CSC384
Constraint Satisfaction Problems
Part 1

Bahar Aameri & Sonya Allin

Summer 2020

Credits

CSP slides are drawn from or inspired by a multitude of sources including :

Alan Mackworth
Faheim Bacchus
Sheila McIlraith
Andrew Moore
Hojjat Ghaderi
Craig Boutillier

Constraint Satisfaction Problems (CSPs)

- Chapter 6
  - 6.1: Formalism
  - 6.2: Constraint Propagation
  - 6.3: Backtracking Search for CSP
  - 6.4 is about local search which is a very useful idea but we won't cover it in class.

Also see Poole and Mackworth chapter 4:
https://artint.info/2e/html/ArtInt2e.Ch4.html

Constraint Satisfaction Problems (CSPs) – Introduction

- **Uninformed search problems**
  - use problem-specific state representations and heuristics;
  - are generally concerned about determining paths from the current state to goal states;
  - view states as black boxes with no internal structures.
- **Constraint Satisfaction Problems (CSPs)**
  - care less about paths and more about final (goal) configurations;
  - take advantage of a general state representation.
  - the uniform state representation allows design of more efficient algorithms.
- Techniques for solving CSPs have many practical applications in industry.

- Represent **states** as vectors of feature values.[1]
  - A set of k variables (known as **features**).
  - **Each variable** has a domain of different values.
  - A **state** is specified by an assignment of values to all variables.
  - A **partial state** is specified by an assignment of a value to some of the variables.
- A **goal** is specified as conditions on the vector of feature values.
- **Solving a CSP**: find a set of values for the features (variables) so that the values **satisfy** the specified conditions (constraints).

e.g. (CSC384, CSC269)

e.g. (ba, sa, sm, fb)

CSC384 = ba
CSC269 = ?

CSC384 = ba
CSC269 = sm

[1] Feature vectors provide a general state representation that is useful in many other areas of AI, particularly Machine Learning, Reasoning under Uncertainty, and Computer Vision.

* 8:13 come back *

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

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

Example: Sudoku

- Each **variable** represent a cell.
- **Domain**: a single value for cells already filled in; the set $\{1, ..., 9\}$ for empty cells.
- **State**: any completed board given by specifying the value in each cell.
- **Partial State**: some incomplete filling out of the board.
- **Constrains**: The variables that form
  - a column must be distinct;
  - a row must be distinct;
  - a sub-square must be distinct.

A **CSP** consists of

- A set of **variables** $V_1, ..., V_n$;
- A (finite) **domain** of possible values $Dom[V_i]$ for each variable V_i;
- A set of **constraints** $C_1, ..., C_m$.

- Each variable V_i can be assigned any value from its domain:

$$V_i = d \qquad \text{where} \qquad d \in Dom[V_i]$$

- Each constraint C
  - Has a set of variables it operates over, called its **scope**.
    Example: The scope of $C(V_1, V_2, V_4)$ is $\{V_1, V_2, V_4\}$
  - Given an assignment to variables the C returns
    True if the assignment satisfies the constraint;
    False if the assignment falsifies the constraint.

Formalization of a CSP

- **Solution** to a CSP: An assignment of a value to all of the variables such that every constraint is satisfied.
- A CSP is **unsatisfiable** if no solution exists.

Types of Constraints

- **Unary** Constraints (over one variable)
  $C(X) : X = 2;$
  $C(Y) : Y > 5$

- **Binary** Constraints (over two variables)
  $C(X, Y) : X + Y < 6$

- **Higher-order** constraints: over 3 or more variables.
  $ALL - Diff(V_1, .., V_n)$: $V_1 \neq V_2, V_1 \neq V_3, ..., V_2 \neq V_1, ..., V_n \neq V_1, ..., V_n \neq V_{n-1}$.[2]

Answer a Question! http://etc.ch/xyX3

[2] Later, we will see that this collection of binary constraints has less pruning power than $ALL - Diff$, so $ALL - Diff$ appears in many CSP problems.

Constraint Table

- We can specify the constraints with a table

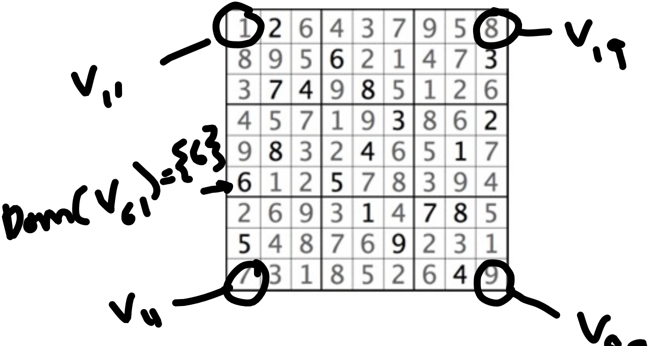
C(1,1,2)= False

V1	V2	V4	C(V1,V2,V4)
1	1	1	False
1	1	2	False
1	2	1	False
1	2	2	False
2	1	1	True
2	1	2	False
2	2	1	False
2	2	2	False
3	1	1	False
3	1	2	True
3	2	1	True
3	2	2	False

T, F

- Often we can specify the constraint more compactly with an expression.

V2 > 1



- **Variables**: $V_{11}, V_{12}, ..., V_{21}, V_{22}, ..., V_{91}, ..., V_{99}$
- **Domains**: $Dom[V_{ij}] = \{1, 2, .., 9\}$ for empty cells
  $Dom[V_{ij}] = \{k\}$, where k is a fixed value, for filled cells.

- **Constraints**:

  - Row constraints:

$All - Diff(V_{11}, V_{12}, V_{13}, ..., V_{19})$

$All - Diff(V_{21}, V_{22}, V_{23}, ..., V_{29})$

...

$All - Diff(V_{91}, V_{12}, V_{13}, ..., V_{99})$

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

$$\begin{cases} V_{11} \neq V_{12} \\ V_{12} \neq V_{13} \\ V_{13} \neq V_{14} \quad V_{14} \neq V_{15} \end{cases}$$

- **Constraints**:

  - Row constraints:

$$All - Diff(V_{11}, V_{12}, V_{13}, ..., V_{19})$$
$$All - Diff(V_{21}, V_{22}, V_{23}, ..., V_{29})$$
$$...$$
$$All - Diff(V_{91}, V_{12}, V_{13}, ..., V_{99})$$

  - Column Constraints:

$$All - Diff(V_{11}, V_{21}, V_{31}, ..., V_{91})$$
$$All - Diff(V_{12}, V_{22}, V_{32}, ..., V_{92})$$
$$...$$
$$All - Diff(V_{19}, V_{29}, V_{39}, ..., V_{99})$$

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

Example: Sudoku

- **Constraints**:

  - Row constraints:

$All - Diff(V_{11}, V_{12}, V_{13}, ..., V_{19})$
$All - Diff(V_{21}, V_{22}, V_{23}, ..., V_{29})$
...
$All - Diff(V_{91}, V_{12}, V_{13}, ..., V_{99})$

  - Column Constraints:

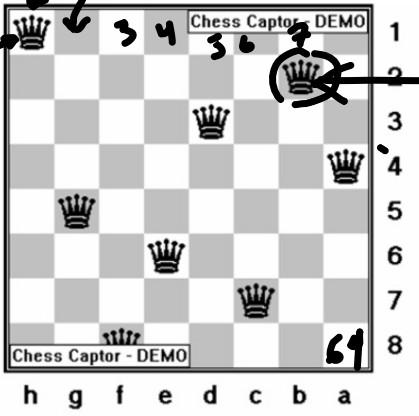
$All - Diff(V_{11}, V_{21}, V_{31}, ..., V_{91})$
$All - Diff(V_{12}, V_{22}, V_{32}, ..., V_{92})$
...
$All - Diff(V_{19}, V_{29}, V_{39}, ..., V_{99})$

  - Sub-Square Constraints:

$All{-}Diff(V_{11}, V_{12}, V_{13}, V_{21}, V_{22}, V_{23}, V_{31}, V_{32}, V_{33})$,
...,
$All{-}Diff(V_{77}, V_{78}, V_{79}, ..., V_{97}, V_{98}, V_{99})$

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

Problem Statement: Place N Queens on an $N \times N$ chess board so that no Queen can attack any other Queen.



Problem formulation: N Queens

- **Variables:** $Q_1, Q_2, Q_3 \ldots Q_i \ldots Q_N$

size N

- **Domains:** $\{1, 2, 3, 4 \ldots\ldots N \cdot N\}$

size $N \cdot N$

Q_1: $N \cdot N$ possible value assignments

Q_2: $N \cdot N$ "

# of configurations is $(N \cdot N)^N$

Answer a Question! http://etc.ch/xyX3

Is there a better way to represent the N-queens problem? We know we cannot place two queens in a single row.

Problem Statement: Place N Queens on an $N \times N$ chess board so that no Queen can attack any other Queen.

Num variable s+ domain size

Better Formulation:

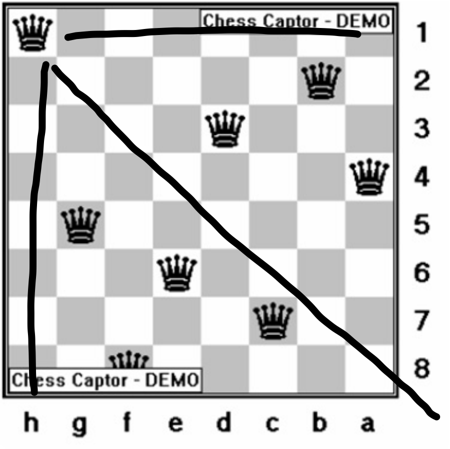
- **Variables:** Q_1 Q_2 Q_3 .... Q_N
- **Domains:** $\{1, 2 \ldots N\}$

N Queens, Dom size of N

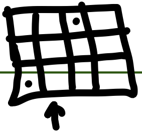
N^N configuration

Answer a Question! http://etc.ch/xyX3

Example: N-Queens



No. $3 = Q_1$, $1 = Q_3$



Constraints:

- Cannot put two Queens in same column: $Q_1, Q_2, \ldots Q_N$

  $Q_i \neq Q_j$

  $Dom(Q_i) = \{1, \ldots 9\}$

- Diagonal constraints:

  $|Q_i - Q_j| \neq |i - j|$

Q_1	Q_2	$C(Q_1, Q_2)$
		$\neq$
1	1	
1	2	T

Solving CSPs: CSP as a Search Problem

A CSP could be formulated as a search problem:

- **Initial State**: Empty assignment.
- **Successor Function**: Assigned values to an unassigned variable.
- **Goal Test**:
  (1) The assignment is complete
  (2) No constraints is violated.

CSP Backtracking Search - Intuition

CSPs do NOT require finding a path (to a goal). They only need the **configuration** of the goal state.
CSPs are best solved by a specialized version search called **Backtracking Search**.

Key Intuitions:

- Searching through the space of partial assignments, rather than paths.
- Decide on a suitable value for one variable at a time.
  Order in which we assign the variables does not matter.
- If a constraint is falsified during the process of partial assignment, immediately reject the current partial assignment.

CSP Backtracking Search - Intuition

CSP Search Tree:

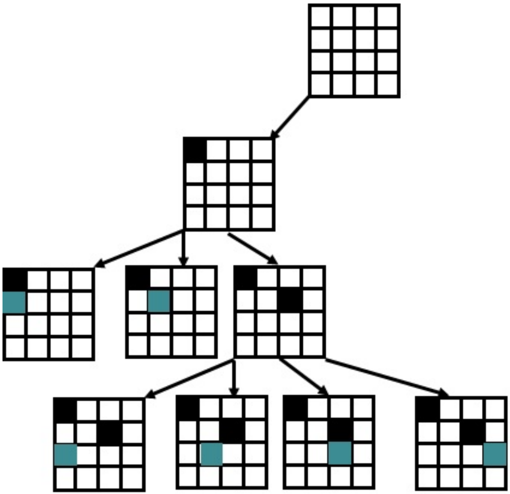
- **Root**: Empty Assignment.
- **Children** of a node: all possible value assignments for a particular unassigned variable.
- The tree **stops descending** if an assignment violates a constraint.
- **Goal Node**:
  (1) The assignment is complete
  (2) No constraints is violated.

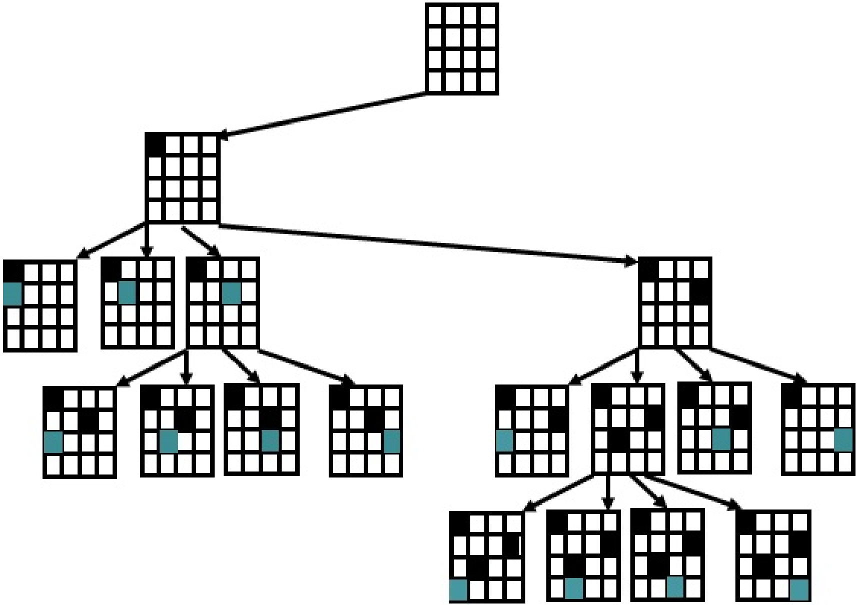
Draw a CSP search tree for Sudoku

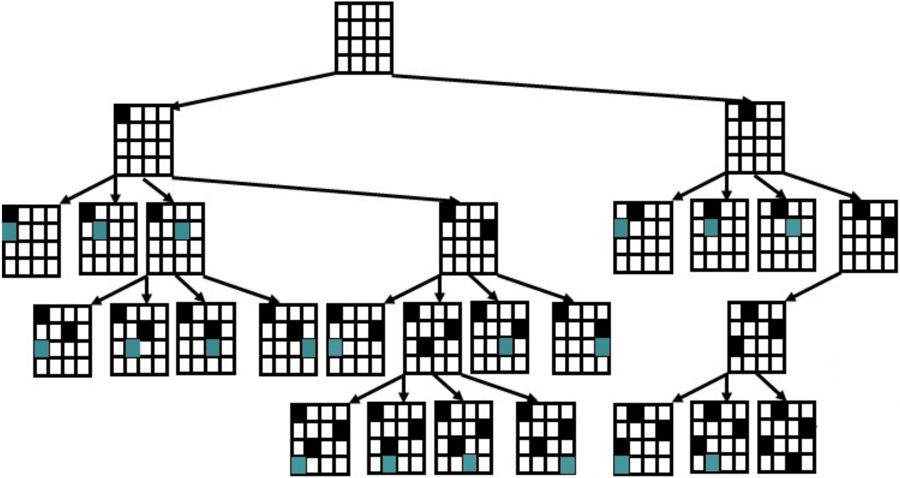
Answer a Question! http://etc.ch/xyX3

Example: 4-Queens

Draw the CSP search tree for 4-Queens.







We will apply a **recursive** implementation:

- If all variables are set, print the solution and terminate.
- Otherwise:
  - Pick an unassigned variable V and assign it a value.
  - Test the constraints corresponding with V and all other variables of them are assigned.
  - If a constraint is unsatisfied, return (backtrack).
  - Otherwise, go one level deeper by invoking a recursive call.

```
def BT(Level):
1.  if all Variables assigned
2.     PRINT Value of each Variable
3.     EXIT or RETURN                   # EXIT for only one solution
                                        # RETURN for more solutions
4.  V := PickUnassignedVariable()
5.  Assigned[V] := TRUE
6.  for d := each member of Domain(V)   # the domain values of V
7.     Value[V] := d
8.     ConstraintsOK := TRUE
9.     for each constraint C such that (i) V is a variable of C and
                                       (ii) all other variables of C are assigned:
10.         if C is not satisfied by the set of current assignments:
11.                 ConstraintsOK := FALSE
12.    if ConstraintsOk == TRUE:
13.         BT(Level+1)
14.  Assigned[V] := FALSE     # UNDO as we have tried all of V's values
15.  RETURN
```

Answer a Question! http://etc.ch/xyX3