Universitá
Ca'Foscari
Venezia

Master course

in Computer Science

Artificial Intelligence: Knowledge Representation and Planning

-

Assignment 1

Studente
Francesco Bruno
Matricola 875812

Anno Accademico
2021 / 2022

Contents

Chapter 1

Sudoku

1.1 Introduction

Sudoku is a 1984 Japan game that needs a bit of logic in order to solve it. It is composed by a $9x9$ matrix, so 81 cells divided by rows, columns, boxes[1] and filled with digits (not all cells have a value and the value can be any number between 1 and 9).
The goal of the game is to fill with a proper digit the cells without an already setted value, thus to have as a result a matrix with no empty cells.
Rules are very easy, if a box, row or column has already a digit X inside it, then all other cells inside that specific row, column or box can't have the same value X.

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

Figure 1.1: *Example of Sudoku*

[1] Each box contains exactly 9 cells and are well defined.

Chapter 2

Requirements

This assignment requires to make a sudoku solver using:

1. Constraint satisfaction approach;
2. Relaxation labelling approach;

At the end of its implementations there must be a comparison between both techniques where advantages and disadvantages of each one are found.
The solver takes as input a *.txt* file where inside it its stored a matrix where "." symbol denotes an empty cell without an assigned value, " " denotes the vertical space between each box of the sudoku, other values we can find are obviously the digits (between 1 and 9). At the end of the execution of the solver, the sudoku will be solved and returned

Chapter 3

Constraint Satisfaction Problem

3.1 Definition

Sudoku can be interpreted as a Constraint Satisfaction Problem (CSP), thus we can define:

- A set of variables;
- A domain set where its content are the possible values of the variables;
- A set of constraints;

In our case variables are the cells of the sudoku and the domain of each one contains the values cells can have (at the beginning between 1 and 9), constraints are divided into two categories, *direct* and *indirect*.

3.2 Constraints

3.2.1 Direct constraints

Direct constraints are basically a consequence of the application of games rules, thus:

- *Row constraint*: each row must have only different digits for each cell;
- *Column constraint*: each column must have only different digits for each cell;
- *Box constraint*:each box must have only different digits for each cell;

3.2.2 Indirect constraints

Indirect constrains tell us that each digit must appear in each row, column, and box, thus a list of techniques used inside the solver are defined in order to help the solver to reduce the domain of each variable.

3.2.2.1 Lone Single

A *Lone Single* happens when a cell can have only a value, thus that cell must be filled with that specific value and all other cells inside the same row, column and box must delete that specific value from the possible choices of that cell.

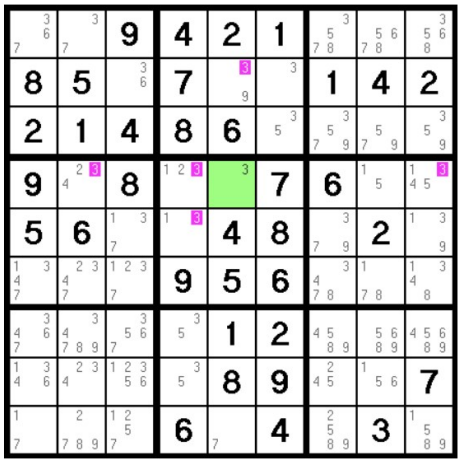

Figure 3.1: *Image taken from* ***https://www.learn-sudoku.com/lone-singles.html***

By watching Figure 3.1 we can notice how the green cell can have only value 3, so other cells in the same row, column and box must delete the choice 3 from their possible values.

3.2.2.2 Hidden Single

A *Hidden Single* happens when a cell have multiple choices but one of these values can be used only in that cell in the entire row, column or block.

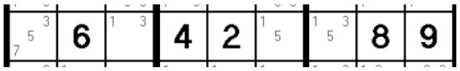

Figure 3.2: *Image taken from* ***https://www.learn-sudoku.com/hidden-single.html***

By watching Figure 3.2 we can notice how the 7 can be assigned only to

the first cell of the row, thus we can assign that value to the cell and then remove it from all possible choices of the cells contained in the same box and column.

3.2.2.3 Naked Pair

A *Naked Pair* happens when two cells have exactly two choices and these have same values for both cells.

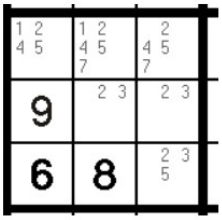

Figure 3.3: *Image taken from https://www.learn-sudoku.com/naked-pairs.html*

By watching Figure 3.3 we can notice how there are two cells with choices 2 and 3, once we've detected it, since surely both of the cells have value 2 or 3 we can remove those digits from the choices of the cells that are in the same box, row or column of the naked pair.

3.2.2.4 Hidden Pair

A *Hidden Pair* happens when two cells have at least two choices and these two are the same for both cells and no other cell inside the same row, box or column have those values.

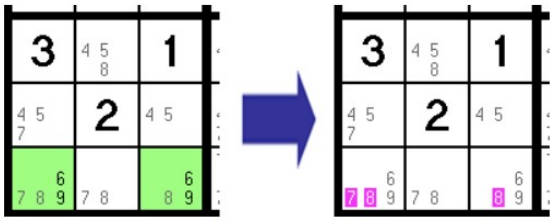

Figure 3.4: *Image taken from https://www.learn-sudoku.com/hidden-pairs.html*

By watching Figure 3.4 we can notice how there are two cells with choices 6 and 8, once we've detected it, since surely both of the cells have value 6 or 8 we can remove the other values from the choices of both cells.

Chapter 4

Constraint Propagation and Backtracking

4.1 Implementation

The aim of this technique applied to this task is to propagate constraints through recursion over a sudoku in order to simplify the domain of a cell and don't search the result of a cell inside the entire domain of the variable but only in a smaller part of it.
So, starting from a sudoku, we simplify it as much as possible by applying both categories of constraints, if the matrix is complete[1] and correct then is returned, otherwise is checked if the content of the sudoku contains some error[2] (then backtrack), if it's not complete and is correct then the algorithm chooses a *Most Constrained Variable* (a non valued cell with the least number of choices) and a random value from its choices is assigned to the cell, so the new sudoku will have at least a new value different compared to the input matrix. This algorithm is then called recursively giving to it as input the output sudoku made above and it will stop only when a correct solution is found or if there are no correct answers to the initial input matrix.

[1] A sudoku is complete iff all cells of the sudoku have an assigned value
[2] An error occurs when there are non valued cell with list of choices empty

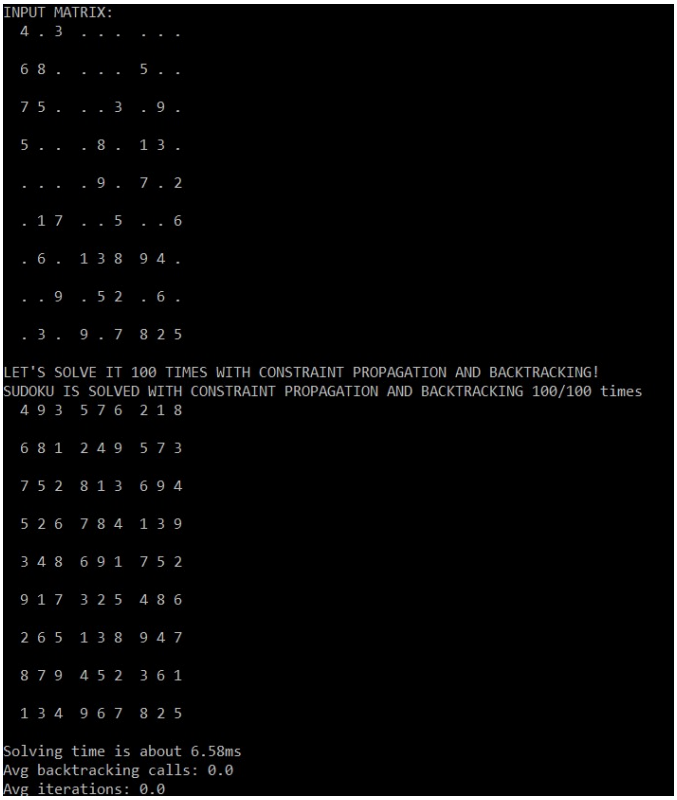

Figure 4.1: *Sudoku solved by using Constraint Propagation and Backtracking*

4.2 Performance

Performances of the solver are calculated by giving as input to the solver different sudokus of various difficulties, in particular each resolution of a single sudoku is repeated 100 times in order to have a better approximation of the results.
By considering:

- **Exec.time**: how long the solver takes to end the execution of the algorithm;
- **Avg Random Choice**: how many random choices are performed by the solver in order to obtain a simpler matrix since cells can't be simplified more;
- **Avg Backtrack calls**: how many backtracking calls are performed by the solver in order to return to the point when recursion was called, so to make a new random choice with a different value;
- **Success rate**: how many solutions returned from the algorithm are correct (given as percentage);

- **Probability updates**: how many times the probability matrix is updated;

I've obtained the following results:

Difficulty	Exec. time	Avg Random Choice	Avg Backtrack calls	Success Rate
Kids	*6.58ms*	0	0	100%
Simple	*18.32ms*	0	0	100%
Easy	*30.52ms*	0	0	100%
Medium	*45.54ms*	2	1	100%
Hard	*60.53ms*	3	1	100%
AI Escargot	*675.0ms*	63	29	100%

As we can read, at an higher difficulty of the sudoku corresponds an higher execution time, the maximum value is obtained by solving AI Escargot sudoku (the most difficult known!), also success rate is for all categories equals to 100%. Another thing we can observe is that at an higher difficulty corresponds also an higher possibility to backtrack and at the same time to choose a random value from the choices of a cell in order to propagate the matrix with a new value fixed.
In conclusion constraing propagation and backtracking solver is pretty good for solving sudokus because it has a 100% success rate and all execution are made within less than one second, even the hardest sudoku ever!

Chapter 5

Relaxation Labelling

5.1 Implementation

When we talk about Relaxation Labelling we are talking of a problem where its goal is to properly assign an object to a label, in our case we assume:

- A set $O = \{o_1, o_2, ..., o_n\}$ of object (cells), where $n = 81$;
- A set $L = \{l_1, l_2, ..., l_m\}$ of labels, where $m = 9$;

As I told above, the aim of the algorithm is to properly assign a correct value (label) to each cell (object) so to obtain a valid solution according to the sudoku rules.
In order to do it, each cell has initially assigned to itself an array of $m = 9$ elements:

$p_i^{(0)}(\lambda) = \{p_i^{(0)}(1), p_i^{(0)}(2), ..., p_i^{(0)}(m)\}$

where $o_i \in O$, $p_i^{(0)}(\lambda) \geq 0$ and $\sum_\lambda p_i^{(0)}(\lambda) = 1$

Each element of the array is the probability of that cell is labelled as λ. In addition, a matrix of compatibility coefficients is used and each cell of this matrix contains a coefficient that measures the strength compatibility between cells.
So basically, the relaxation labelling solver takes as input an initial array of probabilities (defined above) and for all cells of the sudoku it updates the array according to the matrix of compatibility coefficients until a certain convergence point is reached, from now on the solver stops the updates of probabilities and it assigns for each non valued cell a value according to the most likely number (the choice with higher probability).

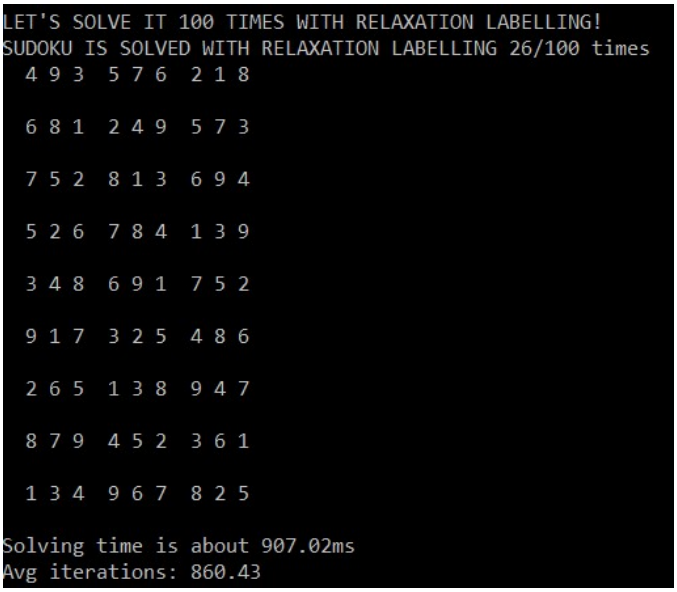

Figure 5.1: *Same sudoku solved by using Relaxation Labelling*

5.2 Performance

Performances of the solver are calculated by giving as input to the solver different sudokus of various difficulties, in particular each resolution of a single sudoku is repeated 100 times in order to have a better approximation of the results.
By considering:

- **Exec.time**: how long the solver takes to end the execution of the algorithm;
- **Success rate**: how many solutions returned from the algorithm are correct (given as percentage);
- **Probability updates**: how many times the probability matrix is updated;

I've obtained the following performances:

Difficulty	Exec. time	Success Rate	Probability updates
Kids	*907.02ms*	26%	860.43
Simple	*1029.23ms*	8%	1080.48
Easy	*999ms*	4%	1051
Medium	*1084ms*	1%	1166.5
Hard	*1018ms*	0%	1068
AI Escargot	*1707ms*	0%	2170

As we can see relaxation labelling solver isn't as good as the constraint propagation and backtracking one, indeed it has obtained very poor results, with the highest success rate of only 26% for kids sudokus, other more difficult categories are even worse!
If we consider the execution time then it's still worse than the other solver because the time needed for obtain a result is way bigger than before and it has a huge probability to return a wrong result.

Chapter 6

Conclusion

After having used and analyzed both techniques, we have noticed how relaxation labelling solver has obtained very bad results in comparison to constraint propagation and backtracking one, this is also caused becacuse of the reason why algorithm stop.
So, if the constraint propagation and backtracking algorithm stops is because a solution is reached, the relaxation labelling one ends only because a point is reached (the convergerce point) but this doesn't mean that the sudoku made is certainly correct, this can lead to uncorrect solutions as we've seen so far.
Summing up what it has been told until now:

- *Completeness*: Constraint Propagation and Backtracking solver is a complete solution since it has solved and it solves every sudoku it takes as input. Relaxation Labelling instead isn't complete since it solves only a few sudokus (only the easiest ones).
- *Execution Time*: Constraint Propagation and Backtracking solver is much faster than the Relaxation Labelling one, for all sudokus and categories. The first one it solves sudokus in less than one second, the other one takes most of the time more than one second;
- *Space Complexity*: Performing Relaxation Labelling problem requires much more space than Constraint Propagation and Backtracking since in addition to the Sudoku matrix and the choices for each cell, there are also the probability matrix and the compatibility coefficients matrix;

In conclusion Relaxation Labelling techinque gives us worse results from every point of view in comparison to the Constraint Propagation and Backtracking one, so this last one is way better in order to solve Sudokus problems and Relaxation Labelling should be used for solving other types of problems.

List of Figures