CONSTRAINT SATISFACTION SEARCH

Fundamentals of Artificial Intelligence

Session 11

Pramod Sharma
pramod.sharma@prasami.com

2

Agenda

- Introduction
- Constraint Satisfaction Search
- Back Tracking Search

4/12/2024

pra-sâmi

3

What is CSP?

- Area of resource allocation: examination scheduling
  - Examinations are to be scheduled in a number of given time slots with a limited number of classrooms each examination needs a classroom
  - Different classrooms are of different capacity and an examination can only be scheduled in a classroom that has enough seats for students who are going to take that examination
  - Some students may take part in several examinations and these examinations cannot be scheduled in the same time slot

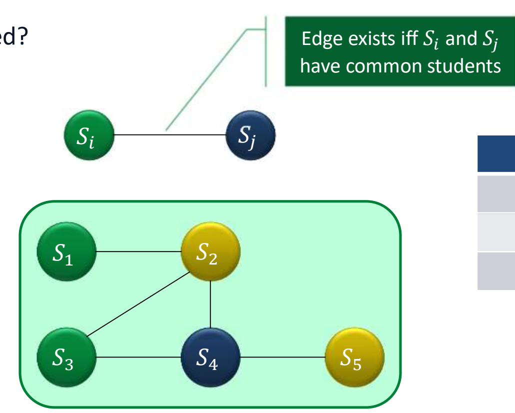
- To model this problem we can make a tuple of (V, D, C)
  - V: Variables – each examination
  - D: Domain – possible time slots and classrooms
  - C : Constraints – that certain examinations cannot be held together e.g. exams taken by same set of students

4

Scheduling of Exams

- How many exams slots do I need?

Student	Subject
X1	S1
X1	S2
X2	S2
X2	S3
X3	S2
X3	S4
X3	S5
⋮	⋮



Slots	Subjects
1	S1, S3
2	S2, S5
3	S4

- Similar exercise is executed when we are scheduling presentations during a conference
  - Customers are placed in a groups with their likely inclination towards a product or a service
  - Sessions are scheduled such that customers can attend sessions of their interest

5

What is CSP?

- Airport gate allocation
- Physical constraints
  - Certain jetways can only accommodate certain types of aircraft
- User preferences
  - Different airlines prefer to park in certain parts of an airport
- A solution to the CSP would be a solution to the airport gate allocation problem
- In all areas of industry and business resource allocation is a key factor to making a profit and a loss

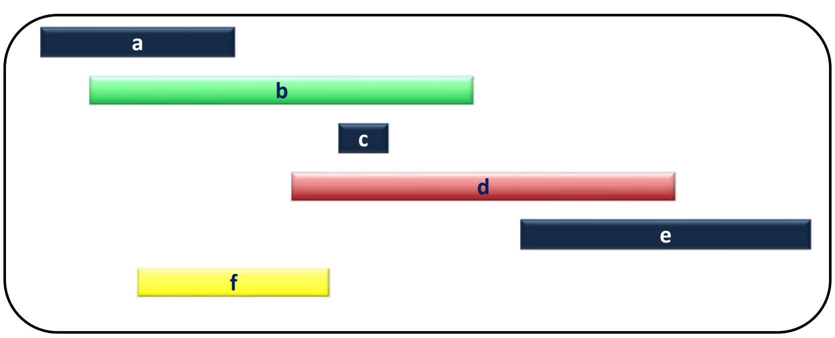
4/12/2024

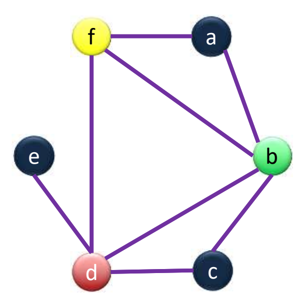
pra-sâmi

6

Register Allocation

- Register memory is the smallest and fastest memory in a computer.
- It is not a part of the main memory and is located in the CPU in the form of registers
- Compiler would like to keep as many local variable in register inside CPU rather than memory
- A register temporarily holds frequently used data, instructions, and memory address that are to be used by CPU





4/12/2024

pra-sâmi

7

Berth Minimization Problem

- Indian Railways is planning a new train between Howrah and Chennai
- There will be reservation Quota between Stations
- Problem:
  - How many berths needed to satisfy berth quotas?

	KGP	BBS	VZM	VSKP	BZA	MAS	Sum
HWH	10	45	70	...	...	200	325
KGP		20				50	385
BBS			50	40		50	460
VZM				30	30	50	450
VSKP					40	40	460
BZA						20	410

- No of person boarding from “HWH” and alighting at KGP = 10
- No of person boarding from “HWH” and alighting at MAS = 200

pra-sâmi

8

CSPs in the Real World

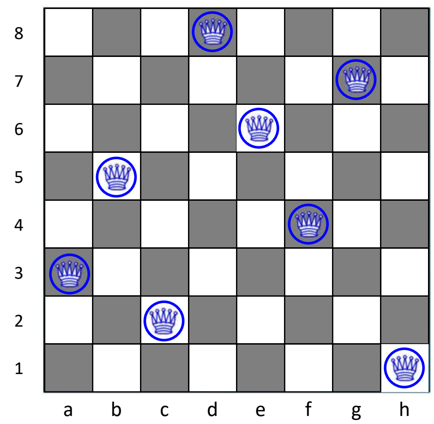
- Scheduling space shuttle repair
- Airport gate assignments
- Transportation Planning
- Supply-chain management
- Computer configuration
- Diagnosis
- UI optimization
- Sudoku
- Etc...

pra-sâmi

9

Example: N-Queens

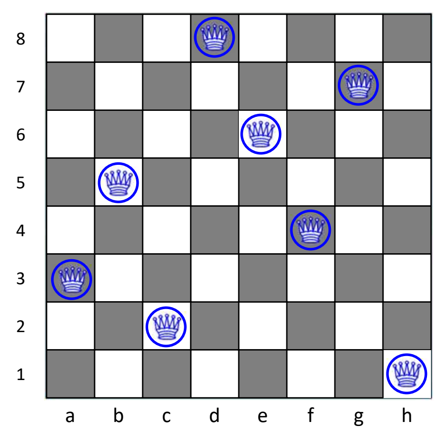
- Place N Queens on an N X N chess board so that no Queen can attack any other Queen



10

Example: N-Queens

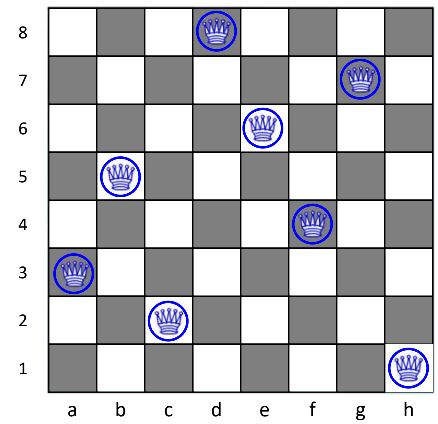
- Problem formulation:
  - N variables (N queens)
  - N^2 values for each variable representing the positions on the chessboard
- This representation has $(N^2)^N$ states (different possible assignments in the search space)
  - For 8-Queens: 64^8 = 281,474,976,710,656
- Is there a better way to represent N^2 the N-queens problem?
  - We know we cannot place two queens in a single row → we can exploit this fact in the choice of the CSP representation already



11

Example: N-Queens

- Lets use one of the constraints:
  - N variables Q_i, one per row
  - Value of Q_i is the row the Queen in column i is placed; possible values {1, …, N}
- This representation has $(N)^N$ states:
  - For 8-Queens: $(8)^8$ = 16,777,216
- Looks way better now!
- The choice of a representation can decided whether or not we can solve a problem!

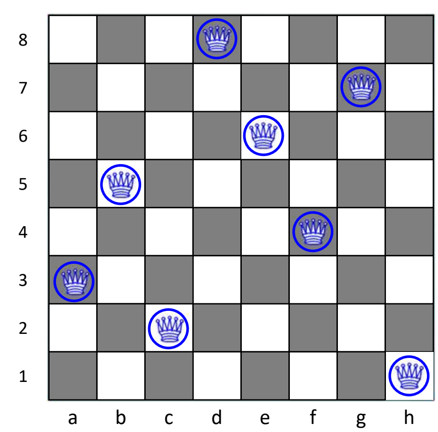


pra-sâmi

12

Example: N-Queens

- Q_1 = 3, Q_2 = 5, Q_3 = 2, Q_4 = 8, Q_5 = 6, Q_6 = 4, Q_7 = 7, Q_8 = 1
- Constraints:
  - Can't put two Queens in same column $Q_i \neq Q_j$ for all i ≠ j
  - Diagonal constraints $|Q_i - Q_j| \neq$ i-j
  - i.e., the difference in the values assigned to Q_i and Q_j can't be equal to the difference between i and j.



pra-sâmi

13

Example: Sudoku

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Sudoku becomes easy (under 0.1s)

14

Sudoku

- States : partial assignment of digits
  - Stated Problem : Not minimal
  - Assume all squares are filled except lower – right 3 x 3 sub-block
  - What actually needed to be in that state?
- Put a digit in blank square
  - Problem :
    - Redundancy : <*num-initial-blanks*>! Ways to reaching goal
    - Rigidity : What order? Is it fixed? Not so easy to decide
    - Way too many
- Cost: ∞ if new digit violates a constraint, else 0
- We can keep fighting with states… but can we have a general systematic approach

15

Variable Based Models

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

- End goal is an assignment of digits to squares
- Sequence of action is just a mean to an end
- May be define model in terms of assignment and let algorithms define the state-actions pair
- Overall the order of application of actions has no effect on outcome

16

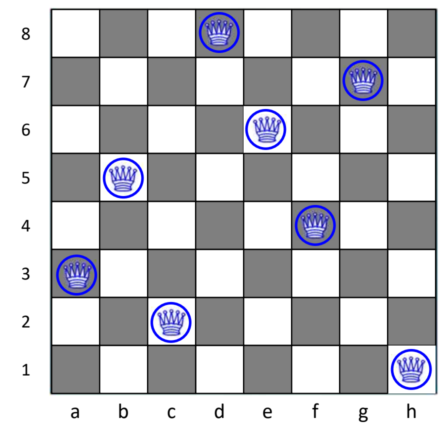
Constraint Satisfaction Search

- In a Standard search problem:
  - State is a **black box** – any data structure that supports successor function and goal test
- Constraint satisfaction search:
  - “State” is defined by variables X_i with values from domain D_i
  - “Goal test” is a set of constraints specifying allowable combinations of values for subsets of variables
- It’s a tuple of (V, D, C)
  - V: Variables
  - D: Domain
  - C : Constraints
- Example : Sudoku
  - V = 9 x 9 squares
  - D = 1 to 9 digits
  - Constraints : no repeat digits in any row, column or square blocks!

17

What is CSP?

- The N-queens problem can be modeled as a CSP also
  - The problem is to place N queens on N distinct squares in an N x N chess board
    - No two queens should be under attack
    - Two queens threaten each other if and only if they are on the same row column or diagonal
- Variables : each of the rows(rank) a variable $\{V_1, V_2, V_3, V_4, V_5, V_6, V_7$ and $V_8\}$
- Domains D_i : $\{D_1, D_2, D_3, D_4, D_5, D_6, D_7$ and $D_8\}$
- Constraints: For each constraint R_{ij}
    - No two queens on the same row.
    - No two queens on the same column $V_i \neq V_j$
    - No two queens on the same diagonal: |i-j| ≠ $|V_i - V_j|$



4/12/2024

pra-sâmi

18

Example: Sudoku

- Variables: $V_{11}, V_{12}, \ldots, V_{21}, V_{22}, \ldots, V_{91}, \ldots, V_{99}$
- Domains:
  - Domain[V_{ij}] = {1-9} for empty cells
  - Domain[V_{ij}] = {k} a fixed value k for filled cells.

- Row constraints:
  - CR1($V_{11}, V_{12}, V_{13}, \ldots, V_{19}$)
  - CR2($V_{21}, V_{22}, V_{23}, \ldots, V_{29}$)
  - ....
  - CR9($V_{91}, V_{92}, \ldots, V_{99}$)
- Column Constraints:
  - CC1($V_{11}, V_{21}, V_{31}, \ldots, V_{91}$)
  - CC2($V_{21}, V_{22}, V_{13}, \ldots, V_{92}$)
  - ....
  - CC9($V_{19}, V_{29}, \ldots, V_{99}$)
- Sub-Square Constraints:
  - CSS1($V_{11}, V_{12}, V_{13}, V_{21}, V_{22}, V_{23}, V_{31}, V_{32}, V_{33}$)
  - CSS2($V_{14}, V_{15}, V_{16}, \ldots, V_{34}, V_{35}, V_{36}$)

4/12/2024

pra-sâmi

19

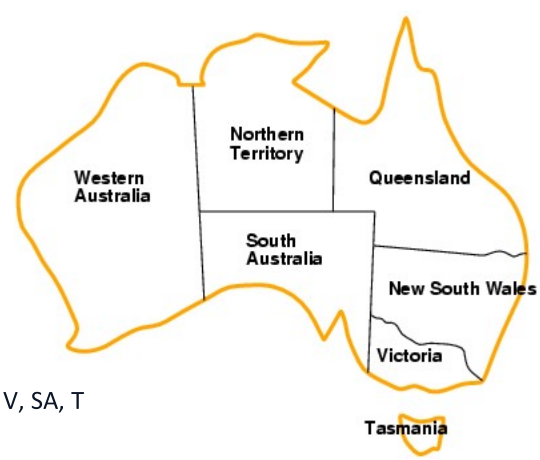
Example: Sudoku

- Each of these constraints is over 9 variables, and they are all the same constraint:
  - Any assignment to these 9 variables such that each variable has a unique value satisfies the constraint.
  - Any assignment where two or more variables have the same value falsifies the constraint.
- Special kind of constraints called ALL-DIFF constraints.
  - An ALL-DIFF constraint over k variables can be equivalently represented by (k choose 2) “not-equal constraints” (NEQ) over each pair of these variables.
  - e.g. CSS1($V_{11}, V_{12}, V_{13}, V_{21}, V_{22}, V_{23}, V_{31}, V_{32}, V_{33}$) = NEQ($V_{11}, V_{12}$), NEQ($V_{11}, V_{13}$), NEQ($V_{11}, V_{21}$) …, NEQ($V_{32}, V_{33}$)
  - Remember: all higher-order constraints can be converted into a set of binary constraints
- Thus Sudoku has 3 x 9 ALL-DIFF constraints, one over each set of variables in the same row, one over each set of variables in the same column, and one over each set of variables in the same sub-square.

pra-sâmi

20

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, or (WA,NT) in { (red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green) }

pra-sâmi

21

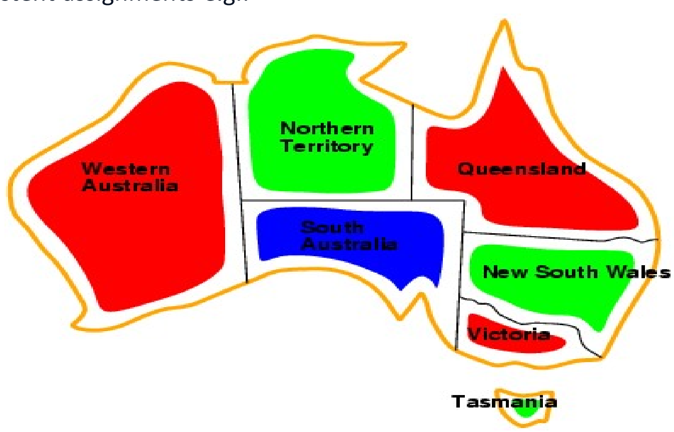
Example: Map-Coloring

- CSP solvers can be faster than state-space searchers
  - The CSP solver can quickly eliminate large swatches of the search space.
- Once we have chosen {SA = blue} in the Australia problem
  - None of the five neighboring variables can take on the value blue.
- Without constraint propagation, a search procedure would have to consider 3^5 = 243 assignments for the five neighboring variables
- With constraint propagation, we have only 2^5 = 32 assignments

22

Example: Map-Coloring

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



23

Example: Cryptarithmetic

- An arithmetic problem which is represented in letters
- Involves the decoding of digit represented by a character
- It is in the form of some arithmetic equation where digits are distinctly represented by some characters.
- The problem requires finding of the digit represented by each character
- Assign a decimal digit to each of the letters in such a way that the answer to the problem is correct
- If the same letter occurs more than once, it must be assigned the same digit each time
- No two different letters may be assigned the same digit

```
 SEND
+MORE
MONEY
```

- Variables: S E N D M O R Y X1 X2 X3 X4
- Domains: {0,1,2,3,4,5,6,7,8,9}
- Constraints: Alldiff (S, E, N, D, M, O, R, Y)

- D + E = Y + 10 * X1
- X1 + N + R = E + 10 * X2
- X2 + E + O = N + 10 * X3
- X3 + S + M = O + 10 * X4,
- S ≠ 0,
- M ≠ 0

pra-sâmi

24

Example: Cryptarithmetic

```
  S E N D
+ M O R E
M O N E Y
```

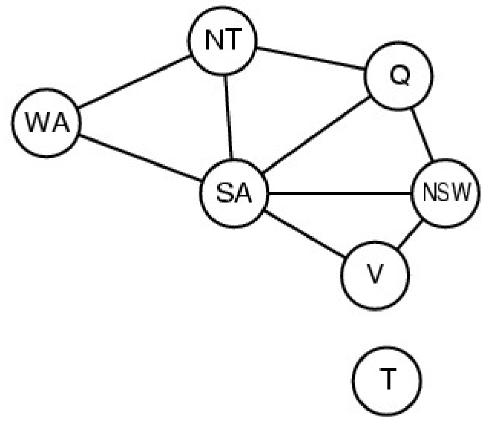
S	-	1	2	3	4	5	6	7	8	9
E	-	1	2	3	4	5	6	7	8	9
N	-	1	2	3	4	5	6	7	8	9
D	-	1	2	3	4	5	6	7	8	9
M	-	1	2	3	4	5	6	7	8	9
O	-	1	2	3	4	5	6	7	8	9
R	-	1	2	3	4	5	6	7	8	9
Y	-	1	2	3	4	5	6	7	8	9

pra-sâmi

26

Constraint Graph

- Easier to Visualize on a Constraint Graph
- Binary CSP: each constraint relates two variables
- Constraint graph: nodes are variables arcs are constraints

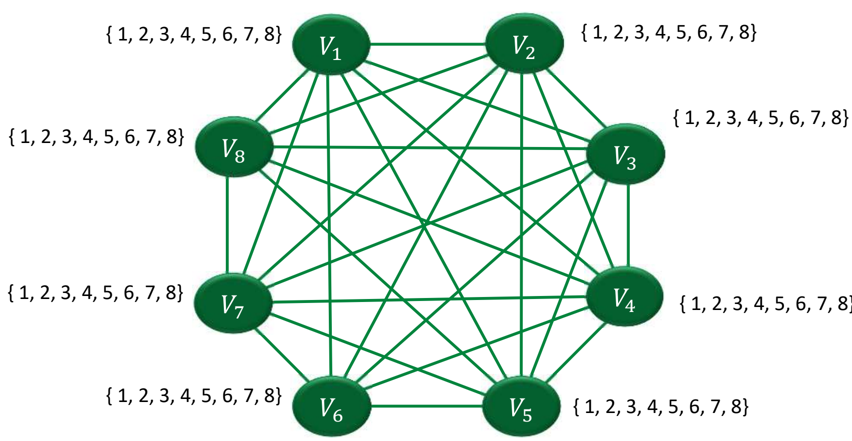


4/12/2024

pra-sâmi

27

Constraint Graph – n-Queens



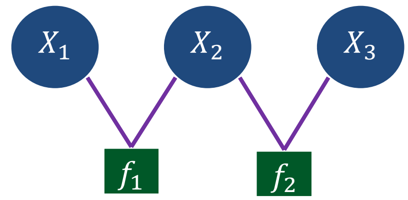
4/12/2024

pra-sâmi

28

Factored Graphs

- A factored graph is graphical way of representing a CSP
- Variables: Nodes are round in shape
  - X_1 , X_2 , X_3 … X_n
- Factor (Constraints): Nodes are square in shape
  - f_1 , f_2 , f_3 … f_n
- Edges represent dependencies between variables and constraints
  - f_1(X) = $[X_1 = X_2]$
  - f_2 (X) = [$X_2 \neq X_3$]

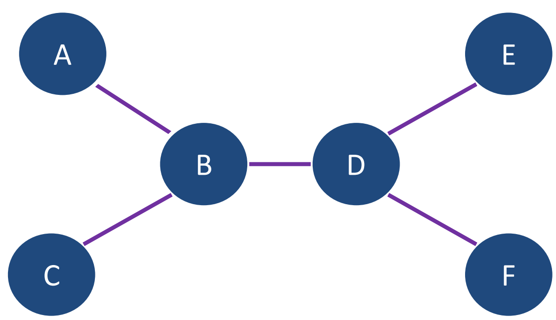


4/12/2024

pra-sâmi

29

Trees are Easy



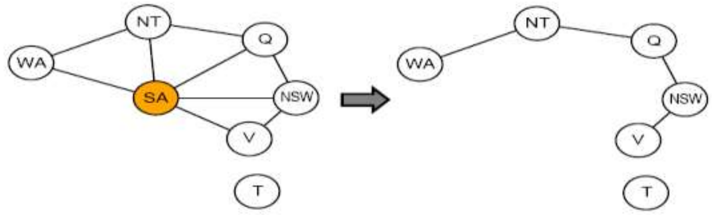
- Theorem:
  - If a constraint graph has no loops then the CSP can be solved in o(n * d^2) time
  - Linear in the number of variables!
  - Compare difference with general CSP, where worst case is o(d^n)

4/12/2024

pra-sâmi

30

Cut Set



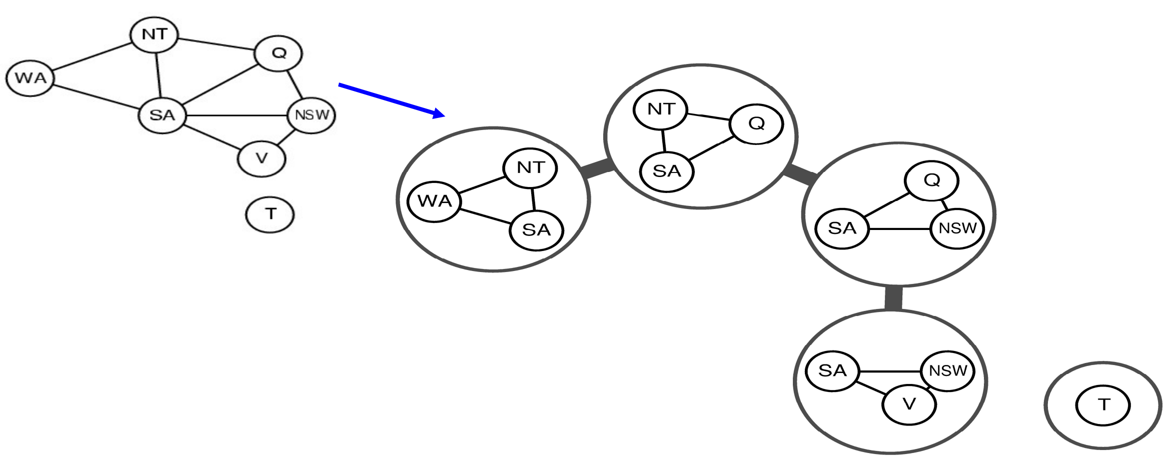
- Cutset Conditioning: initiate a set of variable such that remaining constraint graph is a tree
- Cutset size ➔ runtime O($d^c.(n-c).d^2$)

4/12/2024

pra-sâmi

31

Tree Decomposition



4/12/2024

pra-sâmi

32

Backtracking Search

pra-sâmi

33

Backtracking Search

- Variable assignments are commutative,
  - i.e., [WA = red then NT = green] same as [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 is the basic uninformed algorithm for CSPs.
- Can solve n-queens for n ≈ 25.

pra-sâmi

34

Backtracking

- Suppose you have to make a series of decisions, among various choices, where
  - You don't have enough information to know what to choose
  - Each decision leads to a new set of choices
  - Some sequence of choices (possibly more than one) may be a solution to your problem
- Backtracking is a methodical way of trying out various sequences of decisions, until you find the correct one that "works"
- Backtracking is used to solve problems in which a sequence of objects is chosen from a specified set so that the sequence satisfies some criterion
- It is the procedure whereby, after determining that a node can lead to nothing but dead nodes, we go back ("backtrack") to the node's parent and proceed with the search on the next child.

4/12/2024

pra-sâmi

35

Backtrack Algorithm

- Backtracking is a modified depth-first search of a tree
- Based on depth-first recursive search
- Approach
  - Tests whether solution has been found
  - If found solution, return it
  - Else for each choice that can be made
    - Make that choice
    - Recursive
    - If recursion returns a solution, return it
  - If no choices remain, return failure

4/12/2024

pra-sâmi

36

Improving Backtracking

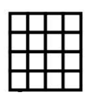
- Search pruning will help us to reduce the search space and hence get a solution faster.

- The idea is to avoid those paths that may not lead to a solutions as early as possible by finding contradictions so that we can backtrack immediately without the need to build a hopeless solution vector.

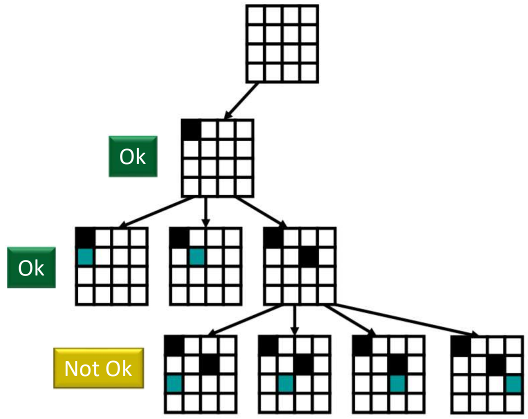
- Backtracking examples
  - Solving a maze
  - Coloring a map
  - Solving a puzzle
  - N queens problem etc.,

38

Backtracking Example: N-Queens

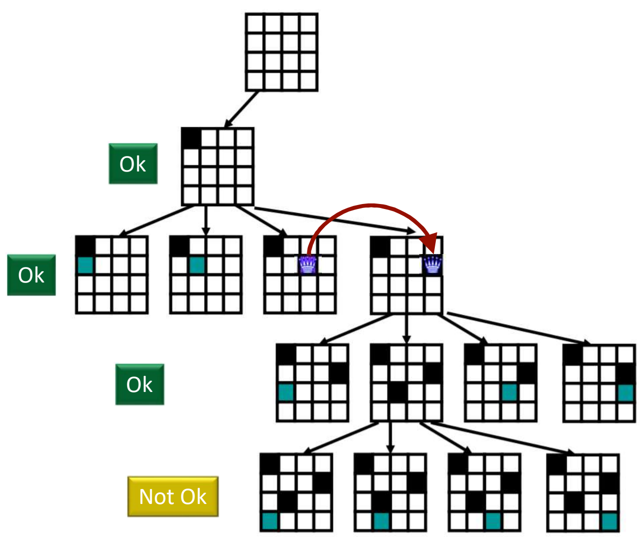


Backtracking Example: N-Queens



Let's go back to last known ok node!

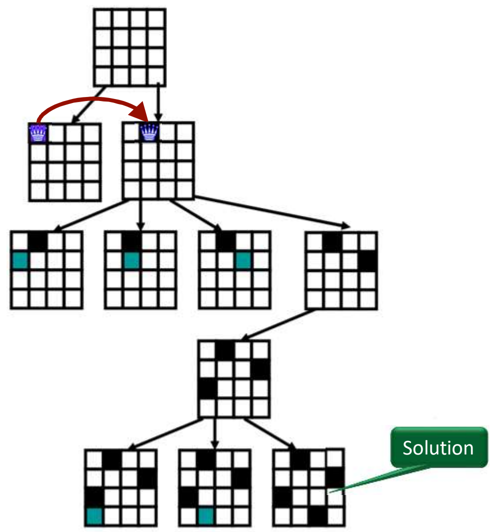
Backtracking Example: N-Queens



Let's go back one more level!

41

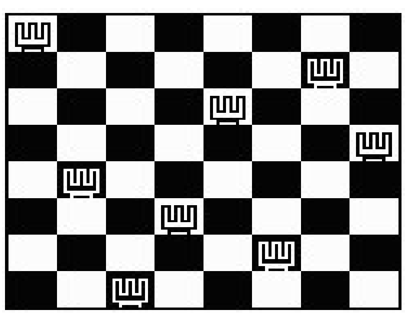
Backtracking Example: N-Queens



42

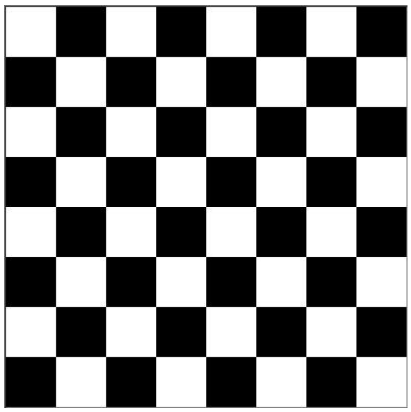
Backtracking - 8 Queens Problem

- Solved by trying to place the first queen, then the second queen so that it cannot attack the first, and then the third so that it is not conflicting with previously placed queens.



43

Backtracking - 8 Queens Problem

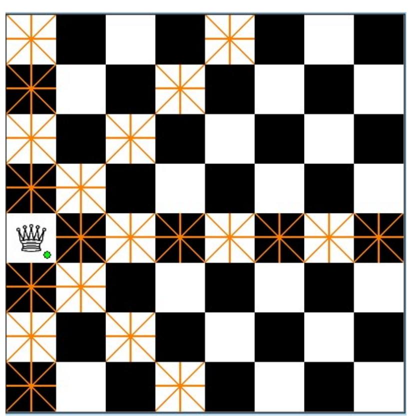


- It is an empty 8 x 8 chess board. We have to place the queens in this board.

pra-sâmi

44

Backtracking - 8 Queens Problem

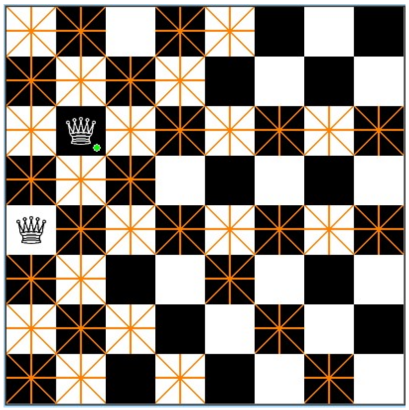


- We have placed the first queen on the chess board

pra-sâmi

45

Backtracking - 8 Queens Problem

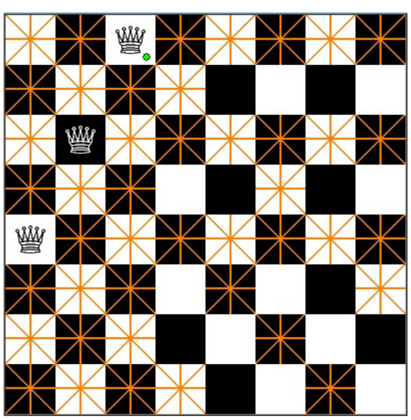


- Then we have placed the second queen on the board.
- The darken place should not have the queens because they are horizontal, vertical, diagonal to the placed queens.

pra-sâmi

46

Backtracking EXAMPLE—8 Queens Problem(cont…)

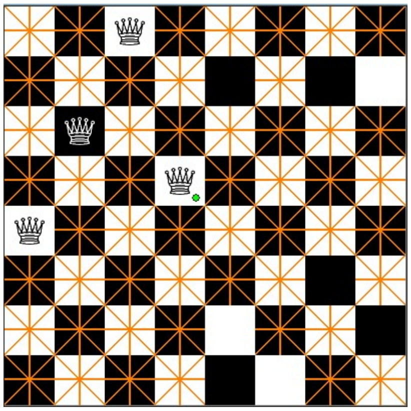


- We have placed the third queen on board.

pra-sâmi

47

Backtracking EXAMPLE—8 Queens Problem(cont…)



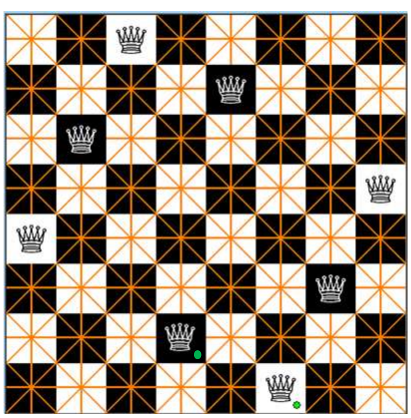
- We have placed the 4th queen on the board.
- We have placed that in the wrong spot, so we backtrack and change the place of that one.

4/12/2024

pra-sâmi

48

Backtracking - 8 Queens Problem



- In this way, we have to continue the process until our goal is reached
  - Must place 8 queens on the board
- Backtracking provides the hope to solve some problem instances of nontrivial sizes by pruning non-promising branches of the state-space tree.
- The success of backtracking varies from problem to problem and from instance to instance.
- Backtracking possibly generates all possible candidates in an exponentially growing state-space tree.

4/12/2024

pra-sâmi

49

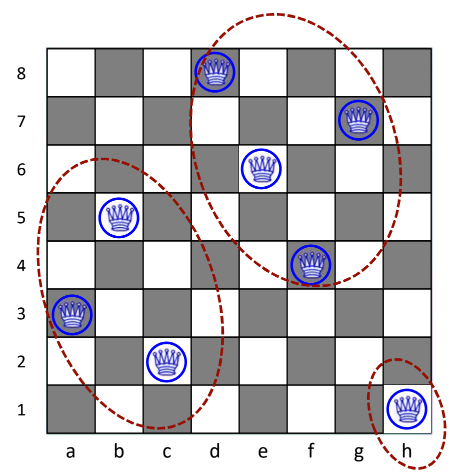
Backtracking search

```
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
   return failure
```


50

Parallelizing Backtrack Algorithm

- First, we have to parallelize the root node of the algorithm.
- Then the sub nodes and the child nodes should be parallelized independently using the other processors.
- For example, if we take the 8 queens problem then it can be easily implemented in parallel.
- The solutions to the n-queens problem can be generated in parallel by using the master-worker technique.



51

Parallelizing Backtrack Algorithm

- The manager generates the upper portion of the search tree by generating those nodes of fixed depth d, for some d.
- The manager dynamically passes each of these sequences to an idle worker, who in turn continues to search for sequences with n-queens property that contain the fixed subsequence of length d.
- The master-worker technique is particularly well-suited for implementation with MPI (Message Passing Interface)
- Parallelizing the backtrack algorithm will gives us a good speedup and efficiency when compared to the normal algorithm
- The speedup and the efficiency will gets drastically increased when it is done in the parallel.

52

Backtracking example

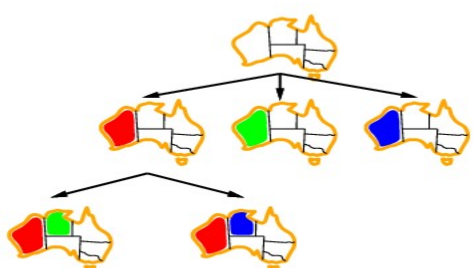
53

Backtracking example

pra-sâmi

54

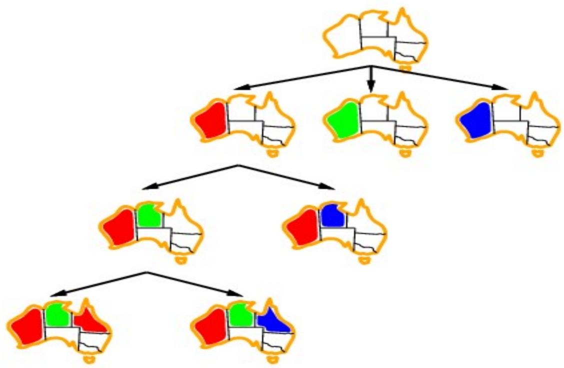
Backtracking example



pra-sâmi

55

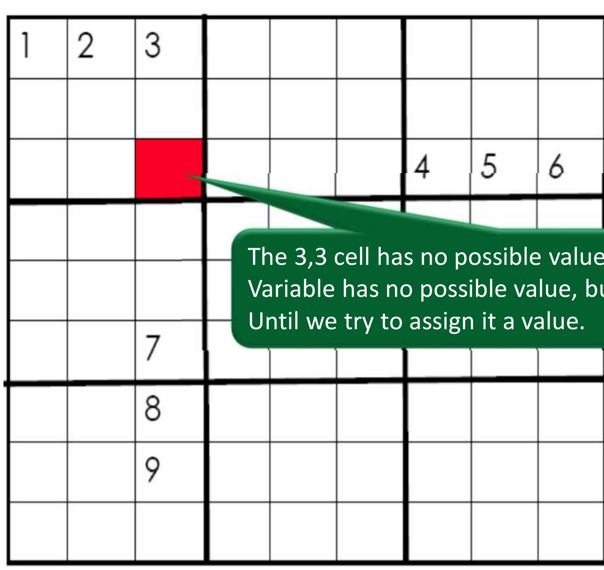
Backtracking example



pra-sâmi

56

Problems with Plain Backtracking



The 3,3 cell has no possible value.
Variable has no possible value, but we don't detect this.
Until we try to assign it a value.

pra-sâmi

57
Improved Search

- General-purpose methods can give huge gains in speed:
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect inevitable failure early?

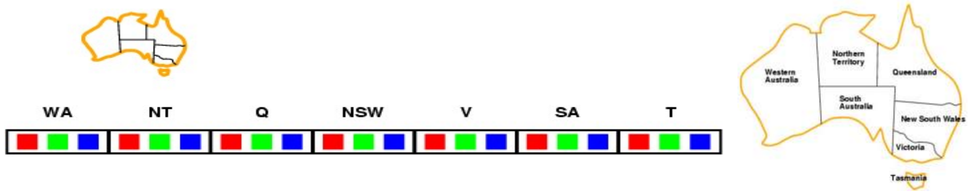
58
Constraint Propagation

- Constraint propagation refers to the technique of “looking ahead” at the yet unassigned variables in the search
- Try to detect obvious failures:
  - “Obvious” means things we can test/detect efficiently
- Even if we don’t detect an obvious failure we might be able to eliminate some possible part of the future search
- Propagation has to be applied during the search; potentially at every node of the search tree
- Propagation itself is an inference step which needs some resources (in particular time)
  - If propagation is slow, this can slow the search down to the point where using propagation actually slows search down!
  - There is always a tradeoff between searching fewer nodes in the search, and having a higher nodes/second processing rate
- We will look at two main types of propagation!

59

Forward Checking

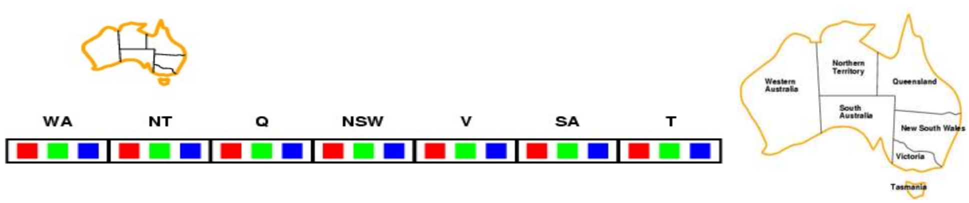
- Forward checking is an extension of backtracking search that employs a “modest” amount of propagation (look ahead)
- When a variable is instantiated we check all constraints that have only one un-instantiated variable remaining.
- For that un-instantiated variable, we check all of its values, pruning those values that violate the constraint.



60

Forward Checking

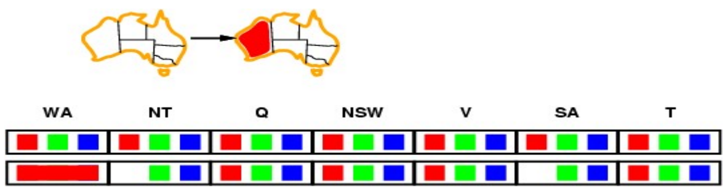
- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values.



61

Forward Checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values.



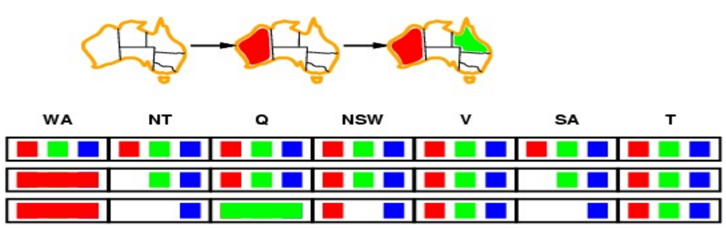
4/12/2024

pra-sâmi

62

Forward Checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values



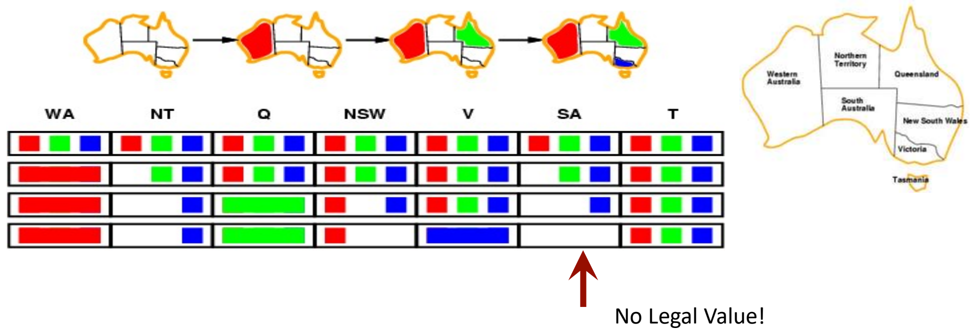
4/12/2024

pra-sâmi

63

Forward Checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

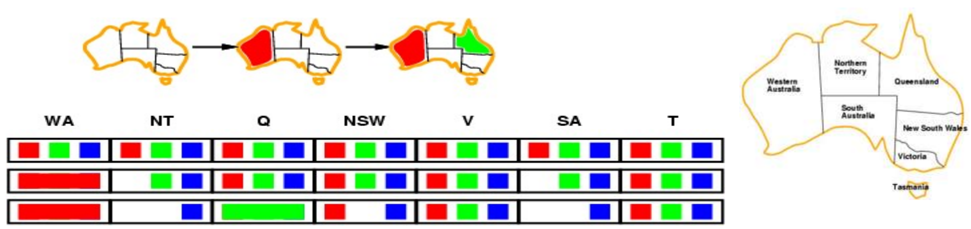


pra-sâmi

64

Forward Checking

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:



- NT and SA cannot both be blue!
- Constraint propagation repeatedly enforces constraints locally.

pra-sâmi

65

FC: Minimum Remaining Values Heuristics (MRV)

- FC also gives us for free a very powerful heuristic to guide us which variables to try next:
  - Always branch on a variable with the smallest remaining values (smallest Current Domain).
  - If a variable has only one value left, that value is forced, so we should propagate its consequences immediately.
  - This heuristic tends to produce skinny trees at the top. This means that more variables can be instantiated with fewer nodes searched, and thus more constraint propagation/DWO failures occur with less work.
  - We can find an inconsistency such as in the Sudoku example much faster.

66

MRV Heuristic: Human Analogy

- What variables would you try first?

8	1	5	6					4
6				7	5		8	
				9				
9				4	1	7		
	4						2	
		6	2	3				8
				5				
	5		9	1				6
1					7	8	9	5

- Domain of each variable: {1, 2, …, 9}

67

MRV Heuristic: Human Analogy

- What variables would you try first?

8	1	5	6					4
6				7	5		8	
				9				
9				4	1	7		
	4						2	
		6	2	3				8
				5				
	5		9	1				6
1					7	8	9	5

- Domain of each variable: {1, 2, …, 9}

4/12/2024

pra-sâmi

68

MRV Heuristic: Human Analogy

- What variables would you try first?

8	1	5	6					4
6				7	5		8	
				9				
9				4	1	7		
	4						2	
		6	2	3				8
				5				
	5		9	1				6
1					7	8	9	5

- Most restricted variables! = MRV

- Domain of each variable: {1, 2, …, 9}

- $Cell_{1,5}$: impossible values:
  - Row: {1, 4, 5, 6, 8}
  - Column: {1, 3, 4, 5, 7, 9}
  - Subsquare: {5, 7, 9, 6} ➔ Domain = {2}

- $Cell_{9,5}$: impossible values:
  - Row: {1, 5, 7, 8, 9}
  - Column: {1, 3, 4, 5, 7, 9}
  - Sub-square: {1, 5, 7, 9} ➔ Domain = {2, 6}

  - After assigning value 2 to $Cell_{1,5}$: Domain = {6}

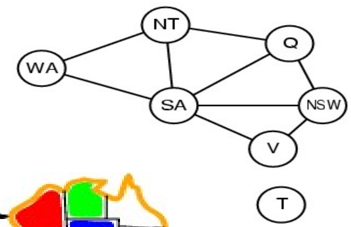
4/12/2024

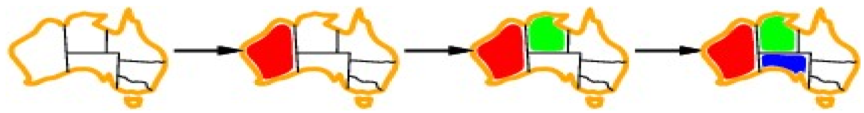
pra-sâmi

69

Most Constrained Variable

- Most constrained variable:
  - Choose the variable with the fewest legal values.



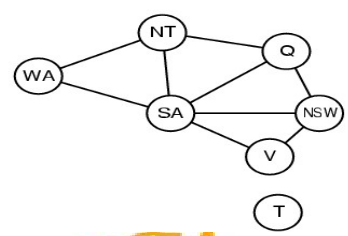


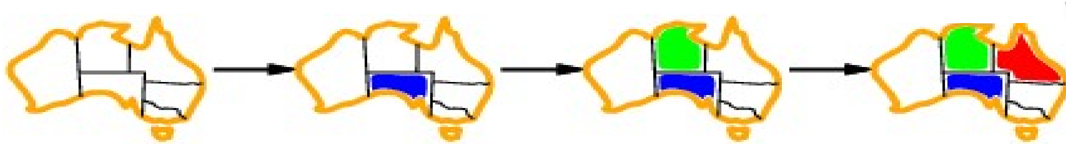
- a.k.a. minimum remaining values (MRV) heuristic.

70

Most Constraining Variable

- Tie-breaker among most constrained variables
- Most constraining variable:
  - choose the variable with the most constraints on remaining variables.

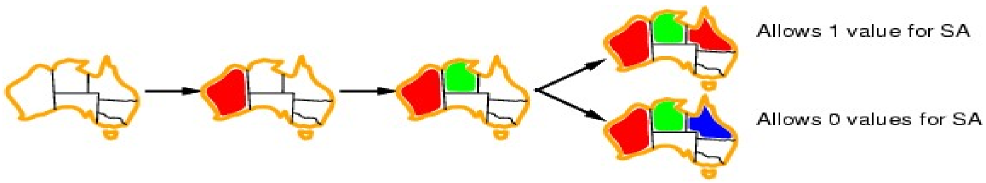




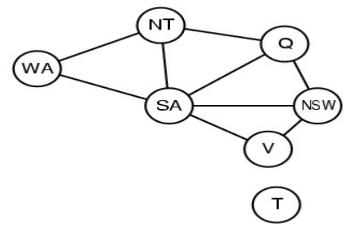
71

Least Constraining Value

- Given a variable, choose the least constraining value:
  - the one that rules out the fewest values in the remaining variables.



- Combining these heuristics makes 1000 queens feasible.



pra-sâmi

72

Empirically

- Forward Checking often is about 100 times faster than BackTracking
- Forward Checking with MRV (minimal remaining values) often 10000 times faster.
- But on some problems the speed up can be much greater
  - Converts problems that are not solvable to problems that are solvable.
- Other more powerful forms of consistency are commonly used in practice. Arc???

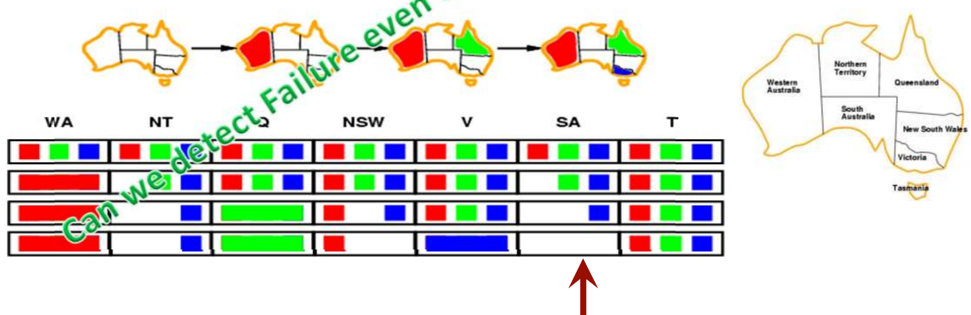
pra-sâmi

73

Forward Checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

Can we detect Failure even earlier than this?



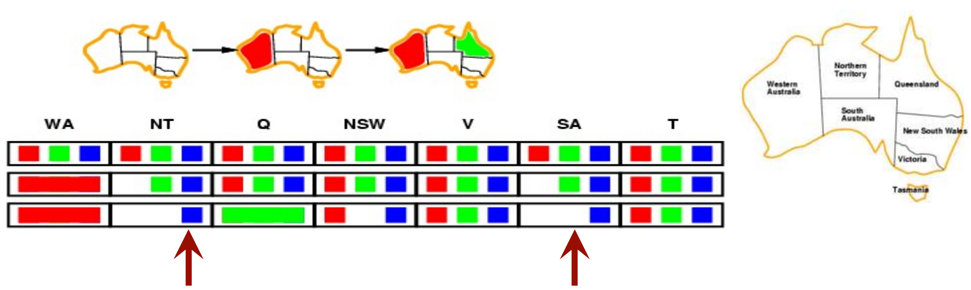
No Legal Value!

pra-sâmi

74

Forward Checking

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures
  - In fact, The failure should have been apparent at step 2 itself



- NT and SA cannot both be blue!

pra-sâmi

75

Constraint Propagation: Arc Consistency

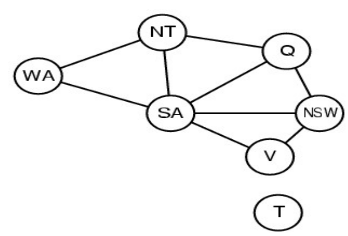
- Another form of propagation:
  - Make each arc consistent –C(X,Y) is consistent iff for every value of X there is some value of Y that satisfies C
- Idea:
  - Ensure that every binary constraint is satisfiable (2-consistency)
  - Binary constraints = arcs in the constraint graph
  - Remember: All higher-order constraints can be expressed as a set of binary constraints
- Can remove values from the domain of variables:
  - e.g. C (X, Y) : X > Y Domain (X) = { 1, 5, 11 } and Domain (Y) = { 3, 8, 15 }
    - For X=1 there is no value of Y such that 1>Y → remove 1 from domain X
    - For Y=15 there is no value of X such that X>15, so remove 15 from domain Y
    - We obtain more restricted domains Dom(X)={5,11} and Dom(Y)={3,8}
  - Have to try much fewer values in the search tree
- Removing a value from a domain may trigger further inconsistency, so we have to repeat the procedure until everything is consistent
  - For efficient implementation, we keep track of inconsistent arcs by putting them in a Queue
- This is stronger than forward checking. why?

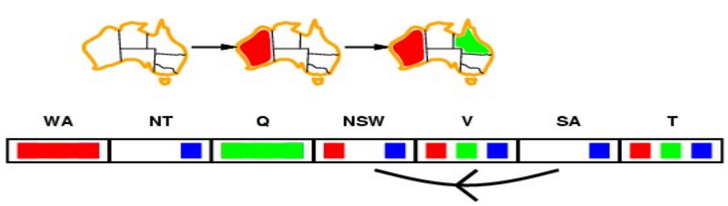
4/12/2024

76

Constraint Propagation: Arc Consistency

- Simplest form of propagation makes each arc consistent
- X →Y is consistent iff
  - for every value x of X there is some allowed y



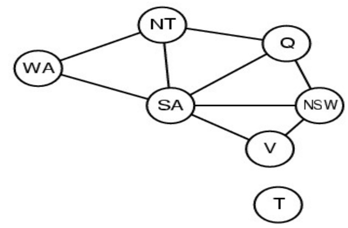


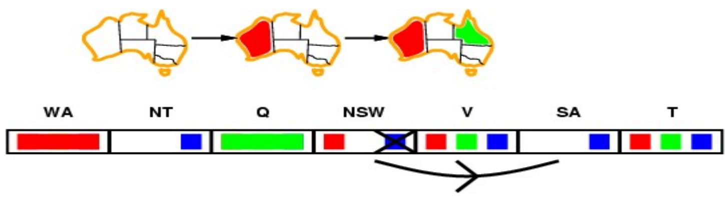
4/12/2024

77

Arc consistency

- Simplest form of propagation makes each arc consistent
- X →Y is consistent iff
  - for every value x of X there is some allowed y



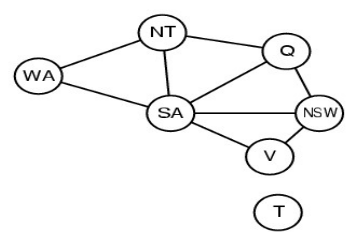


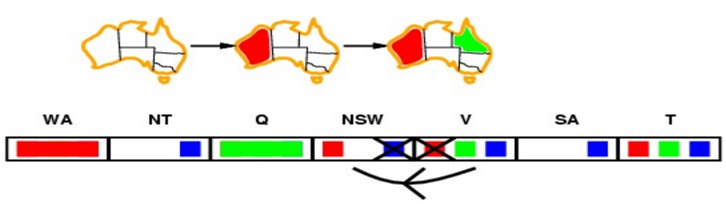
pra-sâmi

78

Arc consistency

- Simplest form of propagation makes each arc consistent
- X →Y is consistent iff
  - for every value x of X there is some allowed y





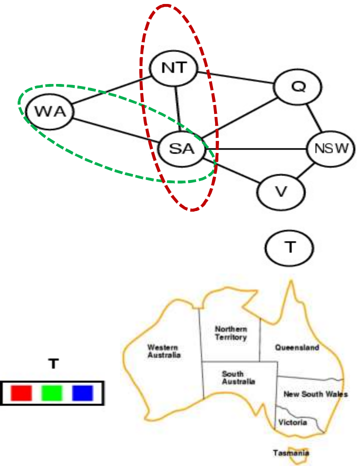
- If X loses a value, neighbors of X need to be rechecked
- Since NSW loses a value, we need to recheck all constraints involving NSW: other neighbors are Q, V
- Since V loses a value, we need to recheck all constraints involving V: other neighbors are SA
  - Recheck all constraints involving SA

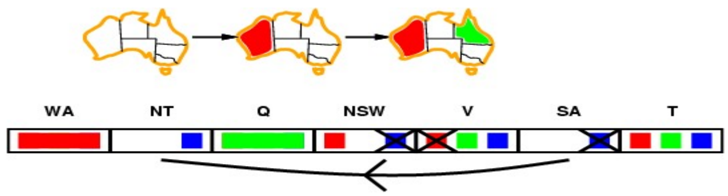
pra-sâmi

79

Arc consistency

- Simplest form of propagation makes each arc consistent
- X → Y is consistent iff
  - for every value x of X there is some allowed y.





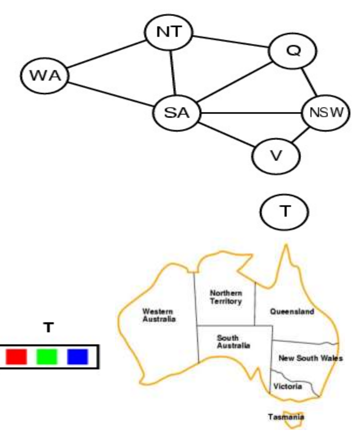
- (SA, NT) is not satisfiable any longer
  - we detected an unavoidable failure in the assignment {WA=red, Q=green}
  - Forward checking would have detected it as well. Why?

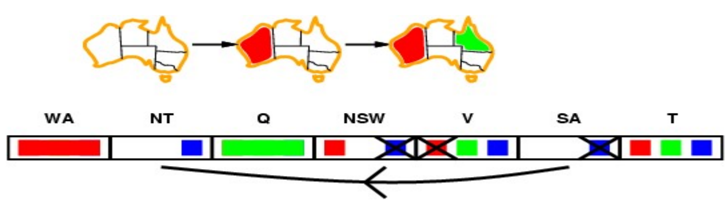
pra-sâmi

80

Arc consistency

- Simplest form of propagation makes each arc consistent
- X → Y is consistent iff
  - for every value x of X there is some allowed y.





- If X loses a value, neighbors of X need to be rechecked
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment

pra-sâmi

81

Arc consistency algorithm AC-3

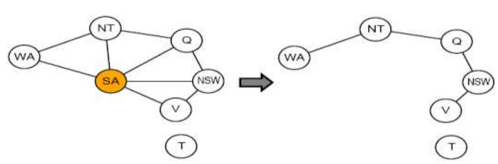
```
function AC-3( csp) returns the CSP, possibly with reduced domains
   inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
   local variables: queue, a queue of arcs, initially all the arcs in csp

   while queue is not empty do
      (Xi, Xj) ← REMOVE-FIRST(queue)
      if RM-INCONSISTENT-VALUES(Xi, Xj) then
         for each Xk in NEIGHBORS[Xi] do
            add (Xk, Xi) to queue

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

- Time complexity: $O(n^2 d^3)$

82

Variable Elimination

- Arc consistency simplifies the network by removing values of variables.
- A complementary method is variable elimination (VE), which simplifies the network by removing variables.



83

Example : Variable Elimination

- Variables A, B, and C
- Domain {1, 2, 3, 4}
- Constraints A < B and B < C

- Note: there may be plenty of other variables but B does not have any constraint on those

A	B
1	2
1	3
1	4
2	3
2	4
3	4

$\bowtie$

B	C
1	2
1	3
1	4
2	3
2	4
3	4

$=$

A	B	C
1	2	3
1	2	4
1	3	4
2	3	4

A	C
1	3
1	4
2	4

85

Local search for CSPs

- Hill-climbing, simulated annealing typically work with "complete" states, i.e., all variables assigned.

- To apply to CSPs:
  - Allow states with unsatisfied constraints.
  - Operators reassign variable values

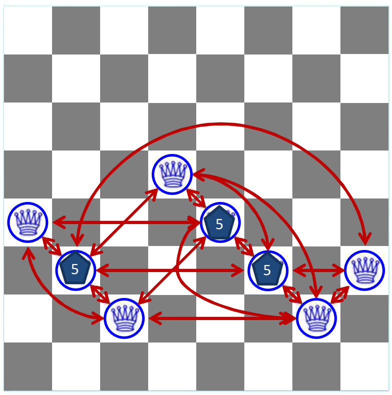
- Variable selection: randomly select any conflicted variable.

- Value selection by min-conflicts heuristic:
  - Choose value that violates the fewest constraints.
  - i.e. Hill-climb with h(n) = total number of violated constraints.

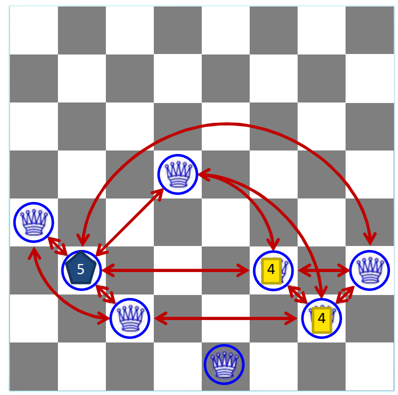
86

Local search for CSPs: n-Queen Problem

- Pick queens with maximum conflict



Eight Queen with heuristic cost estimate h =17



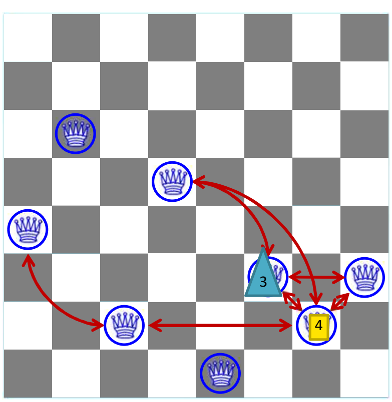
Queens with Highest Constraints moved

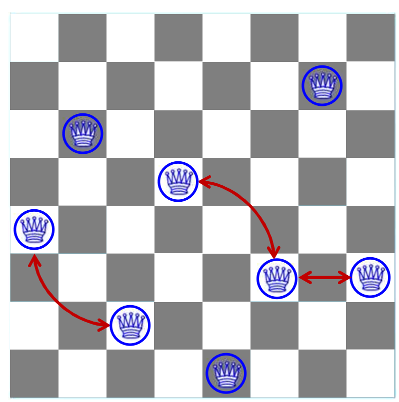
pra-sâmi

87

Local search for CSPs: n-Queen Problem

- Pick queens with maximum conflict



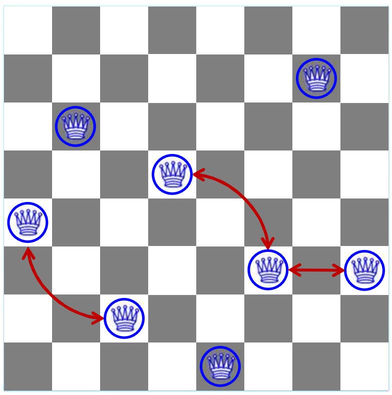


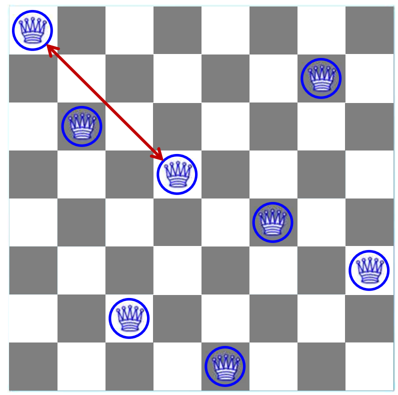
pra-sâmi

88

Local search for CSPs: n-Queen Problem

- Still one conflict – Local minima





pra-sâmi

89

Reflect…

- In which of the following situations might a blind search be acceptable?
  - real-life situation
  - complex game
  - small search space
  - all of the mentioned

- Which search method takes less memory?
  - Depth-First Search
  - Breadth-First search

- __________________ are mathematical problems defined as a set of objects whose state must satisfy a number of constraints or limitations.
  - Constraints Satisfaction Problems
  - Uninformed Search Problems
  - Local Search Problems
  - All of the mentioned

- Which of the Following problems can be modeled as CSP?
  - 8-Puzzle problem
  - 8-Queen problem
  - Map coloring problem
  - All of the mentioned

- What among the following constitutes to the incremental formulation of CSP?
  - Path cost
  - Goal cost
  - Successor function
  - All of the mentioned

pra-sâmi

90

Reflect…

- The term ___________ is used for a depth-first search that chooses values for one variable at a time and returns when a variable has no legal values left to assign.
  - Forward search
  - Backtrack search
  - Hill algorithm
  - Reverse-Down-Hill search

- To overcome the need to backtrack in constraint satisfaction problem can be eliminated by ____________
  - Forward Searching
  - Constraint Propagation
  - Backtrack after a forward search
  - Omitting the constraints and focusing only on goals

4/12/2024 pra-sâmi

91

4/12/2024 pra-sâmi