

9: CONSTRAINT SATISFACTION

COMP702: CLASSICAL ARTIFICIAL INTELLIGENCE



PAPER CLUB

For next week's seminar, please read:

A.M. Smith, M. Mateas. Answer Set Programming for Procedural Content Generation: A Design Space Approach, IEEE Transactions on Computational Intelligence and AI in Games, 2011.

(PDF Link on LearningSpace)



CONSTRAINT SATISFACTION PROBLEMS



CONSTRAINT SATISFACTION PROBLEMS (CSP)

- A Constraint Satisfaction Problem (CSP) is defined by:
 - A set $\{X_1, \dots, X_n\}$ of variables
 - A set $\{D_1, \dots, D_n\}$ of domains, specifying what values each variable can take
 - A set $\{C_1, \dots, C_m\}$ of constraints that the variables must satisfy
- A solution is an assignment of a value to every variable which satisfies all the constraints

SUDOKU

- A 9x9 grid
- Each square contains a number 1-9
- A number cannot appear more than once in any given row, column, or 3x3 square outlined in bold
- Some numbers are given – the puzzle is to fill in the rest
- A properly designed Sudoku puzzle has exactly one solution

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

SUDOKU AS A CSP

- Variables: a subset of $\{X_{1,1}, X_{1,2}, \dots, X_{9,9}\}$
 - One variable per empty square
- All have the same domain: $D_{i,j} = \{1, \dots, 9\}$
- Constraints:
 - $X_{i,j} \neq X_{i,k}$ for all i, j, k with $j \neq k$
 - $X_{j,i} \neq X_{k,i}$ for all i, j, k with $j \neq k$
 - $X_{i,j} \neq X_{k,l}$ for i, j and k, l in the same 3x3 square

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

SOLVING SUDOKU

BACKTRACKING

- Pick an empty square
- Try a possible value for that empty square
- Try to solve the rest of the puzzle (recursively)
- If we run into a dead end, go back and try a different value

SOLVING SUDOKU

BACKTRACKING

Procedure SolveBacktracking(square)

For $n = 1, \dots, 9$

 If square can have value n

 Set value of square to n

 If square is the last empty square or SolveBacktracking(next_empty_square)

 Return true

 End If

 End If

End For

Clear the value of square

Return false

End Procedure

SOLVING SUDOKU

CONSTRAINT PROPAGATION

- Each empty square keeps track of which numbers it could possibly contain
- Update every square
- If a square has only one possible number, fill it in
- Repeat until stuck

SOLVING SUDOKU

CONSTRAINT PROPAGATION

Procedure SolveConstraintPropagation()

Repeat

For each empty square

Determine the list of possible numbers that could go in the square

If the list is empty

Return false // *unsolvable*

Else if the list has a single element

Fill in the number

End If

End For

Until no changes are made

Return true

End Procedure

BACKTRACKING VS CONSTRAINT PROPAGATION

- Backtracking always finds a solution, but is inefficient
- Constraint propagation is efficient, but doesn't always find a solution
- More sophisticated constraint propagation algorithms do exist
- We could also combine the two
 - Use backtracking to “unstick” constraint propagation
 - Use constraint propagation to narrow down the options that backtracking needs to consider

SOLVING SUDOKU

BACKTRACKING + CONSTRAINT PROPAGATION

Procedure SolveConstraintPropagation()

Repeat

For each empty square

Determine the list of possible numbers

If the list is empty

Return false // unsolvable

Else if the list has a single element

Fill in the number

End If

End For

Until no changes are made

If the puzzle is solved

Return true

Else

Return SolveBacktracking(first empty square)

End Procedure

Procedure SolveBacktracking(square)

For $n = 1, \dots, 9$

If square can have value n

Save the state of the board

Set value of square to n

If SolveConstraintPropagation()

Return true

End If

Restore the saved state of the board

End If

End For

Clear the value of square

Return false

End Procedure

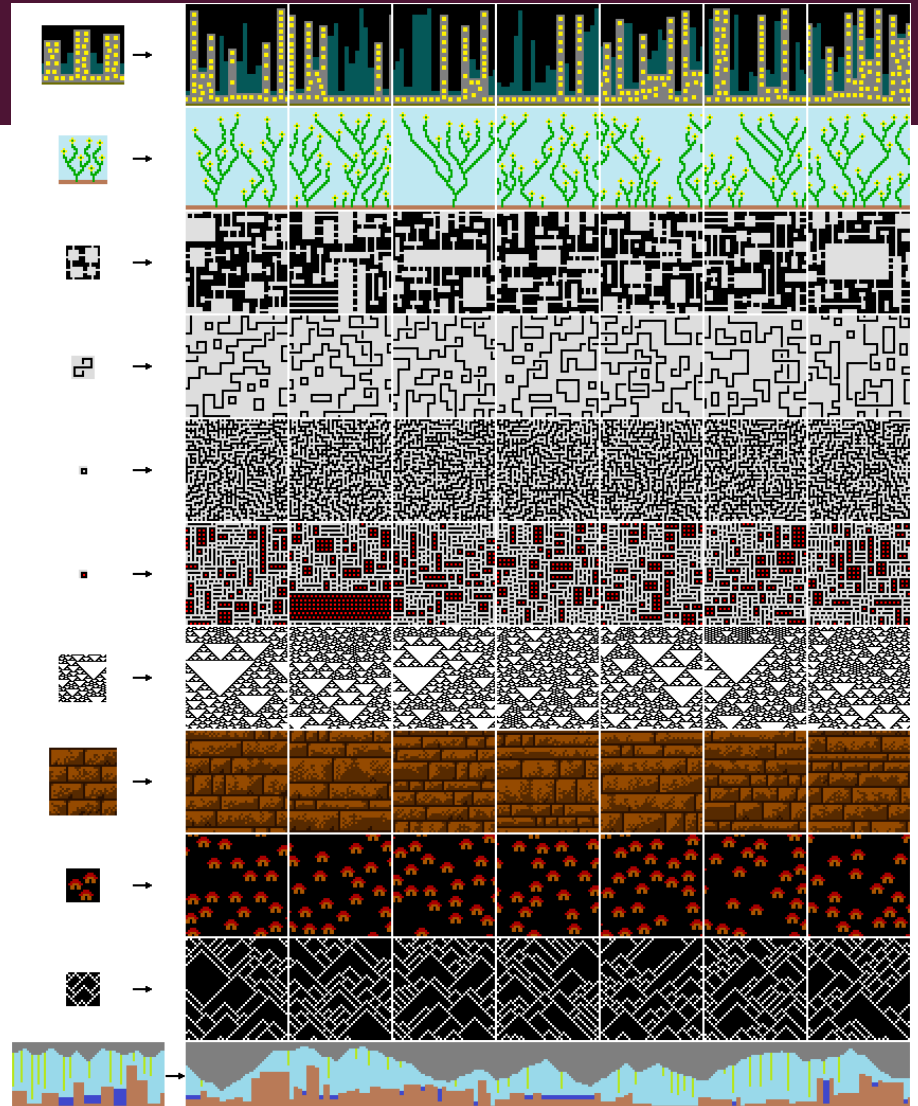


WAVE FUNCTION COLLAPSE



WAVE FUNCTION COLLAPSE

- Procedural Content Generation algorithm
- Invented by Maxim Gumin in 2016
- Named after a concept from quantum mechanics



WAVE FUNCTION COLLAPSE

THE IDEA

- Space is divided into cells, each of which has a state
 - E.g. pixels with a colour; voxels with a 3D mesh from a modular kit
- Constraints on what cell states can neighbour each other are extracted from examples
 - A basic form of machine learning
 - Constraints are probabilistic – not just what is allowed next to a given cell, but what is more or less likely
- Each cell has a probability distribution over states
- A cell is chosen, its state is set according to its probabilities, neighbouring cells are updated according to constraints
- The algorithm backtracks if it gets stuck

CHOOSING THE NEXT CELL

Named after Claude Shannon (1916-2001),
the “father of Information Theory”

- A cell is chosen which has minimal Shannon entropy

$$S = - \sum_i P_i \log P_i$$

- Lower Shannon entropy = less uncertainty about the possible value of the cell

WAVE FUNCTION COLLAPSE RESOURCES

- <https://github.com/mxgmnn/WaveFunctionCollapse>
- <http://www.proccjam.com/tutorials/wfc/>
- <http://oskarstalberg.com/game/wave/wave.html>
- <https://twitter.com/exutumno>