Slithering the Link

David Anderson, Dan Hamalainen, Edward Kwiatkowski,
Valerie Lambert, Sam Spaeth

Carleton College Department of Computer Science

March 12, 2016

Roadmap

What is a Slitherlink Puzzle?

How to Define a Slitherlink Puzzle

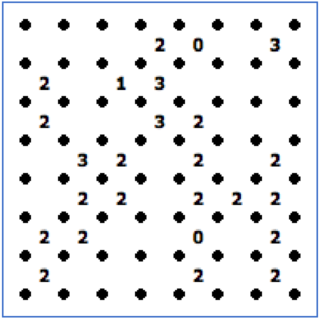
How to Solve a Slitherlink Puzzle

How to Make a Slitherlink Puzzle

What is Slitherlink?

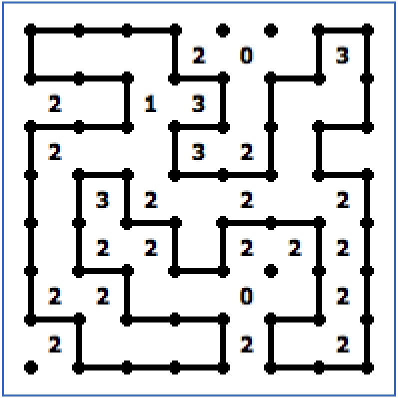
Logic puzzle developed by Nikoli
Played on:

- a rectangular lattice of dots, creating "cells"
- with some cells containing numbers



What is Slitherlink?

Objective of the game is to create a single loop throughout the puzzle where:

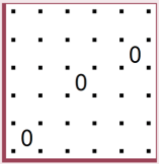


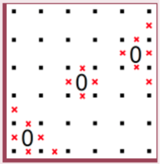
- the final solution is a continuous line that does not cross itself
- each numbered cell corresponds to the number of solution lines around it
- the puzzle should have ONLY ONE unique solution

Solving a Slitherlink Puzzle

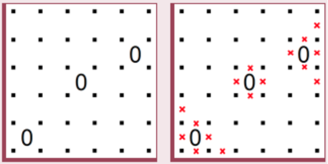
	2	1	3			3
	3		3		2	
	3		3	2	3	
2			2	1		
2			3	2	2	
2			0			1
			3	3	2	2

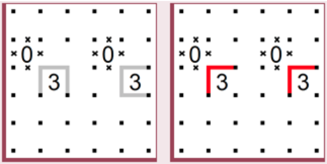
Solving a Slitherlink Puzzle



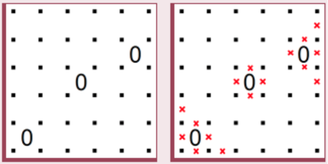


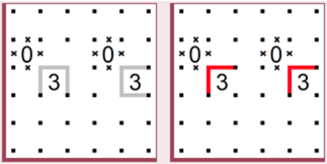
Solving a Slitherlink Puzzle

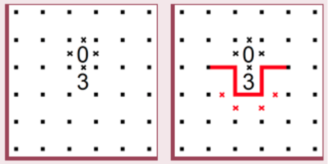




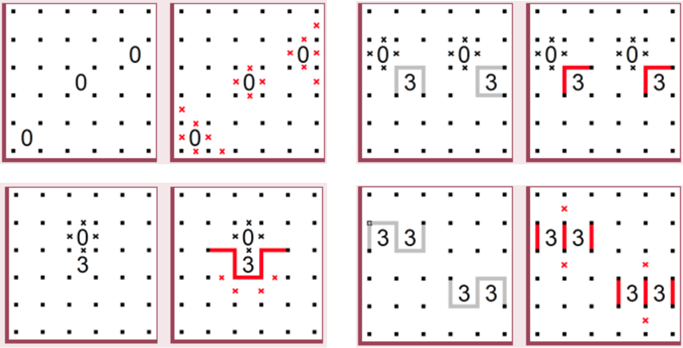
Solving a Slitherlink Puzzle



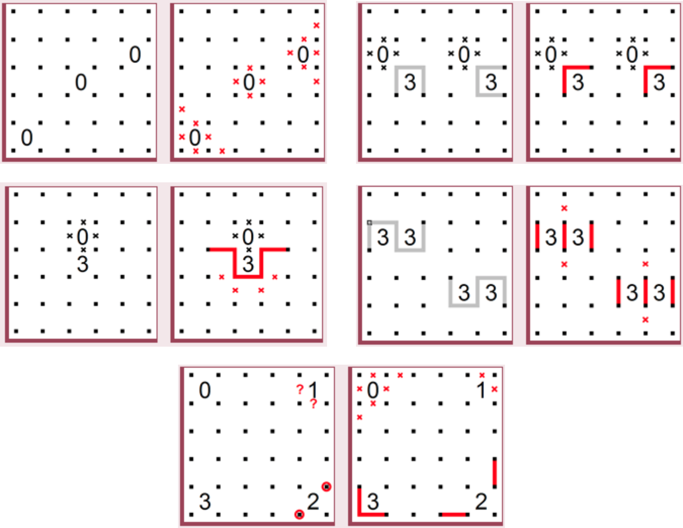




Solving a Slitherlink Puzzle

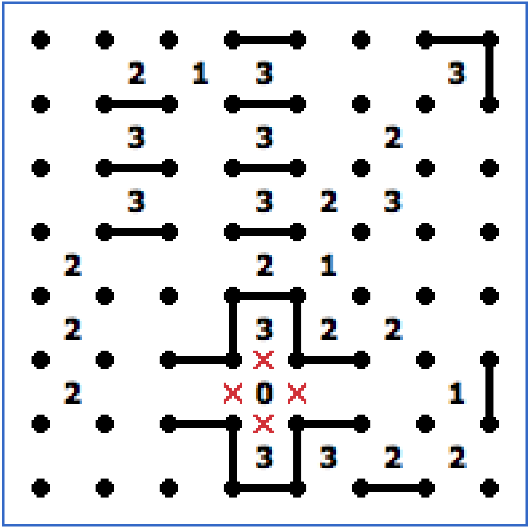


Solving a Slitherlink Puzzle



Conceptis Puzzles Slitherlink Techniques

Solving a Slitherlink Puzzle



Solving a Slitherlink Puzzle

Solving a Slitherlink puzzle is an NP-complete problem, as well as determining if there are multiple solutions.

On the NP-completeness of the Slither Link Puzzle Takayuki YATO

Puzzle Representation

Components of a puzzle:

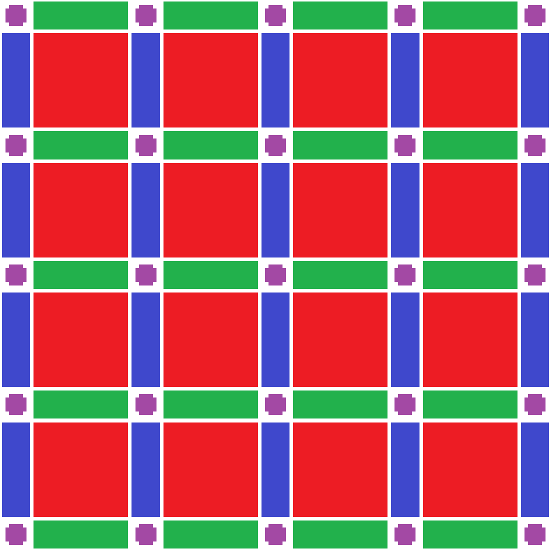
- the grid
  - lines
  - numbers
- rules and contradictions
- contours
- what it means to be solved

The Grid

M by N grid has 3 2D arrays:

- M by N 2D array for numbers; values 0 to 3 or empty
- $M + 1$ by N 2D array for horizontal lines; values line, x, or empty
- M by $N + 1$ 2D array for vertical lines; values line, x, or empty

A rule-based approach to the puzzle of Slither Link. Stefan Herting.



Rules and Contradictions

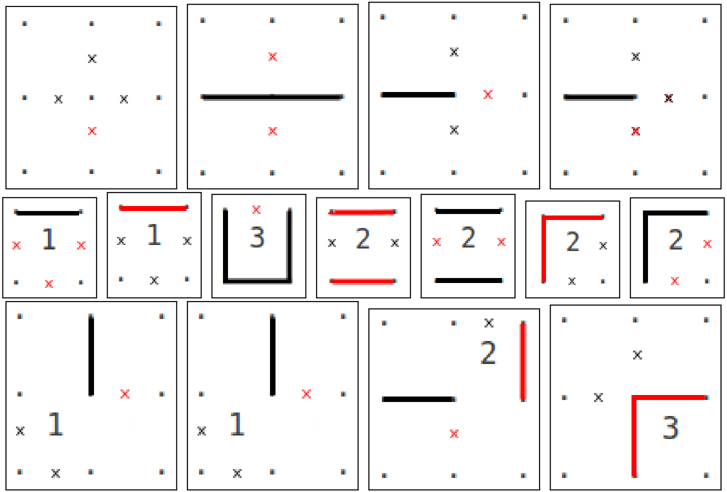
Each rule has:

- dimensions
- prerequisites
- consequences

Each contradiction has:

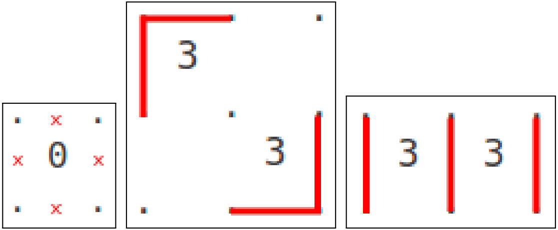
- dimensions
- prerequisities

Examples of Rules

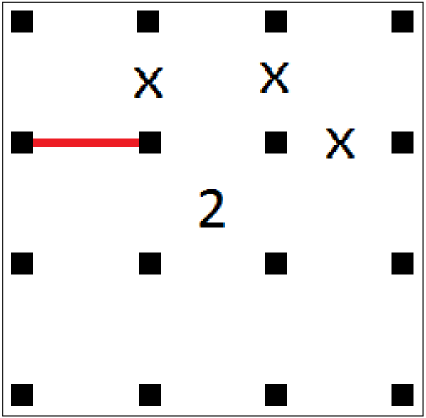


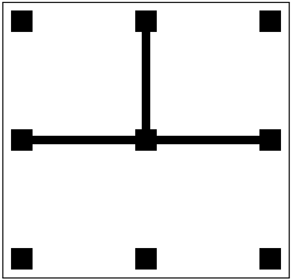
Static Rules

Static rules are rules that do not contain lines or x's in their prerequisites. We identified 3 static rules.



Rule and Contradiction in action





We chose to cover rules that are at most 3 by 3 in dimension, and contradictions that are at most 2 by 2 in dimension.

Contours

- Use 2D array to keep track of contour endpoints.
- Update endpoints as we add lines.
- Keep track of the number of open and closed contours as we add lines.

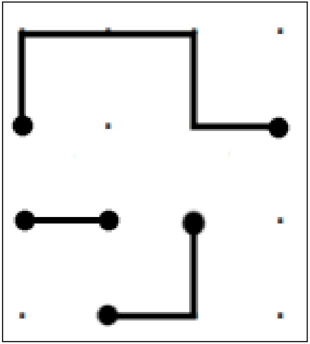
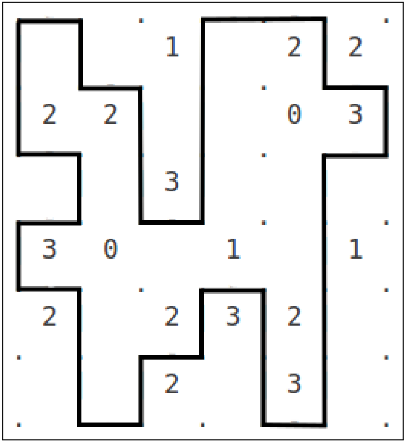


Table 1: Contour Endpoint Array

3,1			0,1
1,2	0,2	1,3	
	2,2		

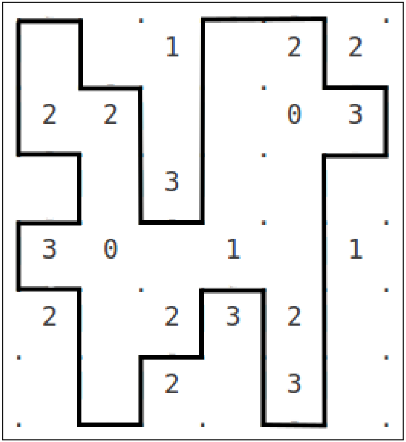
$$numClosed\_ = 0$$
$$numOpen\_ = 3$$

How Can We Tell Our Grid is Solved?



- Every number in the grid is satisfied

How Can We Tell Our Grid is Solved?



- Every number in the grid is satisfied
- There is exactly one closed loop, and no open loops.

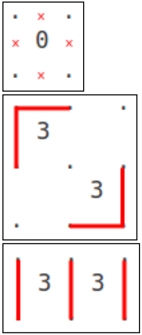
Applying Rules

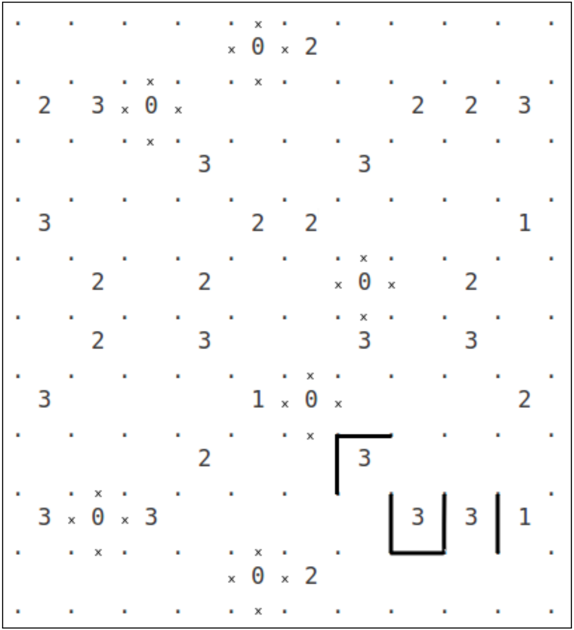
- for every position in the grid...
  - for every defined rule...
    - for every orientation...

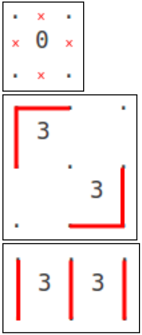
Do the prerequisites in the rule match where we're looking at on the grid?

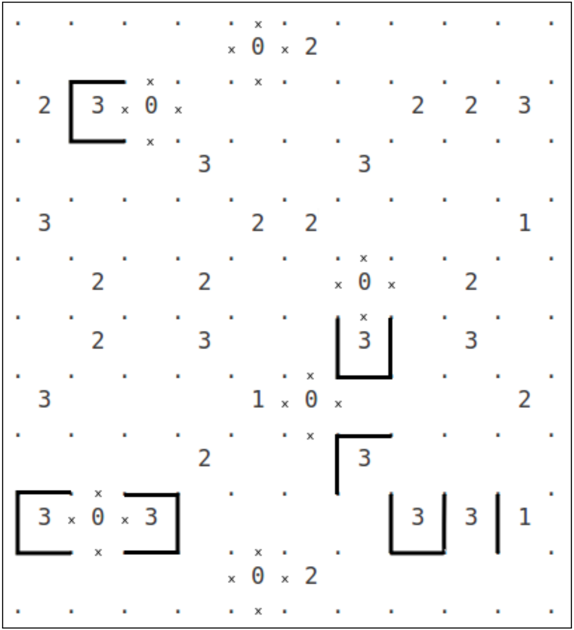
- If so, add consequences to the grid.

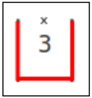
				0	2				
2	3	0					2	2	3
			3			3			
3				2	2				1
	2		2			0		2	
	2		3			3		3	
3				1	0				2
			2			3			
3	0	3					3	3	1
				0	2				

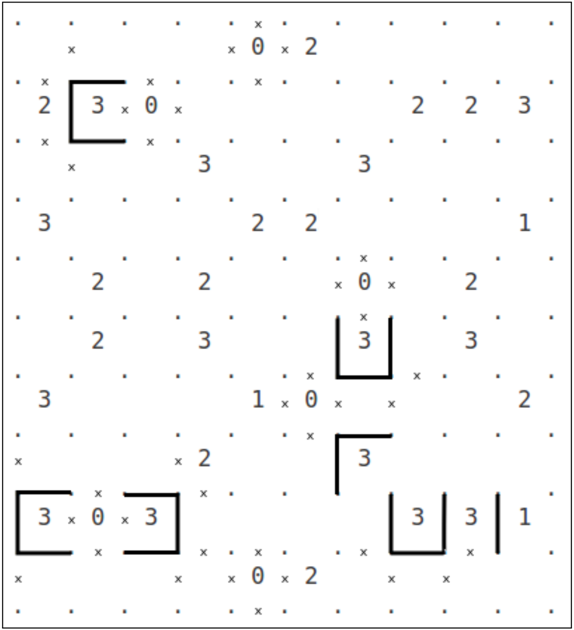


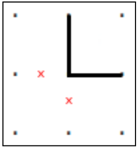

0 2
2 3 0 2 2 3
3 3
3 2 2 1
2 2 0 2
2 3 3 3
3 1 0 2
2 3
3 0 3 3 3 1
0 2

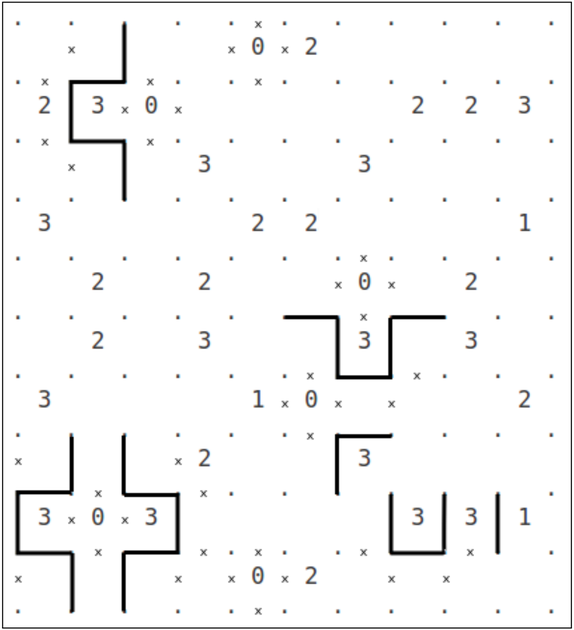

0
3
3
3 3

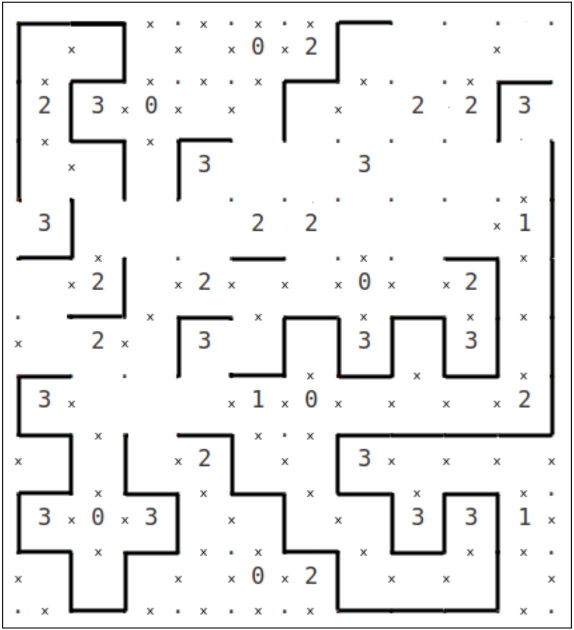





0 2
2 3 0 2 2 3
3 3
3 2 2 1
2 2 0 2
2 3 3 3
3 1 0 2
2 3
3 0 3 3 3 1
0 2







Guessing

Method:

Guessing

Method:

1. Find a particular open position

Guessing

Method:

1. Find a particular open position
2. Guess that position is a LINE

Guessing

Method:

1. Find a particular open position
2. Guess that position is a LINE
    2.1 Run deterministic rules

Guessing

Method:

1. Find a particular open position
2. Guess that position is a LINE
    2.1 Run deterministic rules
    2.2 If there's a contradiction, we know the position is an X

Guessing

Method:

1. Find a particular open position
2. Guess that position is a LINE
   2.1 Run deterministic rules
   2.2 If there's a contradiction, we know the position is an X
3. Guess that position is an X

Guessing

Method:

1. Find a particular open position
2. Guess that position is a $\textsc{Line}$
   2.1 Run deterministic rules
   2.2 If there's a contradiction, we know the position is an X
3. Guess that position is an X
   3.1 Run deterministic rules

Guessing

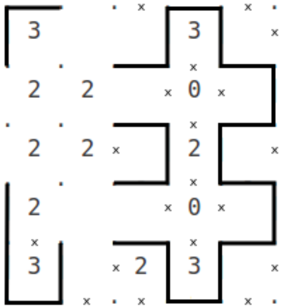
Method:

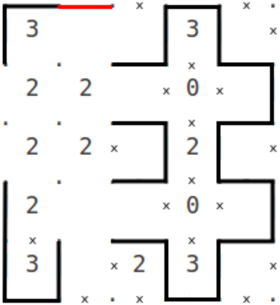
1. Find a particular open position
2. Guess that position is a $\textsc{Line}$
   2.1 Run deterministic rules
   2.2 If there's a contradiction, we know the position is an X
3. Guess that position is an X
   3.1 Run deterministic rules
   3.2 If there's a contradiction, we know the position is a $\textsc{Line}$

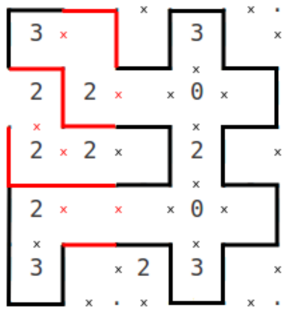
Guessing

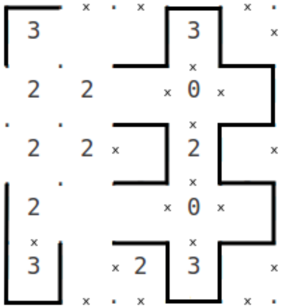
Method:

1. Find a particular open position
2. Guess that position is a Line
   2.1 Run deterministic rules
   2.2 If there's a contradiction, we know the position is an X
3. Guess that position is an X
   3.1 Run deterministic rules
   3.2 If there's a contradiction, we know the position is a Line
4. If neither results in a contradiction, take the intersection of the two resulting grids


3 3
2 2 0
2 2 2
2 0
3 2 3




3 3
2 2 0
2 2 2
2 0
3 2 3


3 3
2 2 0
2 2 2
2 0
3 2 3

Recursive Guessing

If single guesses don't result in anything, we can nest our guesses. New information from nested guesses propagates out to the canonical grid.

Recursive Guessing

If single guesses don't result in anything, we can nest our guesses. New information from nested guesses propagates out to the canonical grid.

We call n nested guesses a "depth n guess".

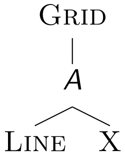
Example of a depth 2 guess

GRID

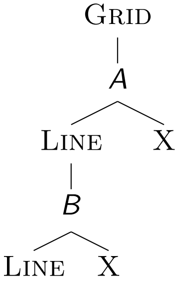
Example of a depth 2 guess

$$\begin{array}{c} \textsc{Grid} \\ | \\ A \end{array}$$

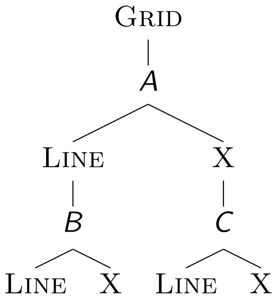
Example of a depth 2 guess



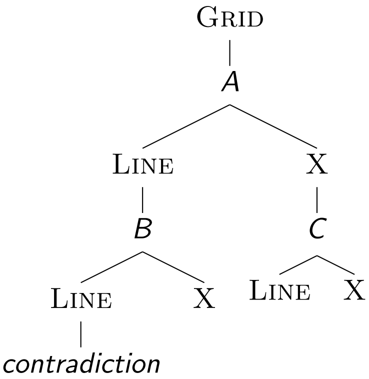
Example of a depth 2 guess



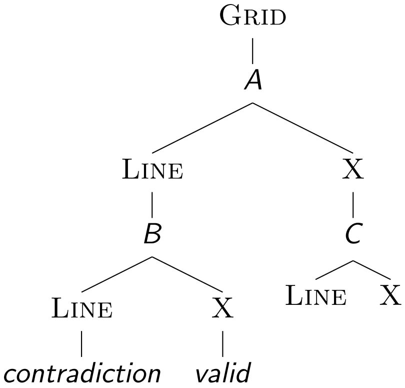
Example of a depth 2 guess



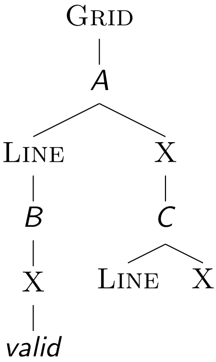
Example of a depth 2 guess



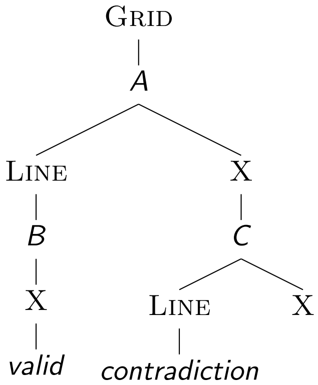
Example of a depth 2 guess



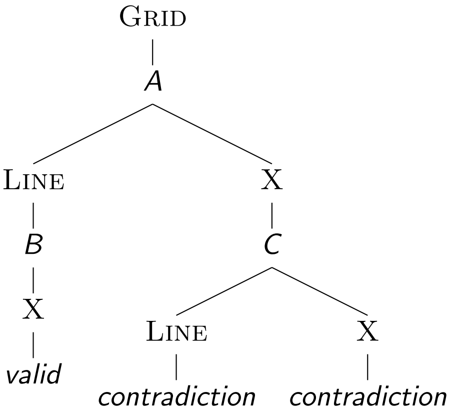
Example of a depth 2 guess



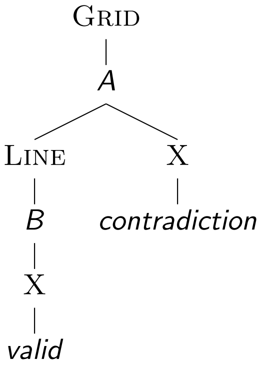
Example of a depth 2 guess



Example of a depth 2 guess



Example of a depth 2 guess



Example of a depth 2 guess

$$
\begin{array}{c}
\textsc{Grid} \\
| \\
A \\
| \\
\textsc{Line} \\
| \\
B \\
| \\
\mathrm{X}
\end{array}
$$

Repeated guessing algorithm

- If at any point new information is found, restart algorithm

Repeated guessing algorithm

- If at any point new information is found, restart algorithm
- Run deterministic rules

Repeated guessing algorithm

- If at any point new information is found, restart algorithm
- Run deterministic rules
- Run every possible guess (depth 1 guessing)

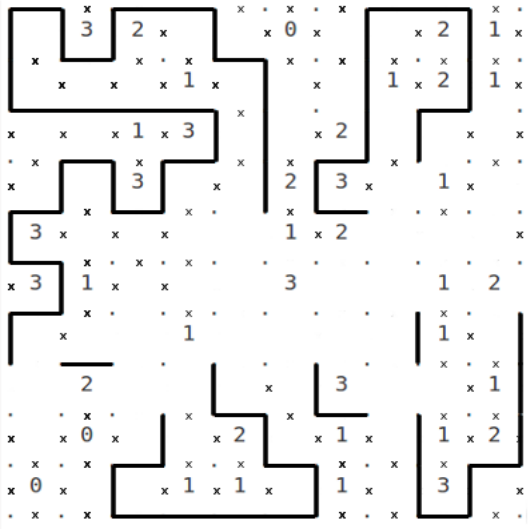
Repeated guessing algorithm

- If at any point new information is found, restart algorithm
- Run deterministic rules
- Run every possible guess (depth 1 guessing)
- Run every possible guess, and within each guess, make all possible guesses (depth 2 guessing)

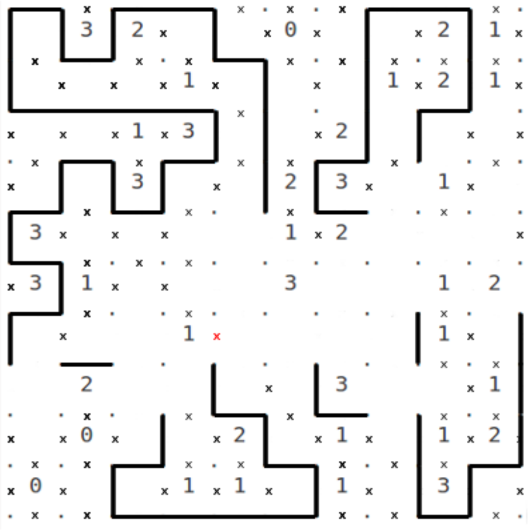
Repeated guessing algorithm

- If at any point new information is found, restart algorithm
- Run deterministic rules
- Run every possible guess (depth 1 guessing)
- Run every possible guess, and within each guess, make all possible guesses (depth 2 guessing)
- ...

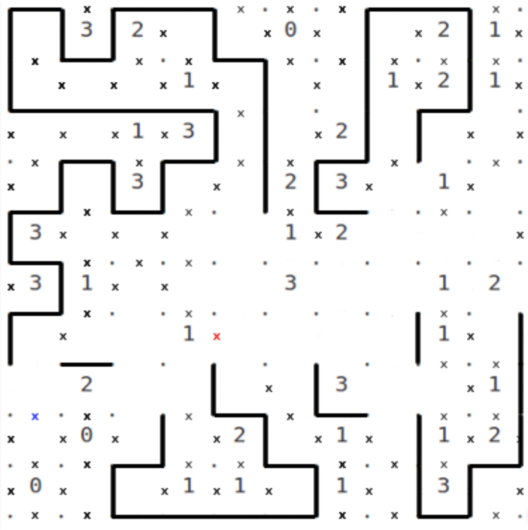
Solving a Slitherlink Puzzle



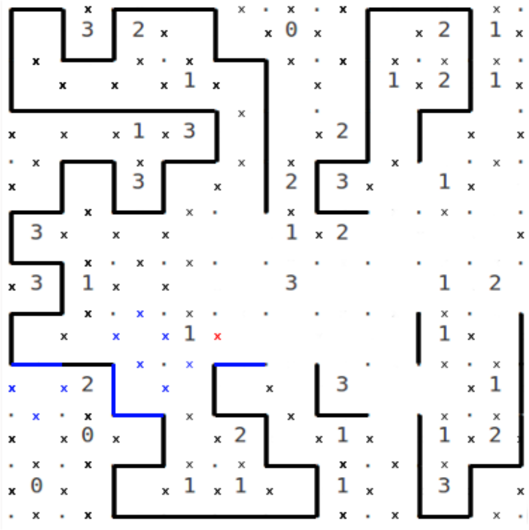
Solving a Slitherlink Puzzle



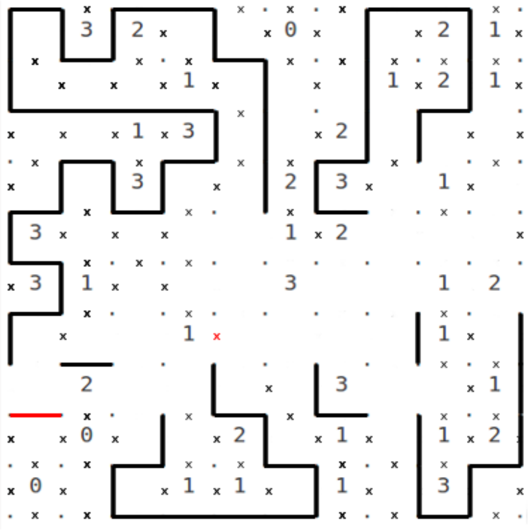
Solving a Slitherlink Puzzle



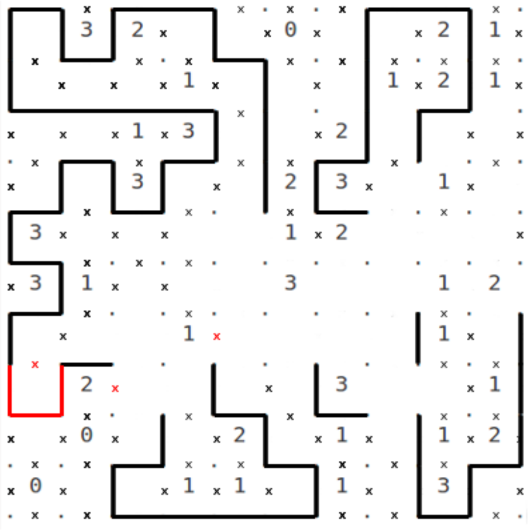
Solving a Slitherlink Puzzle



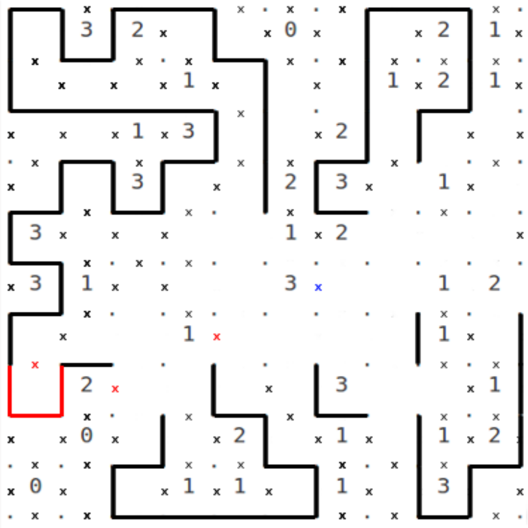
Solving a Slitherlink Puzzle



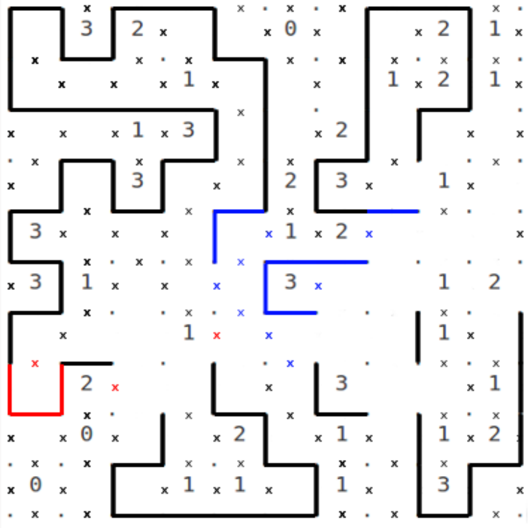
Solving a Slitherlink Puzzle



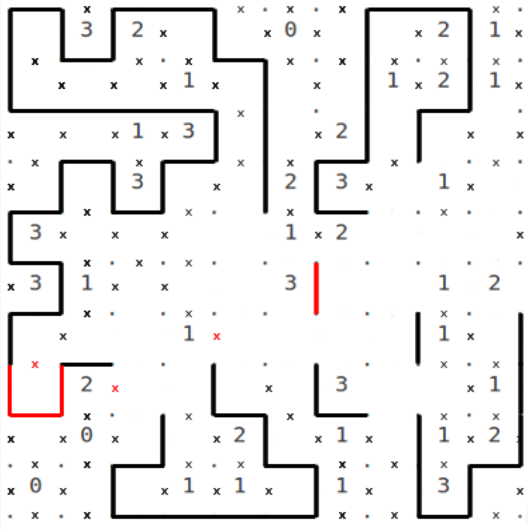
Solving a Slitherlink Puzzle



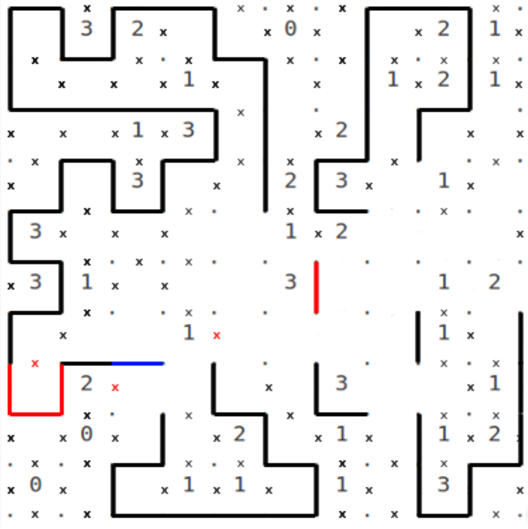
Solving a Slitherlink Puzzle



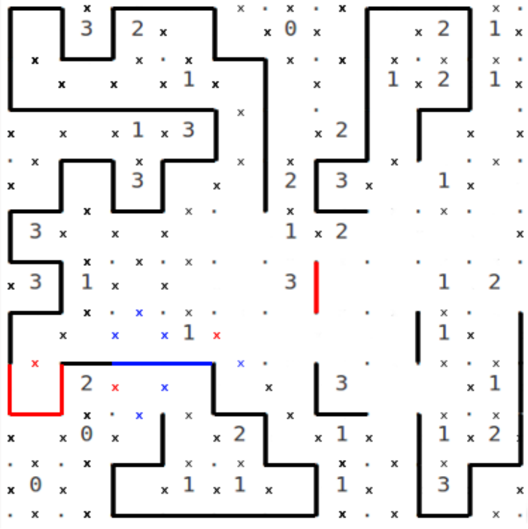
Solving a Slitherlink Puzzle



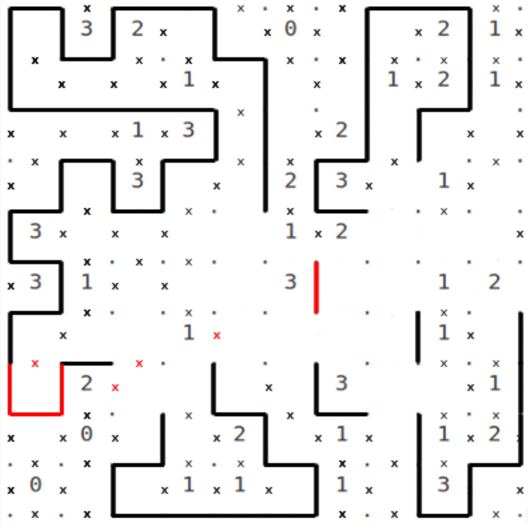
Solving a Slitherlink Puzzle



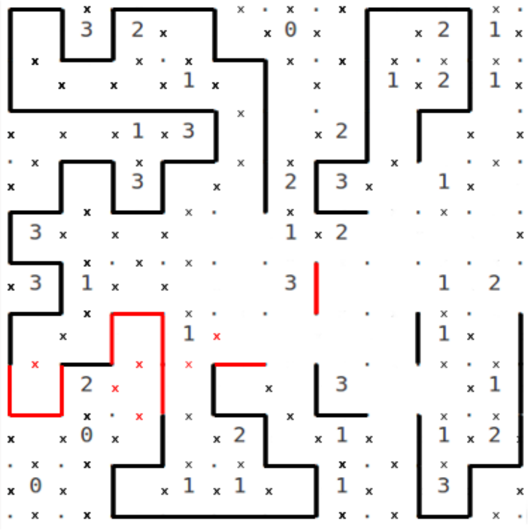
Solving a Slitherlink Puzzle



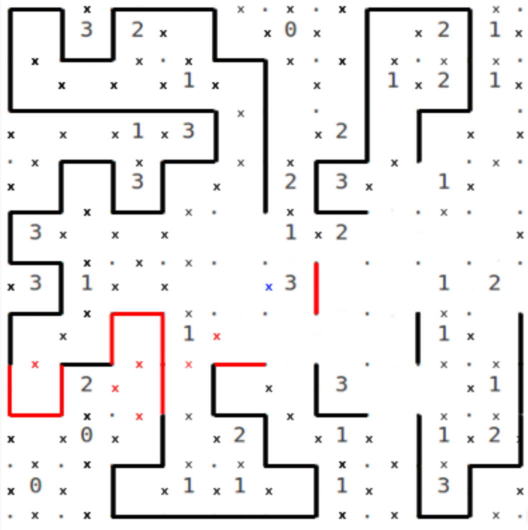
Solving a Slitherlink Puzzle



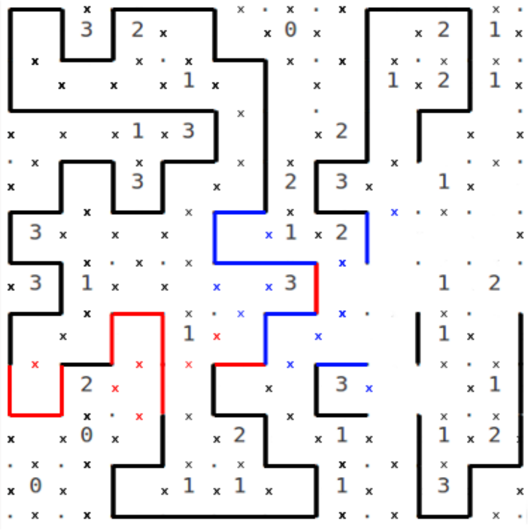
Solving a Slitherlink Puzzle



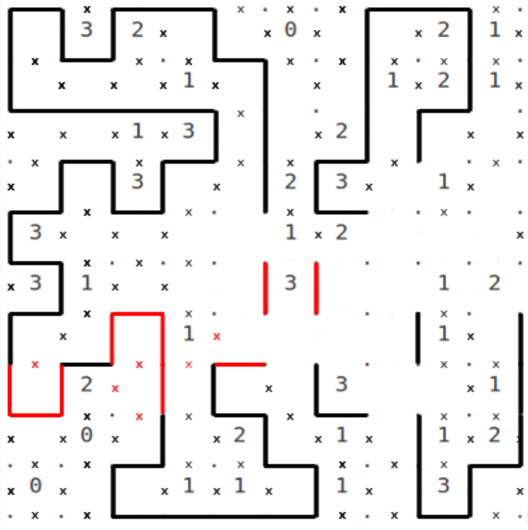
Solving a Slitherlink Puzzle



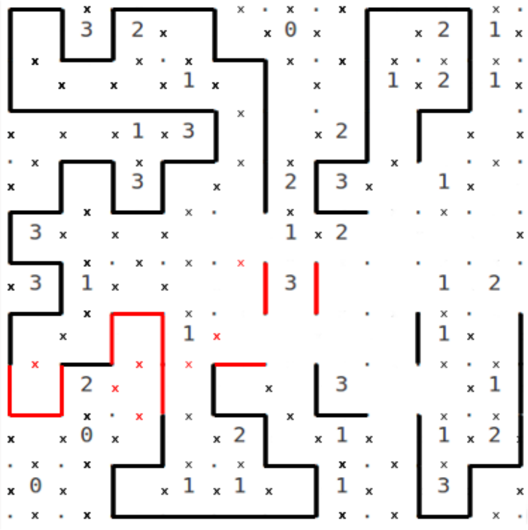
Solving a Slitherlink Puzzle



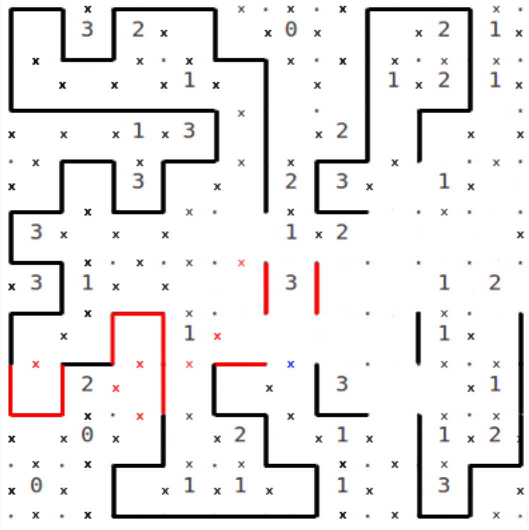
Solving a Slitherlink Puzzle



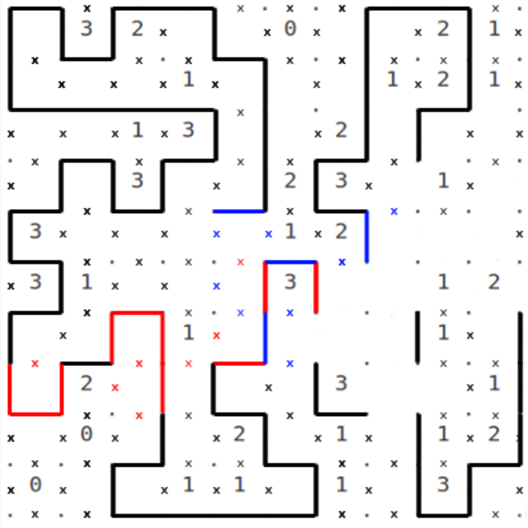
Solving a Slitherlink Puzzle



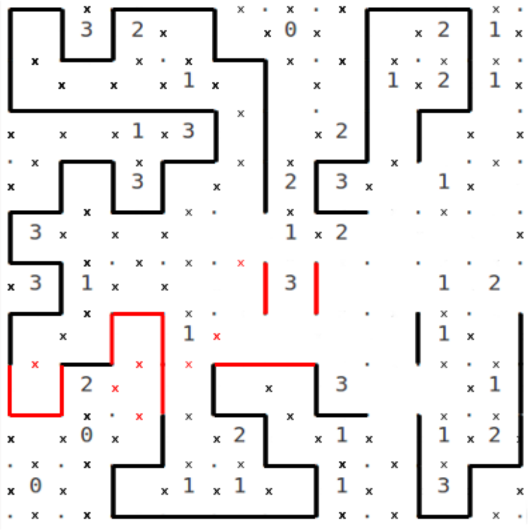
Solving a Slitherlink Puzzle



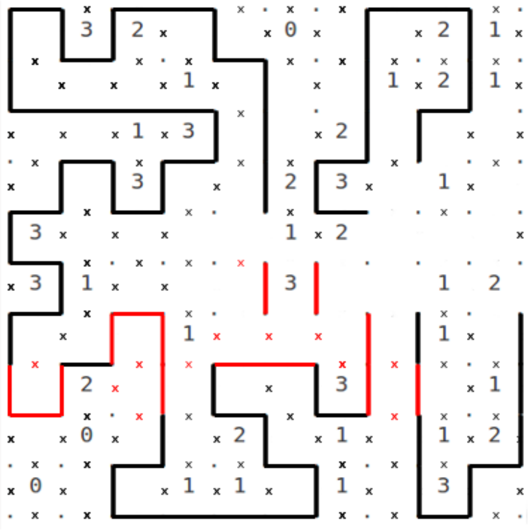
Solving a Slitherlink Puzzle



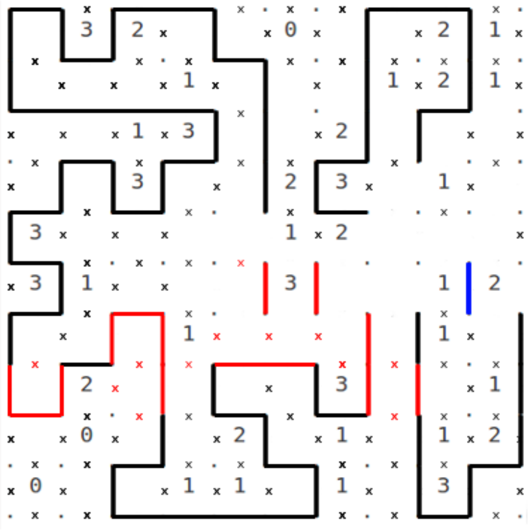
Solving a Slitherlink Puzzle



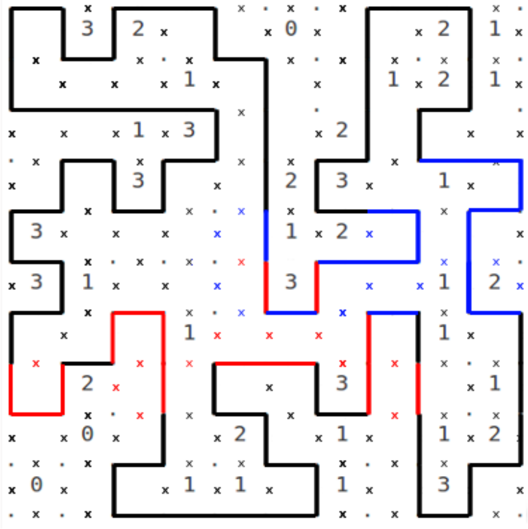
Solving a Slitherlink Puzzle



Solving a Slitherlink Puzzle



Solving a Slitherlink Puzzle



Checking for Multiple Solutions

For every guess we made to get to the solution:

Checking for Multiple Solutions

For every guess we made to get to the solution:

- Go back to the state of the grid before the guess was made, and solve the corresponding grid with the opposite guess at that same spot:

Checking for Multiple Solutions

For every guess we made to get to the solution:

- Go back to the state of the grid before the guess was made, and solve the corresponding grid with the opposite guess at that same spot:
  - If the opposite guess eventually leads to a contradiction, we know that the original guess has to be true (given all previous guesses). Continue to check other guesses.

Checking for Multiple Solutions

For every guess we made to get to the solution:

- Go back to the state of the grid before the guess was made, and solve the corresponding grid with the opposite guess at that same spot:
  - If the opposite guess eventually leads to a contradiction, we know that the original guess has to be true (given all previous guesses). Continue to check other guesses.
  - If the opposite guess eventually leads to one or more solutions, then we know that this grid has more than one solution.

Checking for Multiple Solutions

For every guess we made to get to the solution:

- Go back to the state of the grid before the guess was made, and solve the corresponding grid with the opposite guess at that same spot:
  - If the opposite guess eventually leads to a contradiction, we know that the original guess has to be true (given all previous guesses). Continue to check other guesses.
  - If the opposite guess eventually leads to one or more solutions, then we know that this grid has more than one solution.

If the opposite of every guess we had to make leads to a contradiction, then we know that the original solution we found is the only one.

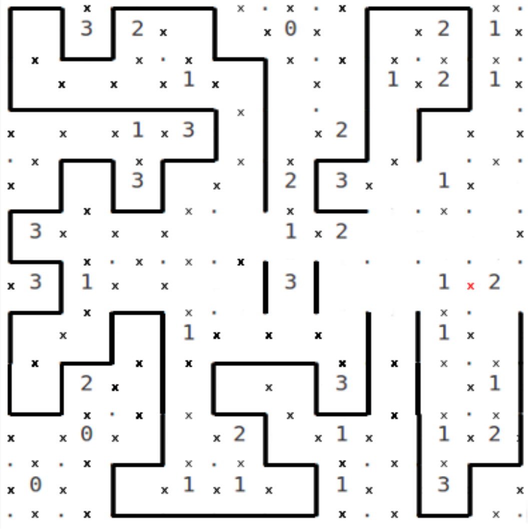
Checking for Multiple Solutions

For every guess we made to get to the solution:

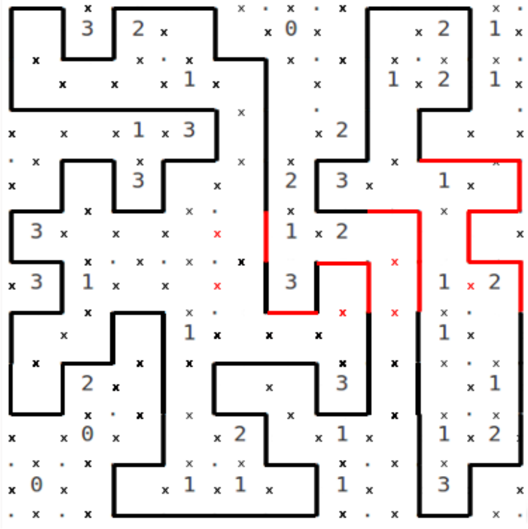
- Go back to the state of the grid before the guess was made, and solve the corresponding grid with the opposite guess at that same spot:
  - If the opposite guess eventually leads to a contradiction, we know that the original guess has to be true (given all previous guesses). Continue to check other guesses.
  - If the opposite guess eventually leads to one or more solutions, then we know that this grid has more than one solution.

If the opposite of every guess we had to make leads to a contradiction, then we know that the original solution we found is the only one.

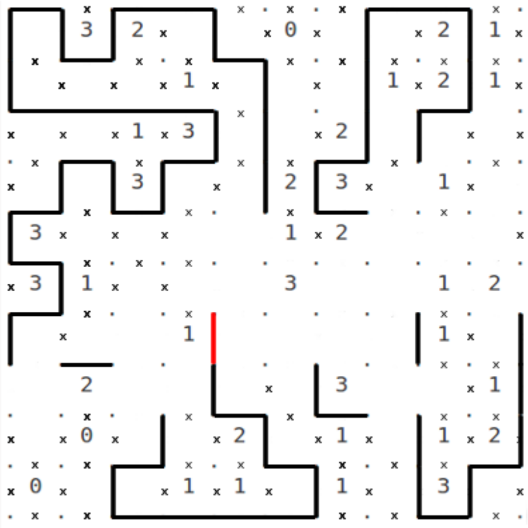
Checking for Multiple Solutions



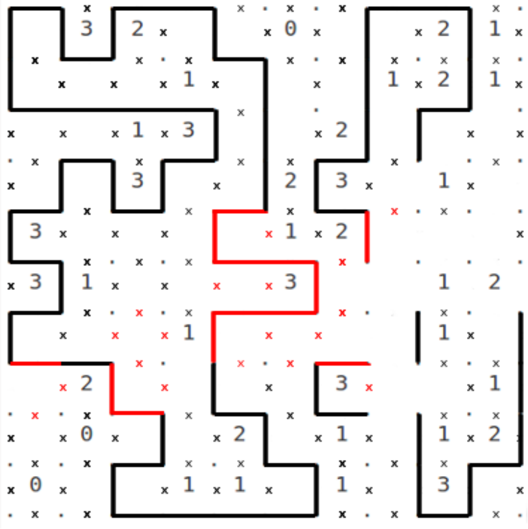
Checking for Multiple Solutions



Checking for Multiple Solutions



Checking for Multiple Solutions



Time Complexity

Important details:

Time Complexity

Important details:

- For each depth, $\mathcal{O}(mn)$ new guesses, each taking $\mathcal{O}(mn)$ time to instantiate

Time Complexity

Important details:

- For each depth, $\mathcal{O}(mn)$ new guesses, each taking $\mathcal{O}(mn)$ time to instantiate
- Guessing at the max depth is by far the most important factor in runtime

Time Complexity

Important details:

- For each depth, $\mathcal{O}(mn)$ new guesses, each taking $\mathcal{O}(mn)$ time to instantiate
- Guessing at the max depth is by far the most important factor in runtime
- We can maximize this by never filling anything in at lower depths

Time Complexity

Overall runtime $\mathcal{O}((mn)^d)$ with d bounded above by $\mathcal{O}(mn)$

Table 2: Empty grid completion time

Size	Max Depth	Time (sec)
3x3	0	0.001111
3x3	1	0.033743
3x3	2	1.19998
3x3	3	57.2004
3x3	4	2587.52

Empirical Results: Typical Puzzles

Table 3: Solve Times

Size	Max Depth	Solve Time
10x10	1	0.048542 seconds
10x10	1	0.385348 seconds
10x10	2	1.77571 seconds
10x10	2	3.23716 seconds
10x10	3	150.824 seconds*
30x25	1	1.95466 seconds
30x25	1	2.38471 seconds
30x25	1	4.4892 seconds
40x30	1	1.97524 seconds
40x30	3	66.268 seconds*

*These puzzles were determined to have multiple solutions
Puzzles taken from $nikoli.com$, $kakuro - online.com$, and $puzzle - loop.com$.

Make a Slitherlink Puzzle

Overview

1. Make a loop
2. Fill grid with numbers
3. Remove some numbers

Making a Loop

Start with an empty $m \times n$ grid, a simple rule, and three lists:

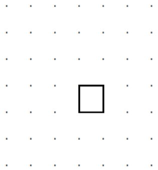
1. *available*
2. *expandable*
3. *unexpandable*

Making a Loop

Start with an empty $m \times n$ grid, a simple rule, and three lists:

1. *available*
2. *expandable*
3. *unexpandable*

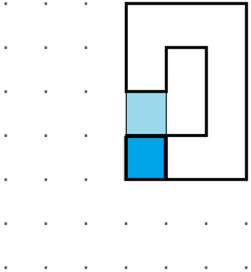
Start with every location in the grid in *available* but none in *unexpandable* and *expandable*. Then, add one random location to *expandable* and remove it from *available*.



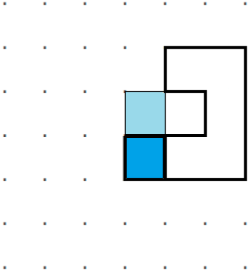
Bad Stuff

Rule: When expanding from a location, *cur*, in *expandable* to an adjacent location in *available* , make sure that adding *pos* to *expandable* doesn't cause any bad stuff

Opposite:



Opposite kitty-corners:



If *opposite* or either of the *opposite kitty-corners* are in *expandable*, then do not add *pos* to the loop.

Making a Loop cont.

1. Choose a location, *cur* in *expandable* at random

Making a Loop cont.

1. Choose a location, *cur* in *expandable* at random
2. Look at neighbors to see if and where the loop can expand from *cur*.

Making a Loop cont.

1. Choose a location, *cur* in *expandable* at random
2. Look at neighbors to see if and where the loop can expand from *cur*.
    2.1 If a neighbor is in *available* and it wasn't a valid neighbor, remove it from *available*

Making a Loop cont.

1. Choose a location, *cur* in *expandable* at random
2. Look at neighbors to see if and where the loop can expand from *cur*.
   2.1 If a neighbor is in *available* and it wasn't a valid neighbor, remove it from *available*
   2.2 If there are no valid neighbors, add *cur* to *unexpandable* and remove it from *expandable*

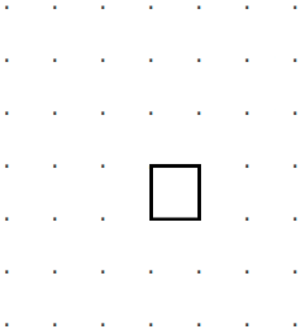
Making a Loop cont.

1. Choose a location, *cur* in *expandable* at random
2. Look at neighbors to see if and where the loop can expand from *cur*.
   2.1 If a neighbor is in *available* and it wasn't a valid neighbor, remove it from *available*
   2.2 If there are no valid neighbors, add *cur* to *unexpandable* and remove it from *expandable*
   2.3 Otherwise, randomly choose an valid neighbor to add to add to *expandable* and take out of *available*

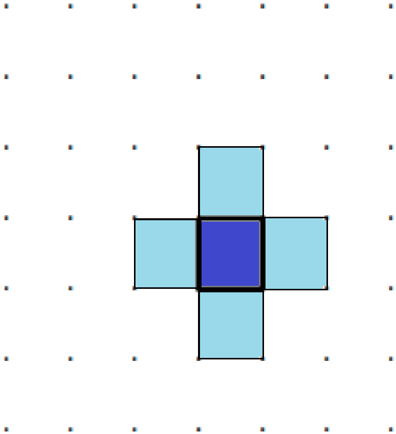
Making a Loop cont.

1. Choose a location, *cur* in *expandable* at random
2. Look at neighbors to see if and where the loop can expand from *cur*.
   2.1 If a neighbor is in *available* and it wasn't a valid neighbor, remove it from *available*
   2.2 If there are no valid neighbors, add *cur* to *unexpandable* and remove it from *expandable*
   2.3 Otherwise, randomly choose an valid neighbor to add to add to *expandable* and take out of *available*
3. repeat until there are no locations in *expandable*

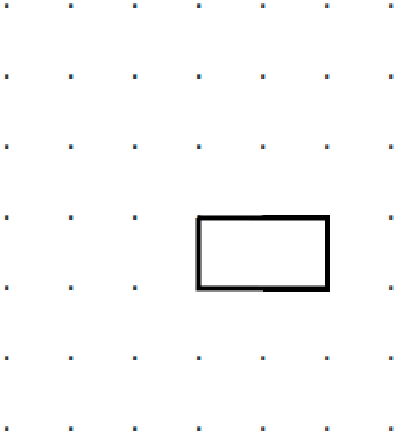
Making a Loop Example



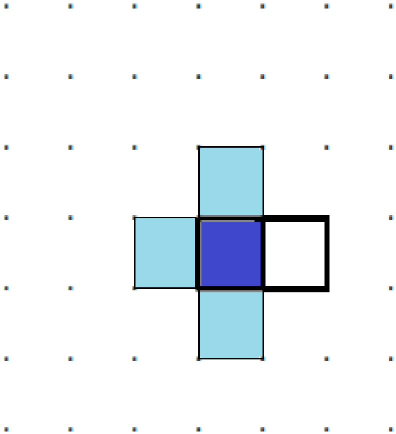
Making a Loop Example



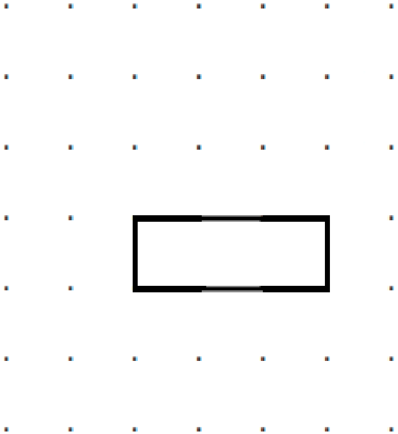
Making a Loop Example



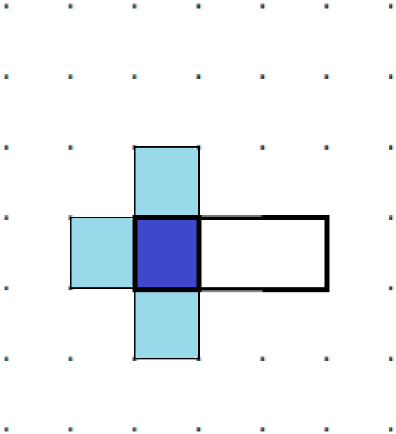
Making a Loop Example



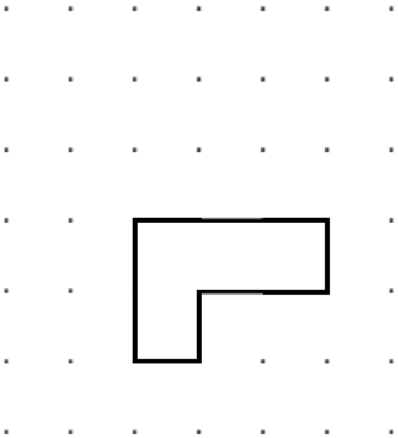
Making a Loop Example



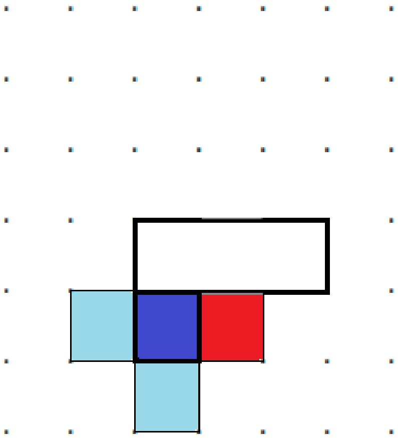
Making a Loop Example



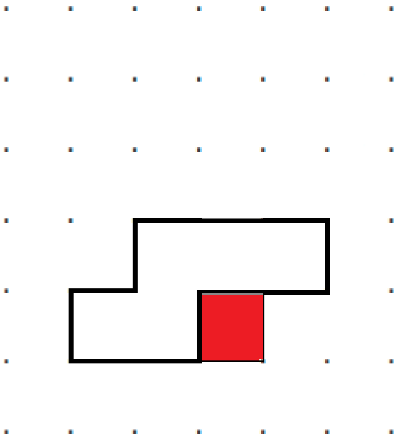
Making a Loop Example



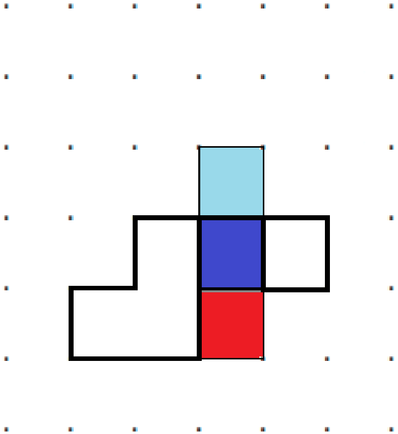
Making a Loop Example



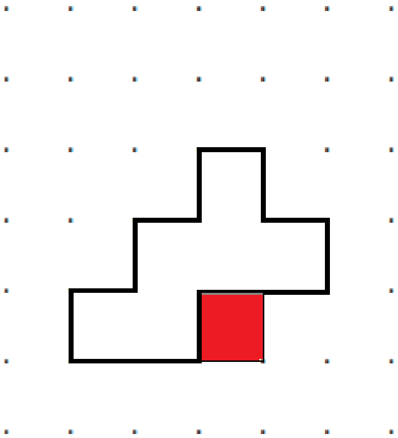
Making a Loop Example



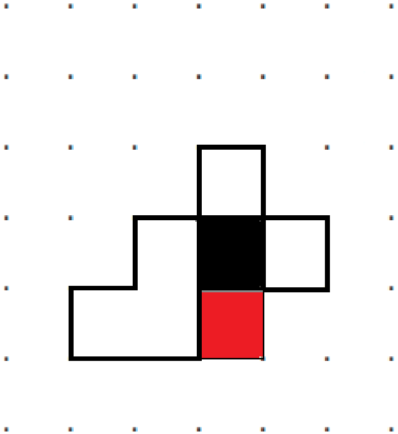
Making a Loop Example



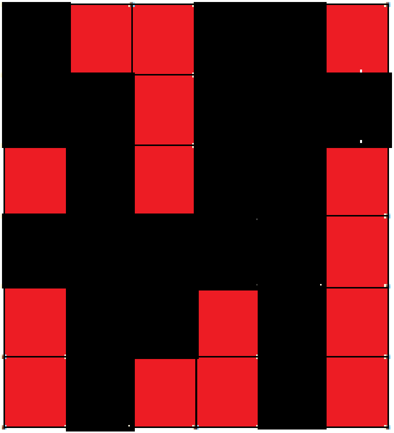
Making a Loop Example



Making a Loop Example

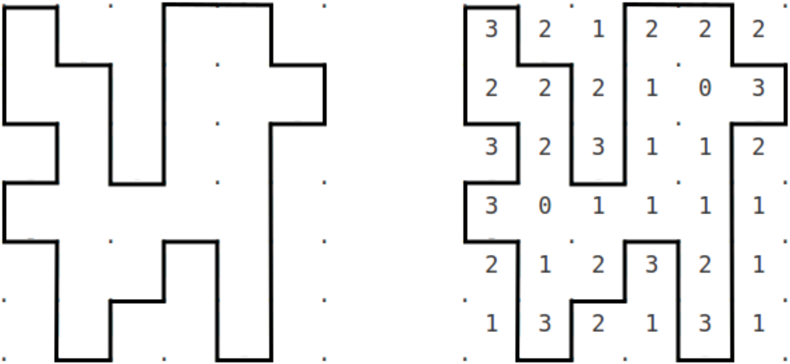


Making a Loop Example



Filling with Numbers

Surprise surprise, this is actually really easy



(sum the differences in the neighboring locations)

Removing Numbers

To make puzzles interesting, we want to remove numbers

We want to do so until we have reached a certain count

Must retain one unique solution

Removing Numbers

The Process:

Removing Numbers

The Process:

1. Pick a number from a set of eligible numbers

Removing Numbers

The Process:

1. Pick a number from a set of eligible numbers
2. Add this number to a stack of ineligible numbers

Removing Numbers

The Process:

1. Pick a number from a set of eligible numbers
2. Add this number to a stack of ineligible numbers
3. Check if eliminating would make the puzzle unsolvable

Removing Numbers

The Process:

1. Pick a number from a set of eligible numbers
2. Add this number to a stack of ineligible numbers
3. Check if eliminating would make the puzzle unsolvable
    3.1 If solvable, remove the number from both the grid and set of eligible numbers

Removing Numbers

The Process:

1. Pick a number from a set of eligible numbers
2. Add this number to a stack of ineligible numbers
3. Check if eliminating would make the puzzle unsolvable
    - 3.1 If solvable, remove the number from both the grid and set of eligible numbers
    - 3.2 If unsolvable, only remove the number from the set of eligible numbers

Removing Numbers

The Process:

1. Pick a number from a set of eligible numbers
2. Add this number to a stack of ineligible numbers
3. Check if eliminating would make the puzzle unsolvable
   3.1 If solvable, remove the number from both the grid and set of eligible numbers
   3.2 If unsolvable, only remove the number from the set of eligible numbers
4. Repeat until set of eligible numbers is empty

Removing Numbers cont.

Once set of eligible numbers is empty:

Removing Numbers cont.

Once set of eligible numbers is empty:

1. Pop numbers off ineligible stack

Removing Numbers cont.

Once set of eligible numbers is empty:

1. Pop numbers off ineligible stack
2. Place each back into the set of eligible numbers

Removing Numbers cont.

Once set of eligible numbers is empty:

1. Pop numbers off ineligible stack
2. Place each back into the set of eligible numbers
3. Do so until most recently eliminated number is found

Removing Numbers cont.

Once set of eligible numbers is empty:

1. Pop numbers off ineligible stack
2. Place each back into the set of eligible numbers
3. Do so until most recently eliminated number is found
4. Keep eliminated in the ineligible stack, but place back into grid

Removing Numbers cont.

Once set of eligible numbers is empty:

1. Pop numbers off ineligible stack
2. Place each back into the set of eligible numbers
3. Do so until most recently eliminated number is found
4. Keep eliminated in the ineligible stack, but place back into grid

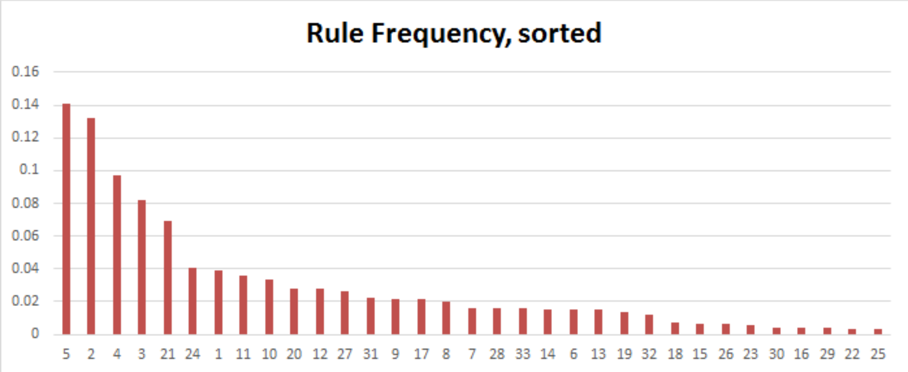
Repeat removing numbers until desired count is reached

Removing Numbers cont.

It's too hard!

Improvements

1. Data
2. Rule set
3. Balancing


Rule Frequency, sorted
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
5 2 4 3 21 24 1 11 10 20 12 27 31 9 17 8 7 28 33 14 6 13 19 32 18 15 26 23 30 16 29 22 25

Ruleset Limitation

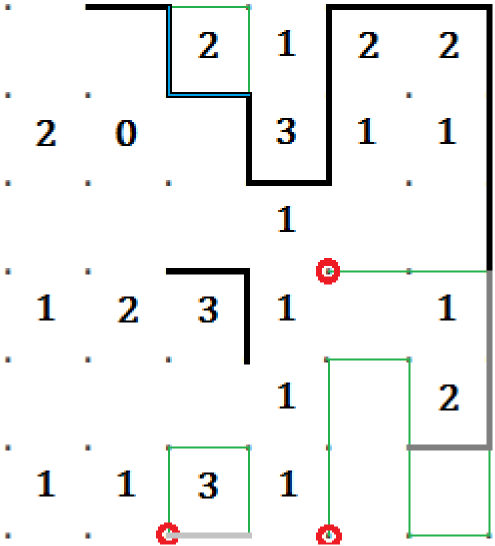
As a result, we created two subsets of rules:

- easy: the rules with a greater than 5% occurrence rate
- hard: the rules with a greater than 1% occurrence rate

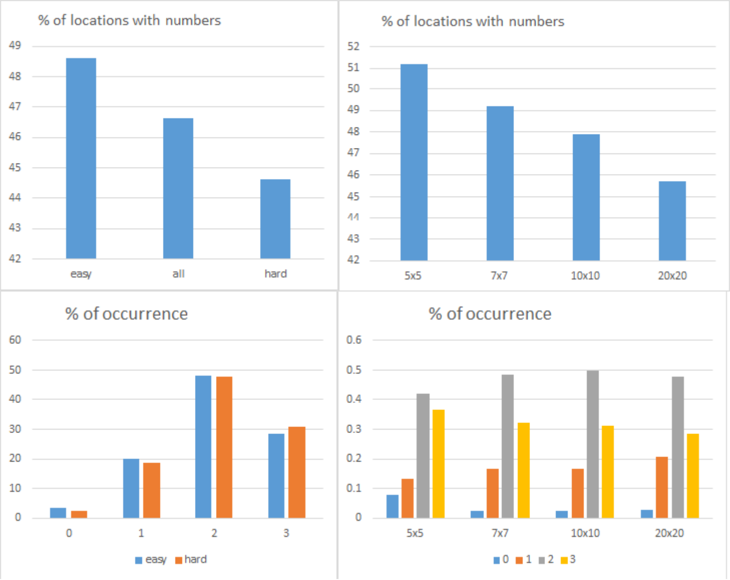
An ‘Easy’ Puzzle

		2	1	2	2
2	0		3	1	1
			1		
1	2	3	1		1
			1		2
1	1	3	1		

A Quick Attempt at the ‘Easy’ Puzzle



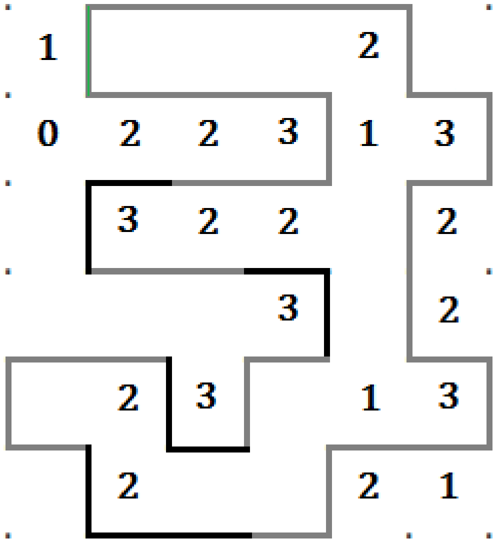
Balancing the Numbers



A Balanced Easy Puzzle

1				2	
0	2	2	3	1	3
	3	2	2		2
			3		2
	2	3		1	3
	2			2	1

A Quick Attempt at the ‘Easy’ Puzzle



Time Complexity

Important details

- The Solver is run on the order of mn times.
- Each time the solver is run, it happends with a maximum depth of one guess which has on the order of $\mathcal{O}((mn)^2)$ time.
- Therefore, the generator run in the order of $\mathcal{O}((mn)^3)$ time.

References

Conceptis Puzzles Slitherlink Techniques
On the NP-completeness of the Slither Link Puzzle Takayuki YATO
Finding All Solutions and Instances of Numberlink and Slitherlink by ZDDs. Ryo Yoshinaka, Toshiki Saitoh, Jun Kawahara, Koji Tsuruma, Hiroaki Iwashita and Shin-ichi Minato.

A rule-based approach to the puzzle of Slither Link. Stefan Herting.

Puzzles and Games: A Mathematical Modeling Approach. Tony Hürlimann, 2015

Solving logical puzzles using mathematical models. KVIS Susanti, S Lukas.

Thank you

Questions?