Knowledge Representation - Assignment 2 Report

Minh Ngo 10897402[1], Casper Thuis 10341943[2]

October 12, 2015

[1] University of Amsterdam
MSc. Artificial Intellegence
minh.ngole@student.uva.nl

[2] University of Amsterdam
MSc. Artificial Intelligence
casper.thuis@gmail.com

1 Introduction

Constraint Satisfaction Problem (CSP) solver can be used to determine if a problem is satisfied by formalizing the task as variables with their domain and constraints between these variables. There are many heuristics that have been introduced (Hemert, 2002) to make the decision making process more efficient. In the current project authors are researching key concepts that are on the one hand light and simple (from the perspective of data structure maintenance) and effective on the other hand (from the perspective of the time complexity). Experiments are be performed on the Sudoku game. Next, it's shown analytically that the complexity of the problem can be reduced significantly even by modelling it in the different way. Later CSP solver with different heuristics (namely arc consistency and splitting strategies) are considered to investigate their influence and contribution into the final effectiveness.

2 Architecture

The CSP solver program has been implemented in Python. In comparison to other programming languages Python is a convenient tool for rapid prototyping. It's dynamic, expressive and contains useful tools for sets manipulation.

The CSP solver implementation consists of following components:

- **Preprocessor** transforms Sudoku board models into the Constraint Satisfaction Problem. It returns variables with their domain and constraints that contain only inequality operations for each of these variables.
- **Solver** implements the core CSP solving algorithm with heuristics. Solver is independent from domain specific knowledge except the insight that all constrains are built from not equality expression that makes the CSP solver less generic, but more efficient. With this domain related knowledge we can model constraints propagation as a manipulation with sets of nominals that are variables with their domain.
- **Filler** turns the solution returned by the **Solver** back to the Sudoku board.

A set of variables is represented as a map, where a key is a variable name and the value is a set of its possible values (i.e. domain). Constraints are represented as a map, where a key is a variable name, and the value is a set of viable names with which the particular variable should not be equal to.

3 Modelling

Two models have been introduced to solve the Sudoku board problem.

3.1 Model 1

The first model tries to fill board cells by numbers. Each cell can be represented as a variable encoded by indexes of row and column **i,j** (we will name each of cells by their indexes later), where $i, j \in \mathcal{N} \cap [0, D)$, D is a dimension of the Sudoku board. The domain of each variable $\mathcal{D}(\mathbf{i}, \mathbf{j}) = \{i\}_{i=1}^{D}$. If a domain of a cell contains only 1 value then we are saying that $\mathcal{D}(\mathbf{i}, \mathbf{j}) = k$, where k is a particular value. The same case is for the case when a cell has already been filled.

Constraints for each cell are defined according to the Sudoku game rules:

- $\mathcal{D}(\mathbf{i}, \mathbf{j}) \neq \mathcal{D}(\mathbf{i}, \mathbf{j}^*),\ j^* \neq j,\ where\ j^* \in \mathcal{N} \cap [0, D)$ that means cells in one column should have different values.
- $\mathcal{D}(\mathbf{i}, \mathbf{j}) \neq \mathcal{D}(\mathbf{i}^*, \mathbf{j}),\ i^* \neq i,\ where\ i^* \in \mathcal{N} \cap [0, D)$ that means cells in one rows should have different values.
- $\mathcal{D}(\mathbf{i}, \mathbf{j}) \neq \mathcal{D}(\mathbf{i}^*, \mathbf{j}^*),\ i^* \neq i,\ j^* \neq j,\ where\ i^*, j^*$ are indexes of elements of the squares where the cell **i,j** is situated, that means cells in one rows should have different values.

3.2 Model 2

The second model defines the Sudoku game from the opposite perspective by filling numbers by cells. The following variables are defined: $\mathbf{K_{i,c}}, \mathbf{K_{i,r}}, \mathbf{K_{i,b}}$, where $K \in \mathcal{N} \cap [1, D],\ i \in \mathcal{N} \cap [0, D)$. We are encoding cell indexes by numbers $\mathcal{N} \cap [0, D \times D)$. We are defining $\mathbf{K_{i,c}}$ as a number $\mathbf{K}$ that can be positioned **only** in the i-th column, $\mathbf{K_{i,r}}$ as a number that can be positioned only in the i-th row, and $\mathbf{K_{i,b}}$ as a number that can be positioned only in the i-th square. These restrictions allow us to define domains of our variables differently depending on their role in the Sudoku game, therefore the maximum size of the variable domain remains D as in the model 1.

Constrains of the second model can be determined as following:

- $\forall i : \cup_{\forall j} \mathbf{K_{i,c}} = \mathbf{K_{j,r}}$, that means that column number can has the same value like in any of row numbers.
- $\forall i : \cup_{j \in C} \mathbf{K_{i,c}} = \mathbf{K_{j,b}}$, where C is a set of squares that intersects a particular i-th column.
- $\forall i : \cup_{\forall j} \mathbf{K_{i,r}} = \mathbf{K_{j,c}}$, that means that row number can has the same value like in any of column numbers.
- $\forall i : \cup_{j \in C} \mathbf{K_{i,r}} = \mathbf{K_{j,b}}$, where C is a set of squares that intersects a particular i-th row.
- The similar constraints with rows and columns that intersect the i-th square can be defined for the square variable $\mathbf{K_{i,b}}$.

Constrains that describe disjunction of equality can be easily transformed to the conjunction of inequality to cells. Let $\mathbf{S}$ be all possible variables in the model domain and $\mathbf{G}$ be variables that our particular variable should be equal to, then we can model conjunction of inequalities by modelling variables from the set $\mathbf{T} = \mathbf{S} \backslash \mathbf{G}$[1].

3.3 Complexity analysis

For simplicity an approximate complexity analysis for the worse case has been done to compare two models. In the first model in the worse case we have $D \times D$ variables with a domain of D values. That means that in the worse case we obtain a decision tree of $D^{D \times D}$ nodes. For each variable there are 3(D-1) constraints (D-1 for row, D-1 for column, D-1 for square).

[1]For detailed explanation please refer to the project source code.

For the second model we have $3D \times D$ variables with a domain of D values, that lead to the decision tree of $D^{3D \times D}$ nodes in the worse case. For each variable $K_{i,b}, b \in \{r, c, b\}$ there are $2D^2 - \frac{D}{3} - 1$ constraints.

With this analysis it can be assumed that model 1 is more expressive and efficient for the CSP solver. In the model 1 an amount of constraints for each variable is linear in comparison to quadratic amount of them in the model 2. In addition, the decision tree of the model 2 is significantly more complex that leads to significantly harder problem.

4 Optimization techniques

4.1 Arc consistency

To increase the speed of the CSP solver propagation is used. In the current project arc consistency has been implemented. It determines if a subset of the variables is consistent. In the case of the Sudoku problem this means validating whether a variable causes the domain of other variables to be reduced. In the implementation of this project it works as the following:

Repeat until there are new changes in variable domains:

1. Go through each variable;
2. Check if the domain of the particular variable consists of one value (atomic);
3. Check if constraints are satisfied for related variables;
4. If not - shrink dependent variable's domain.

4.2 Splitting

For splitting two strategies are used. The first strategy (Split 1) is to split the first considered variable that has a domain size bigger than 1. The other strategy (Split 2) is to split first the variable that has the smallest domain. This splitting requires to loop over all variables to determine the smallest domain (that can lead to some performance decrease in the simple problems, and we will see that in experiments). After the loop the smallest domain is chosen.

5 Experiments

Two experiments have been introduced to compare different models and heuristics. The first experiment is based on the dataset of 1000 Sudoku boards provided by Frank van Harmelen. For each of combinations (model + propagation + splitting strategy), that can be performed in the reasonable time, distributions of the running time, amount of propagations and splitting have been measured. The experiment without propagation are excluded due to the larger runtime of solving it without propagation.

The second experiment has been introduced for combinations that cannot be run in the reasonable time on the previous dataset either because of the complexity of the proposed model or the inefficiency of heuristics. Sudoku boards with different amount of empty cells have been sampled to obtain 200 boards of each category. For each of 200 boards the mean value of the running time, amount of propagations and splitting have been collected to obtain an insight about the model behaviour with the growth of the decision tree.

6 Results

	mean split 1	mean split 2	var split 1	var split 2
time	0.74	0.32	2.6	0.4
split	236	100	265315	37648
propagation	96	51	42978	9449

Table 1: Mean and variance

In the figure 1 mean value of the run-time, propagation and splitting are shown. The graph illustrates how model 1 and model 2 compare in the above mentioned measurements. The y axis contains the variables time, propagation and splitting and the x axis contains the number of empty cells. Although not clearly visible in the graph on the the mean of model 1 runs close to the x axis and is indicated in red.

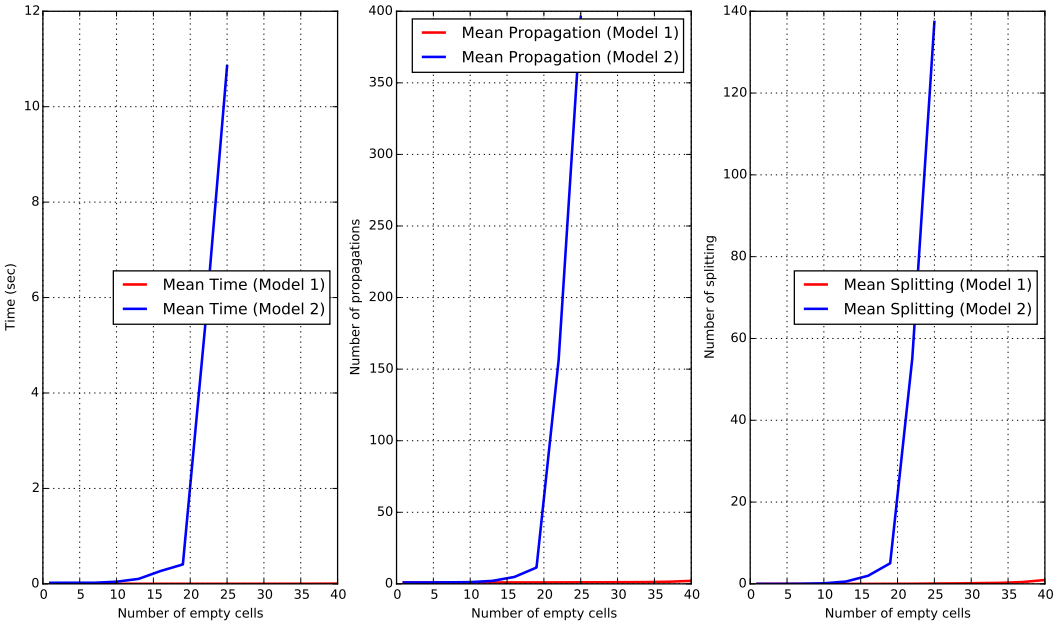

Figure 1: *Time, propagation and split difference between model 1 and model 2*

In figure 2 the difference between the two splitting strategies are shown. The graphs is a bin graph. The higher a bins, the more Sudoku are solved in that time, propagations or splits. The green colour in the graph indicates an overlap between the two models.

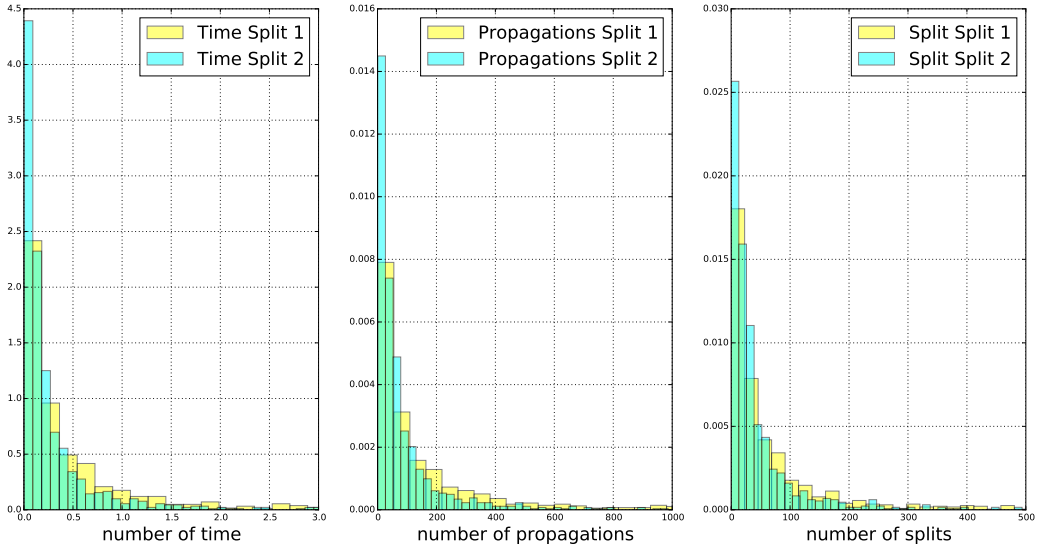

Figure 2: *Split 1 versus split 2*

In figure 3 the difference between every Sudoku individually is shown. This graphs is calculated by subtracting every data point of the splitting strategy 1 and the splitting strategy 2, given us the difference per data point. The red surface indicates an amount of difference between splitting strategies.

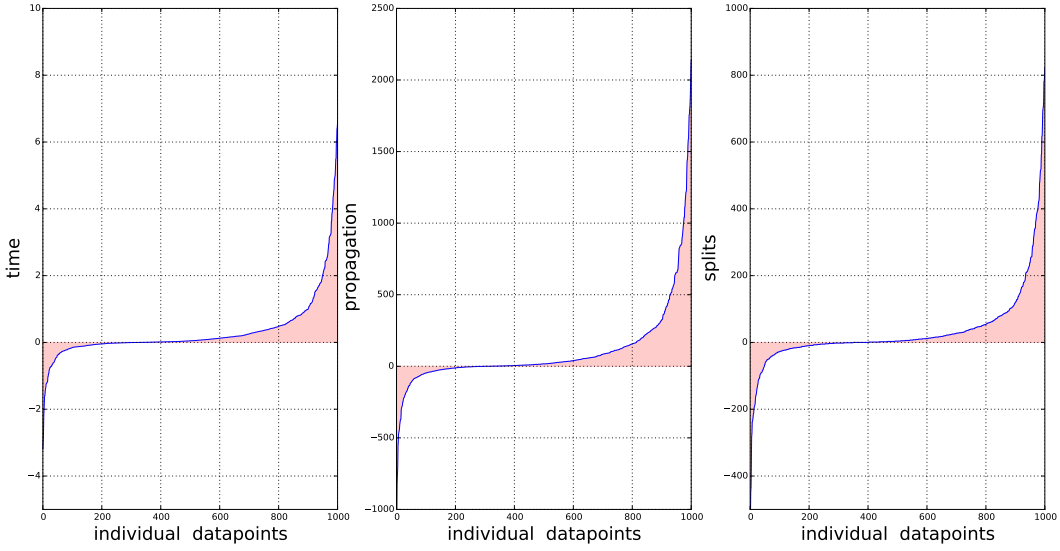

Figure 3: *Difference between data points individually*

7 Conclusion

The results in figure 1 shows that the complexity of the model 2 is greater then the complexity of the model 1 that is expected. However, in our current implementation this means that the Sudoku from the test set could not be solved in a reasonable time. Therefore, this project concludes that model 1 is a better method of modelling given the complexity. In addition, the model 1 is intuitively understand which this project think is of value.

Figure 2 and 3 both shown that the splitting strategy 2, splitting on smallest domain, is better then splitting strategy 1, splitting on first domain. This is clearly seen in both figures and in the table 1. Table 1 also indicates that the mean and variance overall are lower on all measurements. This leads to the conclusion that split 2 is also more consistent. In graphs 2, a larger blue bin can be noticed. This indicates that a large share of the Sudoku puzzles are solved in a smaller time then by splitting strategy 1. This effect is also noticed in 3 where the positive surface is bigger then the negative surface for every measurement. Nonetheless, the negative surface indicates that splitting strategy 2 is not always more effective. This results can be explained by the fact that it takes more time to determine the smallest domain then to solve it by the first domain. On top of that, it might be this strategy still might lead to non-optimal solutions and therefore in some cases make more propagation and splits.

8 Future works

The CSP solver used in current project has a few possibilities for future works. One of the original intentions of this project was to compare different splitting strategies with different modelling methods. This was not preformed due the complexity of the model 2. However, this comparing aspect still is an interesting research. To obtain such results one needs to use more heuristics in the CSP solver to make the model 2 faster. In addition, one could then compare how different heuristics or combination of then preform on different models.

References

J. I. v. Hemert. *Application of evolutionary computation to constraint satisfaction and data mining.* 2002.