Tecnológico de Monterrey

Artificial Intelligence

Ana María González Montoya
Raúl García Furió

# Challenge

**Justification**
We decided to implement a sudoku solver because one day one of us saw a video on youtube about a lego robot that solve a sudoku by recognizing the numbers in a sheet of paper and then it resolve the game writing the numbers. We decided to implement a solver with digit recognition.

**Sudoku:**

The sudoku solver was implemented in Java.

Analysis:

We implemented two versions of a Sudoku solver. The first one use a Depth First Search to solve the Sudoku.

This solution take the first cell and assign a possible value and go to the next cell, if a cell has no more possible values it return to the previous cell and try the next value for that cell. This process continues till the algorithm reach the answer or it discovers there are no more possible answers.

This process is inefficient because in the worst case scenario it will take 600 million steps to reach the answer. And this is not like we solve a sudoku.

When we solve a sudoku the first action we take is to search the best cell to start, the one with the less possible numbers, this help us to reduce the number of possibilities to the next cells. We follow this process till we finish the puzzle

So in order to produce this behaviour we used the Depth First Search with a technique called Constraint Propagation, or Constraint Satisfaction. This technique consider the rules of the Sudoku, so we can detect the error before it happen. With the previous one, we detect the error once we reach the cell and in some cases it will need to backtrack all the sudoku in order to have the right value.

The final algorithm will be as follow:

1. Get the possibles values for each empty cell
2. Sort them based in the number of possible values they could have(from lower to higher)
3. Take the one with the lowest possible answers
4. Iterate the values it could take
5. Repeat the process for the rest of the empty cells

Constraints:

- If a empty cell has no more possible values, try the next value for the actual cell

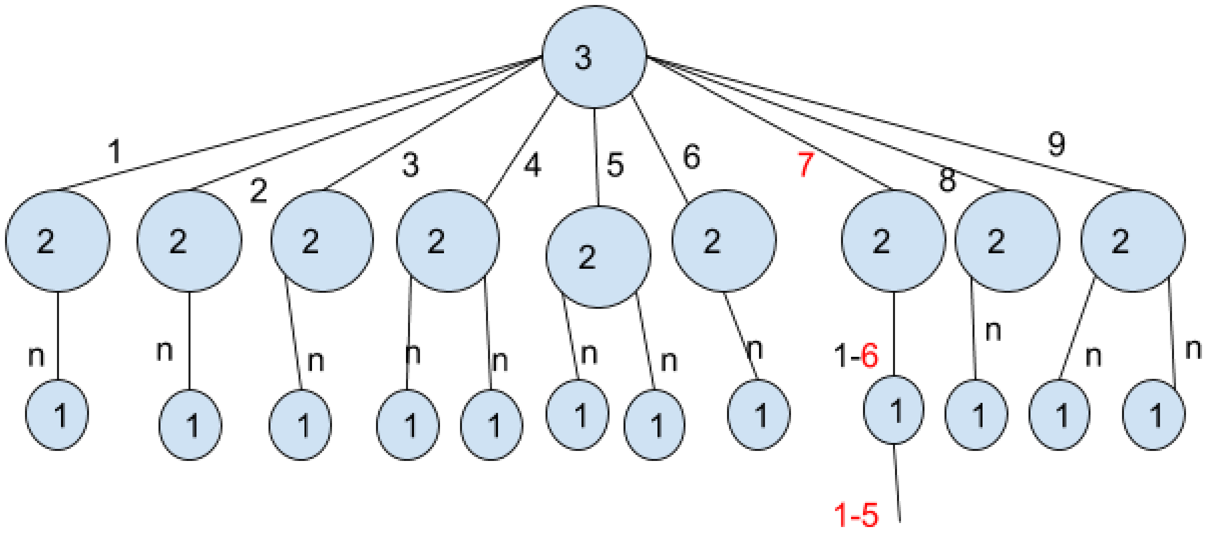
This is a best approach in order to simulate how we solve a sudoku
As we can notice a sort step occurs in the new algorithm. The algorithm with constraint propagation can be executed without this step and it will follow the empty cells in order. But to the algorithm to be more efficient this step help us to simplify the tree that our algorithm will be searching.

This happen because taking the “simpler” cells will reduce the complexity of the rest of the cells by reducing the number of possible number that cell could take. And the search will be easier, more intuitive and will take less recursive calls.

Let's see a little example:

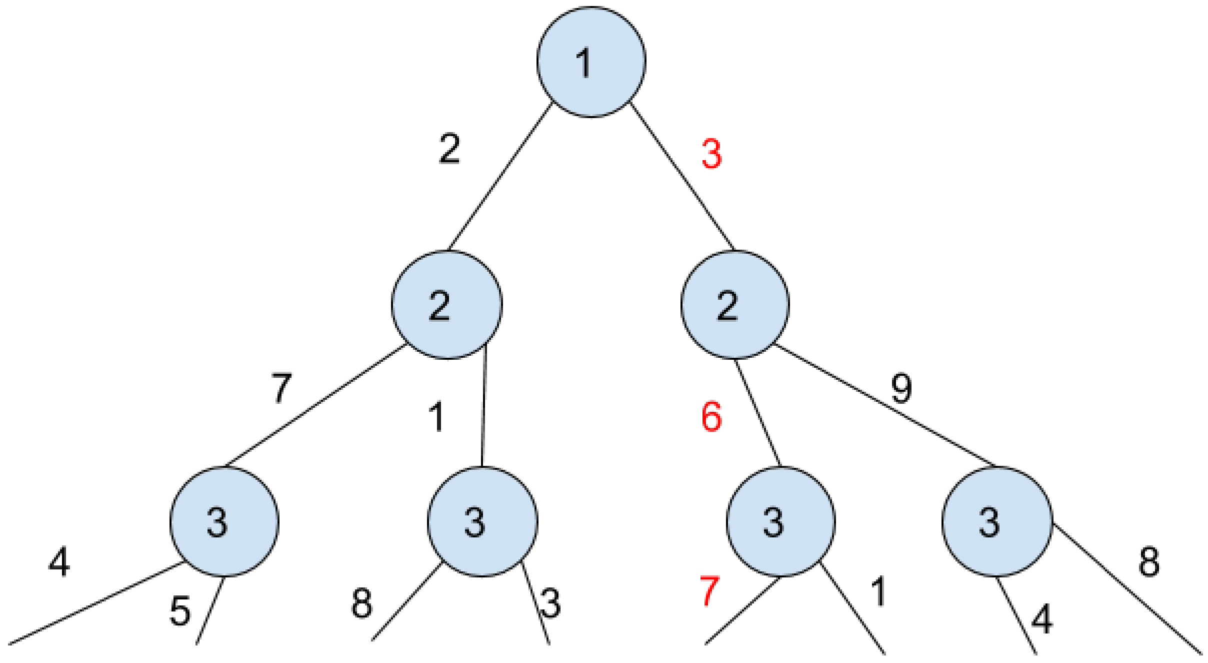
For the nodes(cells) 1, 2, 3 we want the result to be:
Node 1 = 3, Node 2 = 6,  Node 3= 7

If we start with the most difficult node first the tree will be something like this



We will need to check at least 6 possible values of node 3 because we haven’t eliminate possible values from it. and the same occur with the node 2. This tree will expand a lot, so it is more difficult to compute and it will take longer
(We can even draw the finished tree for this easy example)

Otherwise starting with the easy nodes first will produce a tree similar to this



With the constraint propagation technique. the most difficult node will be the last so when we reach that node it will have less possible values to take. And the tree will not expand too much

With this technique our algorithm was able so love harder sudokus in few seconds. While the first approach takes more than 20 minutes for easy sudokus.

**Digit Recognition**
Implemented in java.

For the digit recognition we implement a multiple neural network that consist of an input node that receive 10 inputs, a hidden layer with 36 nodes and an output layer.