Constraint Satisfaction Problem Part 1

Thursday, 23 September 2021 2:42 PM

Logistics

- Tutorial Solutions to be released before that week's tutorial
  - To allow access to solutions as TAs go over them
- Midterm will be based on the material taught until Week 6
  - Open book
  - No internet access allowed during the exam. You must download all the files ahead of time. (Hint: Module materials are sufficient)
- Tutorial 1 Grades are available on Turnitin.
  - In case your handwriting was not legible, your grader may have left you a note; please reach out to the grader in that case.

2

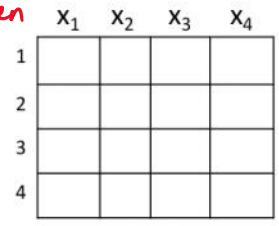
What you will learn Today

- A principled approach to model and solve complex problems
- The engine that ensures you AWS data does not leak to some other user
- The technique behind high profile auctions such as spectrum allocations
- What makes it possible for Microsoft to find bugs in device drivers
- How shipping companies such as FedEx route your packages are delivered on time

3

Constraint Satisfaction Problem (CSP)

- A CSP comprises of three components:
  1. Set of Variables $X = \{ x_1, x_2, ..., x_n \}$ → psn of queen in that col
  2. Set of domains corresponding to each variable: (1 to 4)
     $D : \{ Dx_1, Dx_2, ..., Dx_n \}$ where $Dx_i = domain(x_i)$
  3. Set of constraints



- Find the value for each of the variable in its domain that satisfies all the constraints.

4

Constraint Satisfaction Problem (CSP)

- A CSP comprises of three components:

1. Set of Variables X = { x_1, x_2, ..., x_n }
2. Set of domains corresponding to each variable:

   D : { Dx_1, Dx_2, ..., Dx_n } where Dx_i = *domain*(x_i)
3. Set of constraints

- Find the value for each of the variable in its domain that satisfies all the constraints.

5

4 Queen Problem as CSP

- Variables: { x_1, x_2,x_3, x_4 }

  Each x_i represents the variable for i^{th} column in 4 x 4 grid.
- Domain Dx_i : {1,2,3,4}

  Each variable can possibly be placed in any row.
- Constraints:

  NoAttack(x_i,x_j): True if Queen at x_i can not attack on Queen at x_j Else False.

6

4 Queen Problem as CSP

- Variables: { x_1, x_2,x_3, x_4 }

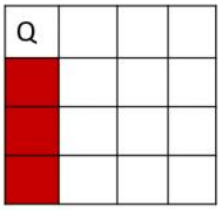
  Each x_i represents the variable for i^{th} column in 4 x 4 grid.

- Domain Dx_i : {1,2,3,4}

  Each variable can possibly be placed in any row.

- Constraints:

  NoAttack(x_i,x_j): True if Queen at x_i can not attack on Queen at x_j Else False.



x_1	x_2	NoAttack(x_1,x_2)
1	1	F
1	2	F
1	3	T

7

4 Queen Problem as CSP

- Variables: { x_1, x_2,x_3, x_4 }

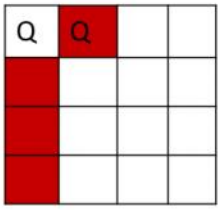
  Each x_i represents the variable for i^{th} column in 4 x 4 grid.

- Domain Dx_i : {1,2,3,4}

  Each variable can possibly be placed in any row.

- Constraints:

  NoAttack(x_i,x_j): True if Queen at x_i can not attack on Queen at x_j Else False.



x_1	x_2	NoAttack(x_1,x_2)
1	1	False

8

4 Queen Problem as CSP

- Variables: { x_1, x_2,x_3, x_4 }

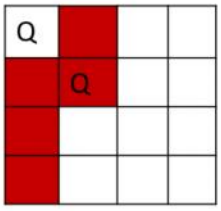
  Each x_i represents the variable for i^{th} column in 4 x 4 grid.

- Domain Dx_i : {1,2,3,4}

  Each variable can possibly be placed in any row.

- Constraints:

  NoAttack(x_i,x_j): True if Queen at x_i can not attack on Queen at x_j Else False.



x_1	x_2	NoAttack(x_1,x_2)
1	1	False
1	2	False

9

4 Queen Problem as CSP

- Variables: { x_1, x_2,x_3, x_4 }

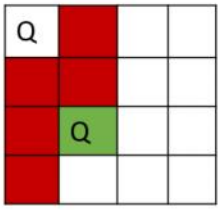
  Each x_i represents the variable for i^{th} column in 4 x 4 grid.

- Domain Dx_i : {1,2,3,4}

  Each variable can possibly be placed in any row.

- Constraints:

  NoAttack(x_i,x_j): True if Queen at x_i can not attack on Queen at x_j Else False.



x_1	x_2	NoAttack(x_1,x_2)
1	1	False
1	2	False
1	3	True

10

4 Queen Problem as CSP

- Variables: { x_1, x_2,x_3, x_4 }

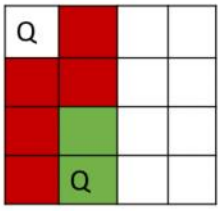
  Each x_i represents the variable for i^{th} column in 4 x 4 grid.

- Domain Dx_i : {1,2,3,4}

  Each variable can possibly be placed in any row.

- Constraints:

  NoAttack(xi,xj): True if Queen at x_i can not attack on Queen at x_j Else False.



x_1	x_2	NoAttack(x_1,x_2)
1	1	False
1	2	False
1	3	True
1	4	True
...	....	...

11

Backtracking Algorithm (Attempt I)

First assign value to each variable, then check for consistency.
If not consistent, Backtrack.

↳ all constraints satisfied

	x_1	x_2	x_3	x_4
1	Q			
2				
3				
4				

12

Backtracking Algorithm (Attempt I)

First assign value to each variable, then check for consistency.
If not consistent, Backtrack.

	x_1	x_2	x_3	x_4
1	Q	Q		
2				
3				
4				

13

Backtracking Algorithm (Attempt I)

First assign value to each variable, then check for consistency.
If not consistent, Backtrack.

	x_1	x_2	x_3	x_4
1	Q	Q	Q	
2				
3				
4				

14

Backtracking Algorithm (Attempt I)

First assign value to each variable, then check for consistency.
If not consistent, Backtrack.

	x_1	x_2	x_3	x_4
1	Q	Q	Q	Q
2				
3				
4				

Backtracking Algorithm (Attempt I)

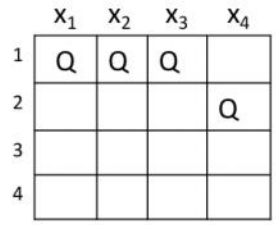
First assign value to each variable, then check for consistency.
If not consistent, Backtrack.

	x_1	x_2	x_3	x_4
1	Q	Q	Q	Q
2				
3				
4				

Backtrack !

Backtracking Algorithm (Attempt I)

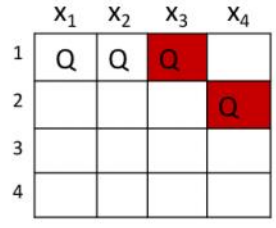
First assign value to each variable, then check for consistency.
If not consistent, Backtrack.



17

Backtracking Algorithm (Attempt I)

First assign value to each variable, then check for consistency.
If not consistent, Backtrack.

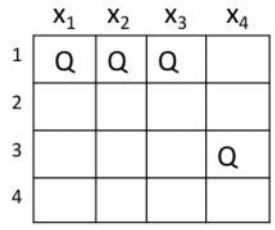


Backtrack !

18

Backtracking Algorithm (Attempt I)

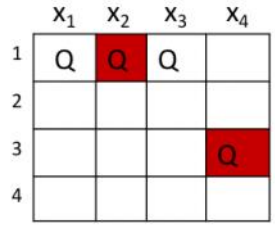
First assign value to each variable, then check for consistency.
If not consistent, Backtrack.



19

Backtracking Algorithm (Attempt I)

First assign value to each variable, then check for consistency.
If not consistent, Backtrack.



Backtrack !

20

Backtracking Algorithm (Attempt I)

First assign value to each variable, then check for consistency.
If not consistent, Backtrack.

	x_1	x_2	x_3	x_4
1	Q	Q	Q	
2				
3				
4				Q

21

Backtracking Algorithm (Attempt I)

First assign value to each variable, then check for consistency.
If not consistent, Backtrack.

	x_1	x_2	x_3	x_4
1	Q	Q	Q	
2				
3				
4				Q

Backtrack !

22

Backtracking Algorithm (Attempt I)

First assign value to each variable, then check for consistency.
If not consistent, Backtrack.

	x_1	x_2	x_3	x_4
1	Q	Q		
2			Q	
3				
4				

23

Backtracking Algorithm (Attempt I)

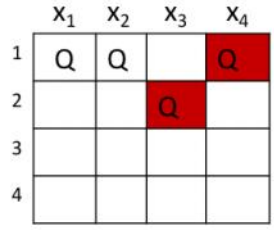
First assign value to each variable, then check for consistency.
If not consistent, Backtrack.

	x_1	x_2	x_3	x_4
1	Q	Q		Q
2			Q	
3				
4				

24

Backtracking Algorithm (Attempt I)

First assign value to each variable, then check for consistency.
If not consistent, Backtrack.



Backtrack !

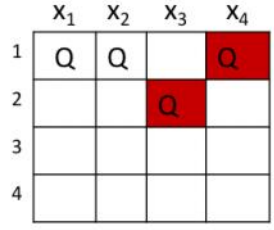
Needs too many steps to find a solution.

Backtracking Algorithm (Attempt I)

First assign value to each variable, then check for consistency.
If not consistent, Backtrack.

```
BacktrackingSearch(prob, assign)
 1: if AllVarsAssigned(prob, assign) then
 2:     if isConsistent(assign) then
 3:         return assign
 4:     else
 5:         return failure
 6: var ← PickUnassignedVar(prob, assign)
 7: for value ∈ OrderDomainValue(var, prob, assign) do
 8:     assign ← assign ∪ (var = value)
 9:     result ← BacktrackingSearch(prob, assign)
10:     if result! = failure then return result
11:     assign ← assign \ (var = value)
12: return failure
```

will always return F



Backtrack !

Backtracking Algorithm (Attempt II)

Before assigning value, checks if it is consistent with the previous assignments.

	x_1	x_2	x_3	x_4
1	Q			
2				
3				
4				

27

Backtracking Algorithm (Attempt II)

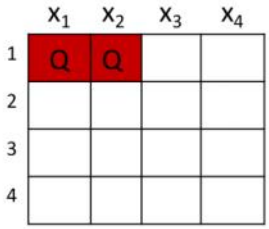
Before assigning value, checks if it is consistent with the previous assignments.

	x_1	x_2	x_3	x_4
1	Q	Q		
2				
3				
4				

28

Backtracking Algorithm (Attempt II)

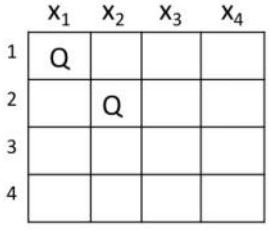
Before assigning value, checks if it is consistent with the previous assignments.



29

Backtracking Algorithm (Attempt II)

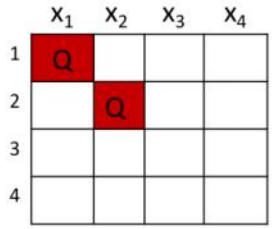
Before assigning value, checks if it is consistent with the previous assignments.



30

Backtracking Algorithm (Attempt II)

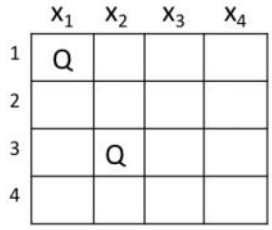
Before assigning value, checks if it is consistent with the previous assignments.



31

Backtracking Algorithm (Attempt II)

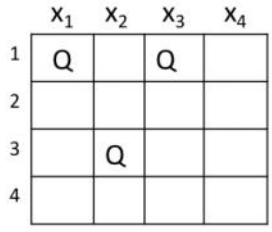
Before assigning value, checks if it is consistent with the previous assignments.



32

Backtracking Algorithm (Attempt II)

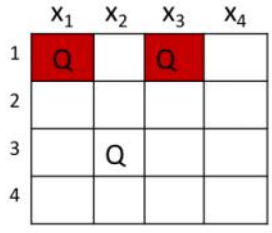
Before assigning value, checks if it is consistent with the previous assignments.



33

Backtracking Algorithm (Attempt II)

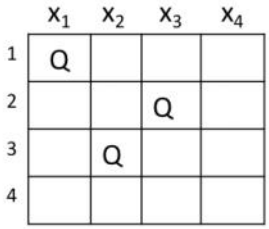
Before assigning value, checks if it is consistent with the previous assignments.



34

Backtracking Algorithm (Attempt II)

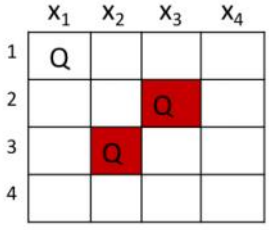
Before assigning value, checks if it is consistent with the previous assignments.



35

Backtracking Algorithm (Attempt II)

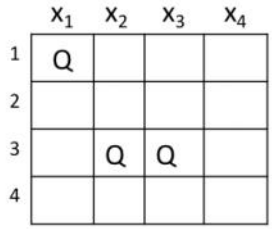
Before assigning value, checks if it is consistent with the previous assignments.



36

Backtracking Algorithm (Attempt II)

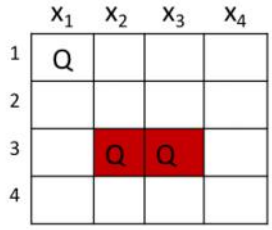
Before assigning value, checks if it is consistent with the previous assignments.



37

Backtracking Algorithm (Attempt II)

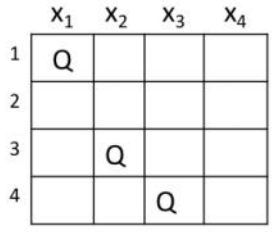
Before assigning value, checks if it is consistent with the previous assignments.



38

Backtracking Algorithm (Attempt II)

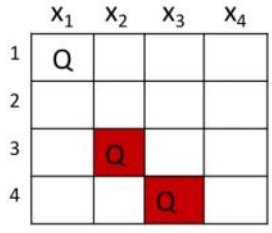
Before assigning value, checks if it is consistent with the previous assignments.



39

Backtracking Algorithm (Attempt II)

Before assigning value, checks if it is consistent with the previous assignments.



Can not assign any value to x_3 which is consistent with the previous assignment. Backtrack !

40

Backtracking Algorithm (Attempt II)

Before assigning value, checks if it is consistent with the previous assignments.

	x_1	x_2	x_3	x_4
1	Q			
2				
3				
4		Q		

41

Backtracking Algorithm (Attempt II)

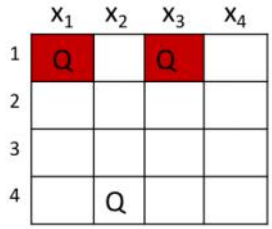
Before assigning value, checks if it is consistent with the previous assignments.

	x_1	x_2	x_3	x_4
1	Q		Q	
2				
3				
4		Q		

42

Backtracking Algorithm (Attempt II)

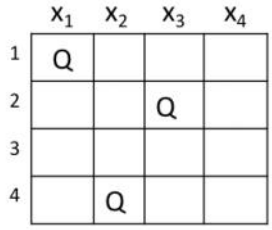
Before assigning value, checks if it is consistent with the previous assignments.



43

Backtracking Algorithm (Attempt II)

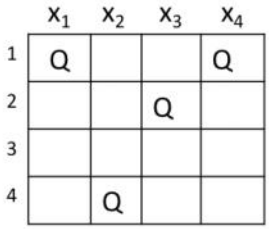
Before assigning value, checks if it is consistent with the previous assignments.



44

Backtracking Algorithm (Attempt II)

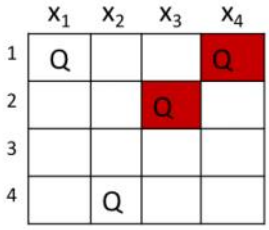
Before assigning value, checks if it is consistent with the previous assignments.



45

Backtracking Algorithm (Attempt II)

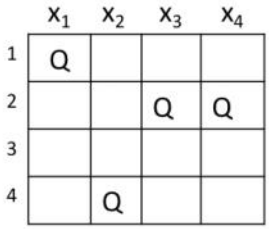
Before assigning value, checks if it is consistent with the previous assignments.



46

Backtracking Algorithm (Attempt II)

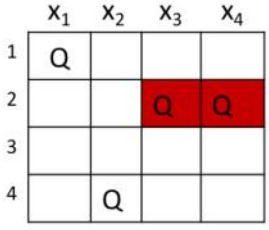
Before assigning value, checks if it is consistent with the previous assignments.



47

Backtracking Algorithm (Attempt II)

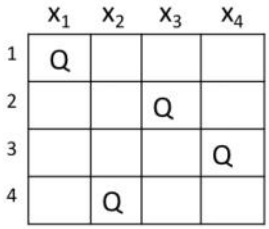
Before assigning value, checks if it is consistent with the previous assignments.



48

Backtracking Algorithm (Attempt II)

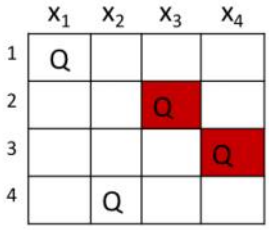
Before assigning value, checks if it is consistent with the previous assignments.



49

Backtracking Algorithm (Attempt II)

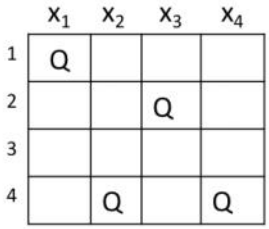
Before assigning value, checks if it is consistent with the previous assignments.



50

Backtracking Algorithm (Attempt II)

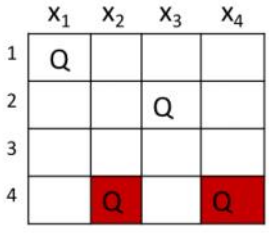
Before assigning value, checks if it is consistent with the previous assignments.



51

Backtracking Algorithm (Attempt II)

Before assigning value, checks if it is consistent with the previous assignments.

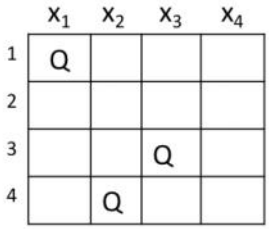


Can not assign any value to x_4 which is consistent with the previous assignment. Backtrack !

52

Backtracking Algorithm (Attempt II)

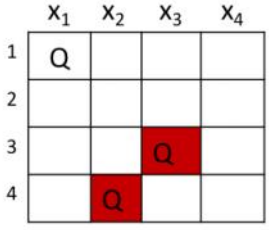
Before assigning value, checks if it is consistent with the previous assignments.



53

Backtracking Algorithm (Attempt II)

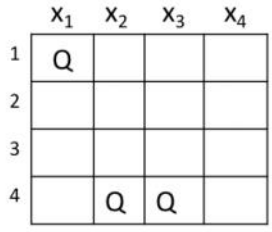
Before assigning value, checks if it is consistent with the previous assignments.



54

Backtracking Algorithm (Attempt II)

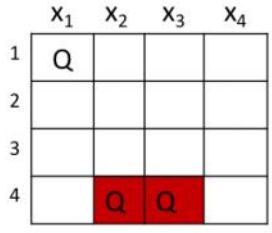
Before assigning value, checks if it is consistent with the previous assignments.



55

Backtracking Algorithm (Attempt II)

Before assigning value, checks if it is consistent with the previous assignments.

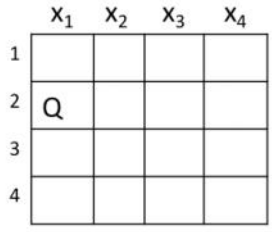


Can not assign any value to x_3 which is consistent with the previous assignment. Backtrack !

56

Backtracking Algorithm (Attempt II)

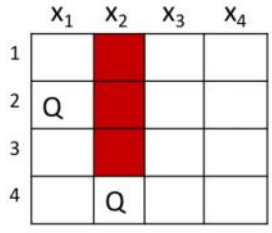
Before assigning value, checks if it is consistent with the previous assignments.



57

Backtracking Algorithm (Attempt II)

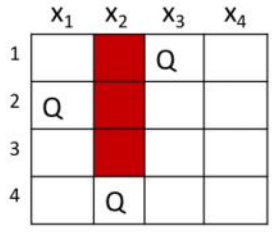
Before assigning value, checks if it is consistent with the previous assignments.



58

Backtracking Algorithm (Attempt II)

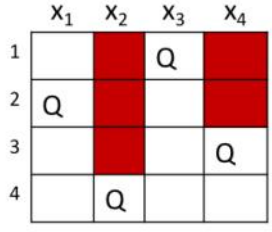
Before assigning value, checks if it is consistent with the previous assignments.



59

Backtracking Algorithm (Attempt II)

Before assigning value, checks if it is consistent with the previous assignments.



60

Backtracking Algorithm (Attempt II)

Before assigning value, checks if it is consistent with the previous assignments.

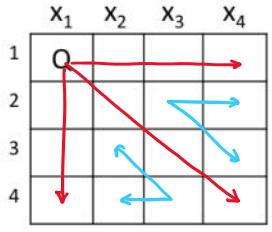
```
BACKTRACKINGSEARCH(prob, assign)
1: if ALLVARASSIGNED(prob, assign) then return assign
2: var ← PICKUNASSIGNEDVAR(prob, assign)
3: for value in ORDERDOMAINVALUE(var, prob, assign) do
4:     if VALISCONSISTENTWITHASSIGNMENT(value, assign) then
5:         assign ← assign ∪ (var = value)
6:         result ← BACKTRACKINGSEARCH(prob, assign)
7:         if result!=failure then return result
8:         assign ← assign \ (var = value)
9: return failure
```

set minus

61

Backtracking Algorithm with Inference

- Assign value to x_i and ***infer*** the restrictions on rest of the variables



$x_2 \neq 1, 2$

$x_3 \neq 1, 3$

$x_4 \neq 1, 4$

If $x_3 = 2$,
$x_4 \neq 2, 3$ ← no solution!

If $x_3 = 4$,
$x_2 \neq 3, 4$ ← no solution!

$x_3 \neq 1, 2, 3, 4$
↳ $x_1 \neq 1$

☆ inference takes a long time
↳ balance assign & check w/ inferring

62

Backtracking Algorithm with Inference

BacktrackingSearch_with_Inference($prob, assign$)

```
1: if AllVariablesAssigned(prob, assign) then return assign
2: var ← PickUnassignedVar(prob, assign)
3: for value in OrderDomainValue(var, prob, assign) do
4:     if ValIsConsistentWithAssignment(value, assign) then
5:         assign ← assign ∪ (var = value)
6:         inference ← Infer(prob, var, assign)
7:         assign ← assign ∪ inference
8:         if inference!=failure then
9:             result ← BacktrackingSearch.(prob, assign)
10:            if result!=failure then return result
11:        assign ← assign \ {(var = value) ∪ inference}
12: return failure
```

63