Constraint Satisfaction

Real-world problems

Scheduling

Building design

Planning

Optimization/satisfaction

VLSI design

Maximizing GPA

Registering for classes

Sudoku

Crossword puzzles

Sudoku as CSP

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

Constraints:
- fill a 9×9 grid with digits 1-9
- each column contains all digits from 1-9
- each row contains all digits from 1-9
- each box contains all digits from 1-9

Image source: https://en.wikipedia.org/wiki/Sudoku

Sudoku as CSP

- Consider the red cell.
  - It cannot have the values 5, 3 or 7 because those already appear in that row.
  - It cannot have the value 8, because it already appears in that column.
  - It cannot have the values 5, 3, 6, 9 or 8 because they already appear in the box.
  - This leaves the following values that are still consistent with the constraints: 1, 2, 4 and 7.

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

Sudoku as CSP

- When it comes to solving Sudoku problems, there are several strategies for picking cells and numbers.
- Most of us will not just pick a random number from the set of remaining values, i.e. 1, 2, 4 and 7 of the red cell.
- However, a computer is fast and can effectively and super-efficiently solve Sudoku problems by picking a random number.
- We will explore that strategy, called backtracking search in the remainder of these slides

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

Formal Definition of CSP

A constraint satisfaction problem (CSP) consists of

- a set of variables $X = \{x_1, x_2, \ldots, x_n\}$
  - each with an associated domain of values $\{d_1, d_2, \ldots, d_n\}$.
  - the domains are typically finite
- a set of constraints $\{c_1, c_2, \ldots, c_m\}$ where
  - each constraint defines a predicate which is a relation over a particular subset of X.
  - e.g., C_i involves variables $\{X_{i1}, X_{i2}, \ldots, X_{ik}\}$ and defines the relation $Ri \subseteq D_{i1} \times D_{i2} \times \ldots D_{ik}$

Goals of CSP

An **instantiation** of a subset of variables S is an assignment of a legal domain value to each variable in S

An instantiation is **legal** iff it does not violate any (relevant) constraints.

A **solution** is an instantiation of all of the variables in the problem.

CSP as a Search Problem

- States are defined by the values assigned so far
  - Initial state: the empty assignment { }
  - Successor function: assign a value to an unassigned variable that does not conflict with current assignment (fail if no legal assignments)
  - Goal test: the current assignment is complete
- Every solution appears at depth *n* with *n* variables
- Path is irrelevant

Backtracking Search

1. Consider the variables in some order
2. Pick an unassigned variable and give it a provisional value such that it is consistent with all of the constraints
3. If no such assignment can be made, we have reached a dead end and need to backtrack to the previous variable
4. Continue this process until a solution is found or we backtrack to the initial variable and have exhausted all possible values

Backtracking Example: Sudoku

- Consider the Sudoku problem on the right
- Let assign variables (or cell values) by row, left to right.

3	6	0	0	2	0	0	8	9
0	0	0	3	6	1	0	0	0
0	0	0	0	0	0	0	0	0
8	0	3	0	0	0	6	0	2
4	0	0	6	0	3	0	0	7
6	0	7	0	0	0	1	0	8
0	0	0	0	0	0	7	0	0
0	0	0	4	1	8	0	0	0
9	7	0	0	3	0	0	1	4

Backtracking Example: Sudoku

- We begin by setting the value of cell <0,2> to 1
- Notice that 1 is consistent with the row, column and box constraints
- We then set the value of cell <0,3> to 5.
- The algorithm will begin with 1, but notice that 1 is not consistent, neither are 2, 3 and 4.

~~1~~,~~2~~,~~3~~,~~4~~,5

1

~~1~~,~~2~~,~~3~~,4

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

3	6	1	5	2	4	0	8	9
0	0	0	3	6	1	0	0	0
0	0	0	0	0	0	0	0	0
8	0	3	0	0	0	6	0	2
4	0	0	6	0	3	0	0	7
6	0	7	0	0	0	1	0	8
0	0	0	0	0	0	7	0	0
0	0	0	4	1	8	0	0	0
9	7	0	0	3	0	0	1	4

Backtracking Example: Sudoku

- Next, we set cell <0,5> to 4, again, we start with 1, but realize that 1, 2 and 3 are violating row, column and box constraints
- We then look at cell <0,6>.
- As it turns out all values from 1-9 violate some constraint.
- We now backtrack.

~~1~~,~~2~~,~~3~~,~~4~~,5

1

~~1~~,~~2~~,~~3~~,4

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

3	6	1	5	2	4	0	8	9
0	0	0	3	6	1	0	0	0
0	0	0	0	0	0	0	0	0
8	0	3	0	0	0	6	0	2
4	0	0	6	0	3	0	0	7
6	0	7	0	0	0	1	0	8
0	0	0	0	0	0	7	0	0
0	0	0	4	1	8	0	0	0
9	7	0	0	3	0	0	1	4

Backtracking Example: Sudoku

- Specifically, we undo the assignment of the value 4 to cell <0,5>.
- However our algorithm does not reconsider values 1-4, instead it continues to go up.
- In other words, we try to assign 5 to cell <0,5>.
- However, we already have 5 in the first row.

1

~~1~~,~~2~~,~~3~~,~~4~~,5

~~1~~,~~2~~,~~3~~,4

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

3	6	1	5	2	4	0	8	9
0	0	0	3	6	1	0	0	0
0	0	0	0	0	0	0	0	0
8	0	3	0	0	0	6	0	2
4	0	0	6	0	3	0	0	7
6	0	7	0	0	0	1	0	8
0	0	0	0	0	0	7	0	0
0	0	0	4	1	8	0	0	0
9	7	0	0	3	0	0	1	4

Backtracking Example: Sudoku

- The digit 6 already exists in the box.
- However, 7 is fine and that is the number we will assign to cell <0,5>
- We now move to the next cell, cell <0,6>
- This time, just like in the movie Groundhog day, we start from 0.
- We find that while 1-3 violate constraints, 4 is fine.

1

~~1~~,~~2~~,~~3~~,~~4~~,5

~~1~~,~~2~~,~~3~~,~~4~~,~~5~~,~~6~~,7

~~1~~,~~2~~,~~3~~,4

3	6	1	5	2	7	4	8	9
0	0	0	3	6	1	0	0	0
0	0	0	0	0	0	0	0	0
8	0	3	0	0	0	6	0	2
4	0	0	6	0	3	0	0	7
6	0	7	0	0	0	1	0	8
0	0	0	0	0	0	7	0	0
0	0	0	4	1	8	0	0	0
9	7	0	0	3	0	0	1	4

Backtracking search algorithm

```
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
```

Algorithm source: Russell and Norvig: AIMA, 2nd Edition, p 142