Constraint Satisfaction

Real-world problems

Scheduling

Building design

Planning

Optimization/satisfaction

VLSI design

Maximizing GPA

Registering for classes

Sudoku

Crossword puzzles

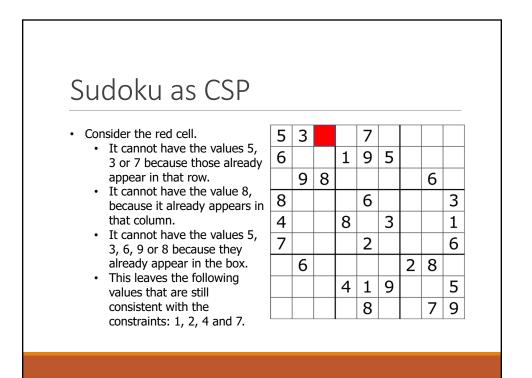
Sudoku as CSP

5 6	3			7				
6			1	9	5			
	9	8					6	
8				6				3
8 4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5 9
				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: <u>https://en.wikipedia.org/wiki/Sudoku</u>



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 superefficiently solve Sudoku problems by picking a random number.
- We will explore that strategy, called backtracking search in the remainder of these slides

5 6	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 $X = \{x_1, x_2, ..., x_n\}$
 - each with an associated domain of values $\{d_1, d_2, ..., d_n\}$.
 - the domains are typically finite
- a set of constraints {c₁, c₂, ..., c_m} where
 - each constraint defines a predicate which is a relation over a particular subset of X.
 - $^\circ$ e.g., C_i involves variables {X_{i1}, X_{i2}, ..., X_{ik}} and defines the relation Ri \subseteq D_{i1} x D_{i2} x ... 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.

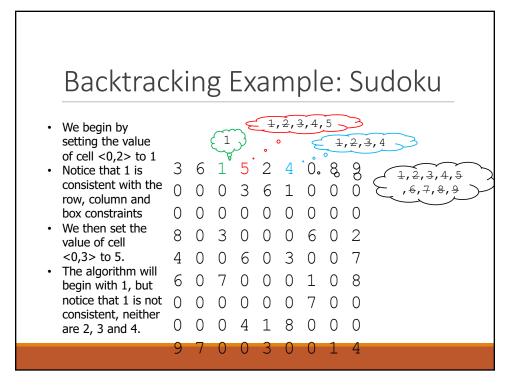
CSP as a Search Problem

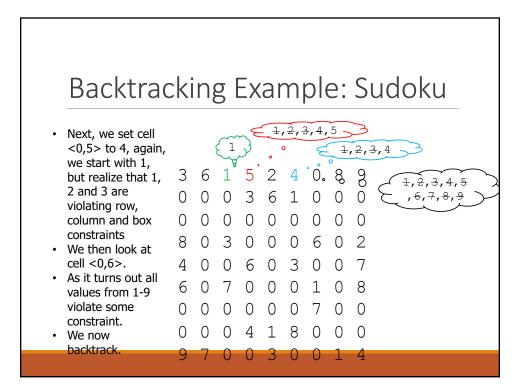
- States are defined by the values assigned so far
 Initial state: the empty assignment { }
 - Successor function: assign a value to an unassigned variable that does not conflict with current assignment (fail if no legal assignments)
 - Goal test: the current assignment is complete
- Every solution appears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

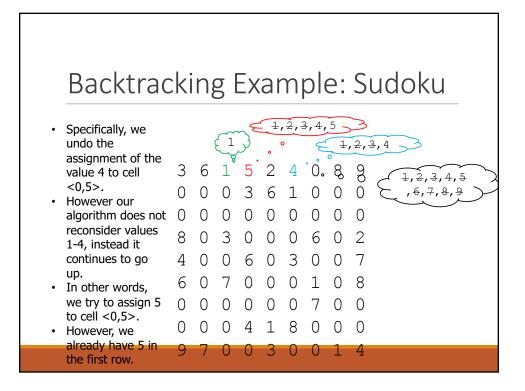
Backtracking Search

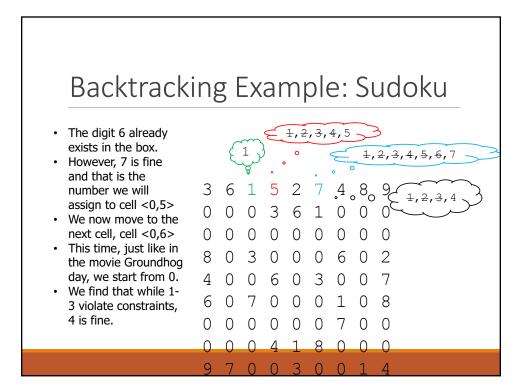
- 1. Consider the variables in some order
- 2. Pick an unassigned variable and give it a provisional value such that it is consistent with all of the constraints
- 3. If no such assignment can be made, we have reached a dead end and need to backtrack to the previous variable
- 4. Continue this process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

Backtracking E	ха	m	np	le	: 5	Su	d	зk	Ű
0			•						
Consider the Sudoku problem on the right	3	6	0	0	2	0	0	8	9
on the rightLet assign variables (or cell	0	0	0	3	6	1	0	0	0
values) by row, left to right.	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)

 $\label{eq:constraction} \begin{array}{l} \textbf{function} \ \textbf{Recursive-Backtracking} (\textit{assignment, csp}) \ \textbf{returns} \ \textbf{a} \ \textbf{solution}, \ \textbf{or} \ \textbf{failure} \end{array}$

if assignment is complete then return assignment var \leftarrow 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

 $result \leftarrow \text{Recursive-Backtracking}(assignment, csp)$

if $result \neq failue$ then return result

remove { var = value } from assignment

 ${\bf return}\ failure$

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