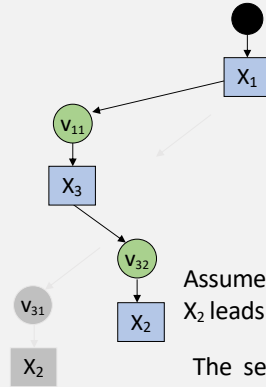


Backtracking Search



Assignment = $\{(X_1, v_{11}), (X_3, v_{32})\}$

Backtracking Search

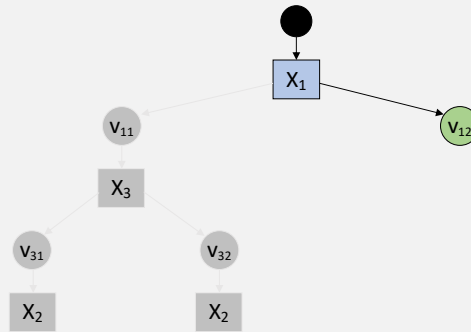


Assume again that no value of X_2 leads to a valid assignment

The search algorithm backtracks to the previous variable (X_3) and tries another value.
But assume that X_3 has only two possible values.
The algorithm backtracks to X_1

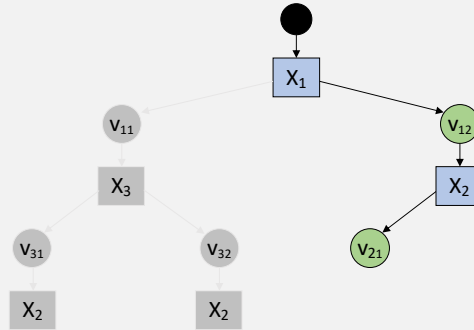
Assignment = $\{(X_1, v_{11}), (X_3, v_{32})\}$

Backtracking Search



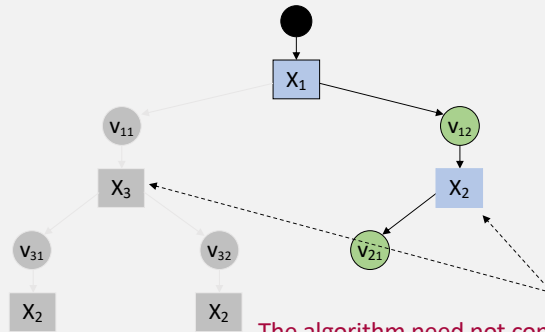
Assignment = $\{(X_1, v_{12})\}$

Backtracking Search



Assignment = $\{(X_1, v_{12}), (X_2, v_{21})\}$

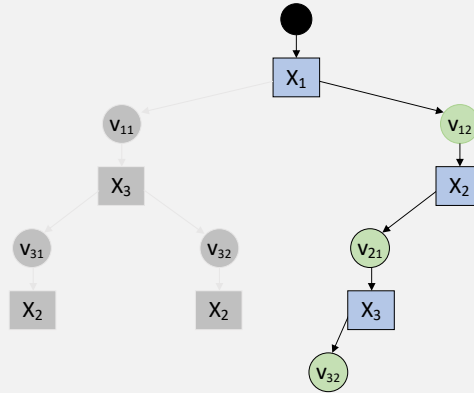
Backtracking Search



The algorithm need not consider
the variables in this sub-tree in the same
order in this sub-tree as in the other

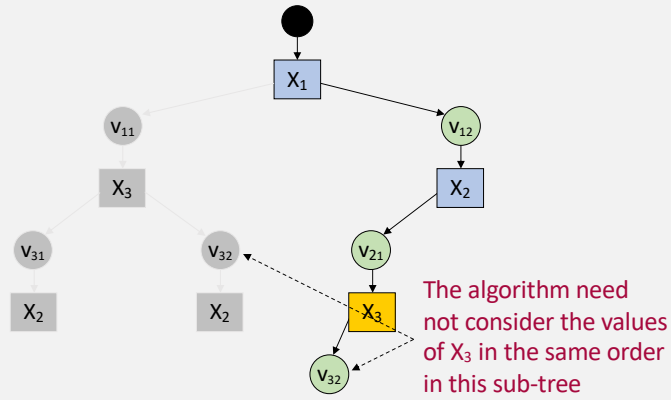
Assignment = $\{(X_1, v_{12}), (X_2, v_{21})\}$

Backtracking Search



Assignment = $\{(X_1, v_{12}), (X_2, v_{21}), (X_3, v_{32})\}$

Backtracking Search



Assignment = $\{(X_1, v_{12}), (X_2, v_{21}), (X_3, v_{32})\}$

PennState
Institute for Computational
and Data Sciences

Center for Artificial Intelligence Foundations & Scientific Applications
Artificial Intelligence Research Laboratory

PennState
Clinical and Translational
Science Institute

Backtracking Search

Assignment = $\{(X_1, v_{12}), (X_2, v_{21}), (X_3, v_{32})\}$

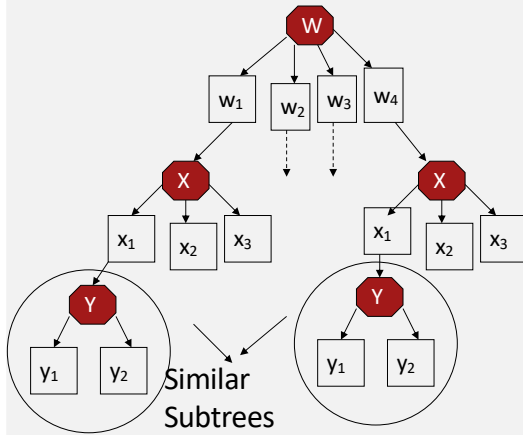
PennState
College of Engineering
Science and Technology

AI 100 Fall 2024

290
Vasant G Honavar

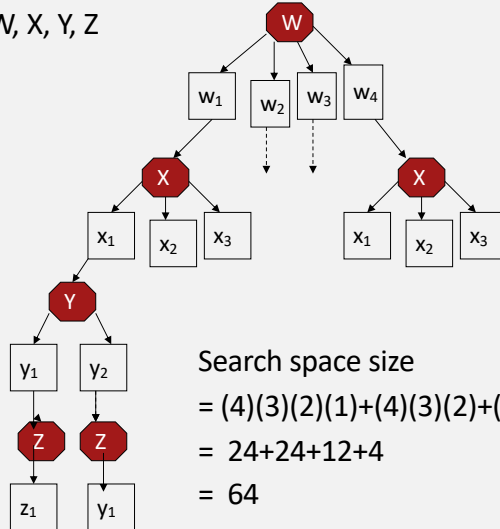
Subtrees have similar topologies

- Consider 4 variables W, X, Y, Z with domains of cardinality 4, 3, 2 and 1 respectively



Variable ordering and Search space size

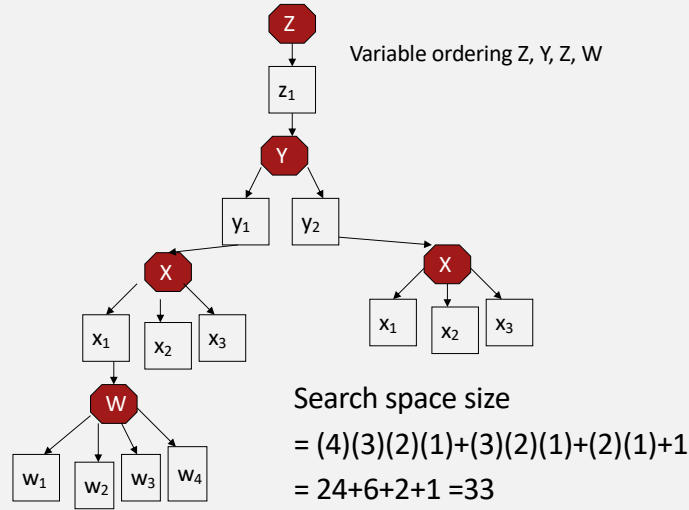
Variable ordering W, X, Y, Z



Search space size

$$\begin{aligned} &= (4)(3)(2)(1) + (4)(3)(2) + (4)(3) + (4) \\ &= 24 + 24 + 12 + 4 \\ &= 64 \end{aligned}$$

Variable ordering and Search space size



Backtracking search

- Standard backtracking fails to exploit special properties of CSP
 - Subtrees have similar topologies
 - Search space has minimal size under a certain ordering of variables (most constrained to least constrained)

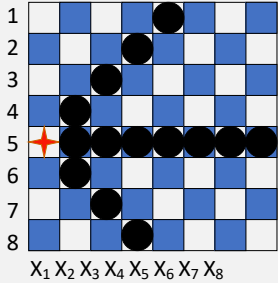
Can we improve the efficiency of backtracking?

- **Which variable X should be assigned a value next?**
The current assignment may not lead to any solution, but the algorithm still does know it. Selecting the right variable to which to assign a value may help discover the contradiction more quickly
- **In which order should X 's values be assigned?**
The current assignment may be part of a solution. Selecting the right value to assign to X may help discover this solution more quickly

We will return to these questions shortly

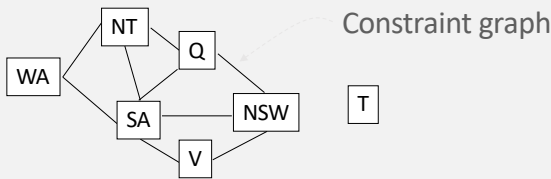
Propagating constraints: Forward checking

A simple constraint-propagation technique:



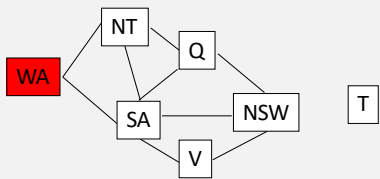
Assigning the value 5 to X_1 leads to removing values from the domains of X_2, X_3, \dots, X_8

Forward Checking in Map Coloring



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

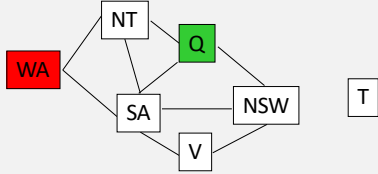
Forward Checking in Map Coloring



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

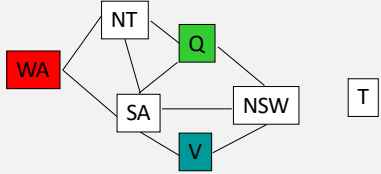
Forward checking removes the value Red of NT and of SA

Forward Checking in Map Coloring



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

Forward Checking in Map Coloring



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

Forward Checking in Map Coloring

Empty set: the current assignment
 $\{(WA \leftarrow R), (Q \leftarrow G), (V \leftarrow B)\}$
 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	R	B	B	RGB

Forward Checking (General Form)

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

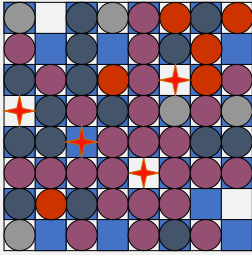
- Which variable X_i should be assigned a value next?
 - Most-constrained-variable heuristic
 - Most-constraining-variable heuristic
- In which order should its values be assigned?
 - Least-constraining-value heuristic

Keep in mind that **all** variables must eventually get a value, while only **one** value from a domain must be assigned to each variable

Most-Constrained-Variable Heuristic

- Which variable X_i should be assigned a value next?
 - Recall that ordering variables in the increasing order of their domain sizes minimizes the size of the search space
 - **Select the variable with the smallest remaining domain**

8-Queens



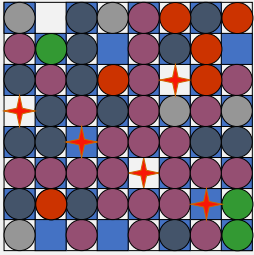
Forward checking

4 3 2 3 4

↑
New assignment

Numbers
of values for
each un-assigned
variable

8-Queens

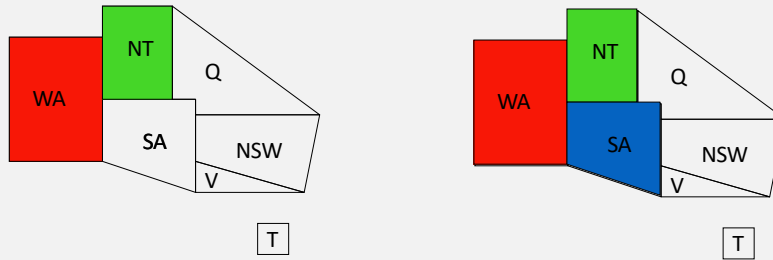


Forward checking

←----- New assignment

←----- New numbers
of values for
each un-assigned
variable

Map Coloring



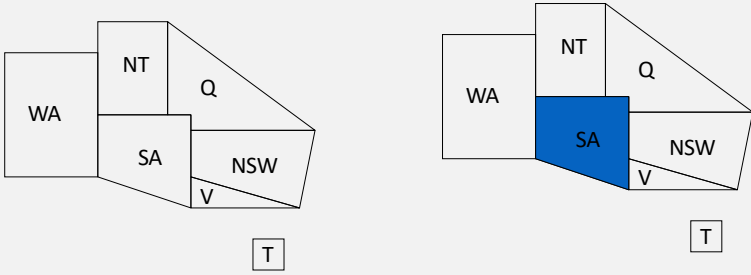
- SA's remaining domain has size 1 (value Blue remaining)
- Q's remaining domain has size 2
- NSW's, V's, and T's remaining domains have size 3

→ 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
- Select SA and assign a value to it (e.g., Blue)

Least-Constraining-Value Heuristic

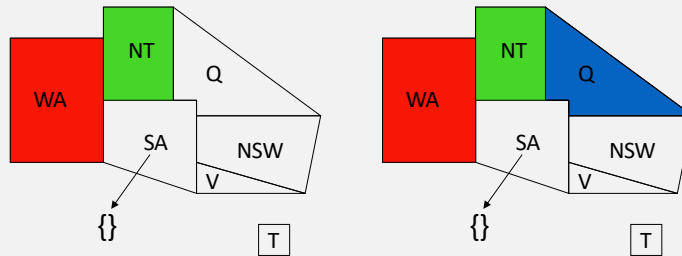
- In which order should X 's values be assigned?

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

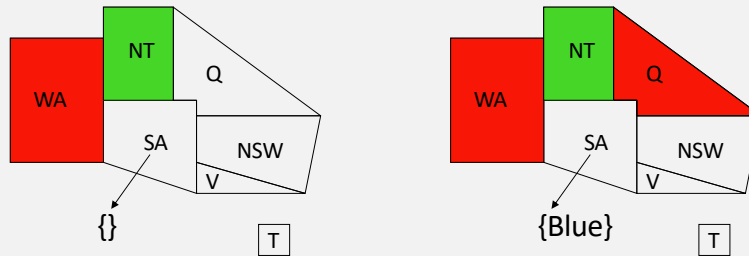
Note: Using this heuristic requires performing a forward-checking step for every value, not just for the selected value

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
- So, 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