Sudoku SAT solver

Theofilos Papapanagiotou, Hung Nguyen

Video presentation of the report and the Code repository are available.

Hypothesis

Enriching the encoding of a Sudoku as a SAT-problem, will improve the performance of the SAT-solver by decreasing the conflicts, propagation's and decisions. The intuition here is that enriching the encoding by adding more constraints, will decrease the search space. This will in turn improve performance by decreasing the amount of conflicts, propagation's and decisions. This experiment aims to find out how different representations of the same problem might perform differently performance-wise. It is interesting to identify which type of encoding technique has the most influence on the performance. We will not only be looking at different ways to encode the Sudoku rules themselves, but also in ways to exploit the assigned variables to do some Sudoku-specific optimizations.

Experimental setup

Tools

To get something meaningful statistics from the experiment, a SAT-solver was needed that keeps track of the statistics. For this, the existing SAT-solver MiniSAT[SE05] was chosen. This is a minimalistic, high-performance and easy-to-use SAT solver that fulfilled the requirements.
To accurately measure the performance of different Sudoku-to-SAT encodings, these had to be created from scratch to ensure it's correct implementation.

Implementation

The rules of Sudoku are as follows:

1. Each cell must contain a digit from 1 to n
2. Each cell can only contain a single digit
3. Each row must contain all digits from 1 to n
4. Each row cannot contain the same digit twice or more
5. Each column must contain all digits from 1 to n
6. Each column cannot contain the same digit twice or more
7. Each region must contain all digits from 1 to n
8. Each region cannot contain the same digit twice or more

These same rules can used to represent a Sudoku problem as a SAT problem, by translating them using propositional logic. We use a variable v_{rc} to represent a value v at row index r and column index c.
This encoding is similar to the encoding introduced by [KJ06]. However, their way of encoding the block constraints didn't work correctly and was very unclear. Therefore, we used our own.

$cell_d$: each cell must contain a digit from 1 to n

$$\bigwedge_{r=1}^{n}\bigwedge_{c=1}^{n}\bigvee_{v=1}^{n} v_{rc}$$

$cell_u$: each cell can only contain a single digit

$$\bigwedge_{r=1}^{n}\bigwedge_{c=1}^{n}\bigwedge_{v_i=1}^{n-1}\bigvee_{v_j=v_i+1}^{n} \neg v_{i_{rc}} \vee \neg v_{j_{rc}}$$

row_d: each row must contain all digits from 1 to n

$$\bigwedge_{r=1}^{n}\bigwedge_{v=1}^{n}\bigvee_{c=1}^{n} v_{rc}$$

row_u: each row cannot contain the same digit twice or more

$$\bigwedge_{r=1}^{n}\bigwedge_{v=1}^{n}\bigwedge_{c_i=1}^{n-1}\bigvee_{c_j=c_i+1}^{n} \neg v_{rc_i} \vee \neg v_{rc_j}$$

col_d: each column must contain all digits from 1 to n

$$\bigwedge_{c=1}^{n}\bigwedge_{v=1}^{n}\bigvee_{r=1}^{n} v_{rc}$$

col_u: each column cannot contain the same digit twice or more

$$\bigwedge_{c=1}^{n}\bigwedge_{v=1}^{n}\bigwedge_{r_i=1}^{n-1}\bigvee_{r_j=r_i+1}^{n} \neg v_{r_ic} \vee \neg v_{r_jc}$$

$block_d$: each region must contain all digits from 1 to n

$$\bigwedge_{v=1}^{n} \bigwedge_{g=1}^{n} \bigvee_{r=\sqrt{n}\cdot\frac{g}{\sqrt{n}}}^{\sqrt{n}\cdot(\frac{g}{\sqrt{n}}+1)} \bigvee_{c=\sqrt{n}\cdot(g \pmod{\sqrt{n}})}^{\sqrt{n}\cdot(g \pmod{\sqrt{n}})+1)} v_{rc}$$

(note: here we assume integer division)
$block_u$: each region cannot contain the same digit twice or more

$$\bigwedge_{v=1}^{n} \bigwedge_{g=1}^{n} \bigwedge_{r_i=\sqrt{n}\cdot\frac{g}{\sqrt{n}}}^{\sqrt{n}\cdot(\frac{g}{\sqrt{n}}+1)} \bigwedge_{c_i=\sqrt{n}\cdot(g \pmod{\sqrt{n}})}^{\sqrt{n}\cdot(g \pmod{\sqrt{n}})+1)} \bigvee_{r_j=r_i+1}^{\sqrt{n}\cdot(\frac{g}{\sqrt{n}}+1)} \bigvee_{c_j=c_i+1}^{\sqrt{n}\cdot(g \pmod{\sqrt{n}})+1)} \neg v_{r_i c_i} \vee \neg v_{r_j c_j}$$

(note: here we assume integer division)

Using these constraints, the rules of Sudoku can be representing in the following three encodings:
$minimal = cell_d \wedge row_u \wedge block_u \wedge assigned$
$efficient = cell_d \wedge cell_u \wedge row_u \wedge col_u \wedge block_u \wedge assigned$
$extended = cell_d \wedge cell_u \wedge row_d \wedge row_u \wedge col_d \wedge col_u \wedge block_d \wedge block_u \wedge assigned$

Optimization's

Taking advantage of the fact that this is a Sudoku-problem, a few optimization's can be performed for every known cell v_{rc} in the Sudoku. These are all implied clauses, which can be removed beforehand. We refer to the encodings with the implied clauses removed, as the optimized encodings.

All other cells on the same column are not number v:

$$\bigwedge_{v_i=1}^{n} \neg v_{i_{rc}} \text{ if } v_i \neq v$$

All other cells on the same column are not number v:

$$\bigwedge_{r_i=1}^{n} \neg v_{r_i c} \text{ if } r_i \neq r$$

All other cells on the same row are not number v:

$$\bigwedge_{c_i=1}^{n} \neg v_{r c_i} \text{ if } c_i \neq c$$

All other cells in the same region are not number v:

$$\bigwedge_{r_i=\sqrt{n}*\frac{r}{\sqrt{n}}}^{\sqrt{n}*(\frac{r}{\sqrt{n}}+1)} \bigwedge_{c_i=\sqrt{n}*\frac{c}{\sqrt{n}}}^{\sqrt{n}*(\frac{c}{\sqrt{n}}+1)} \neg v_{r_i c_i} \text{ if } r_i \neq r \text{ and } c_i \neq c$$

(note: assuming integer division)

Experimental results

Given the output results collected by the execution of the SAT solver using minisat, we are reporting the performance of the 6 different encodings. The selected metrics which we are using to report the performance, are

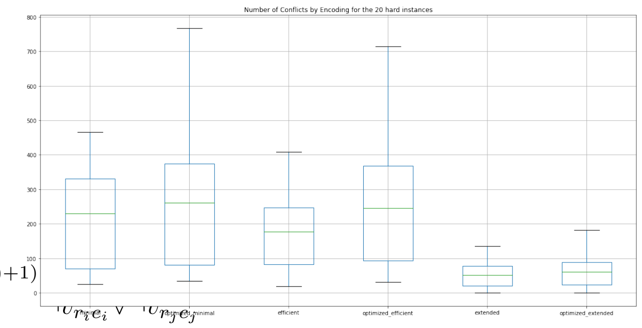

Figure 1: Graph of the conflicts

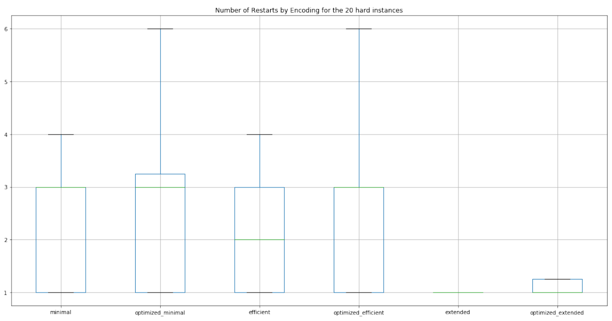

Figure 2: Graph of the restarts

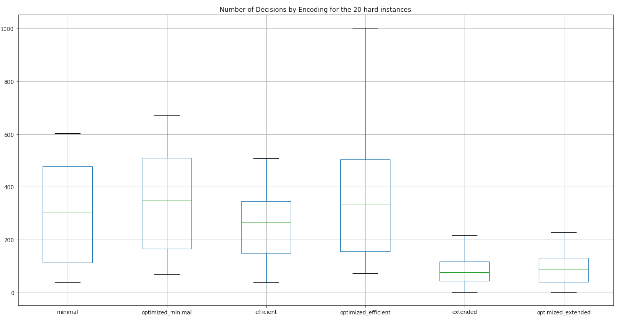

Figure 3: Graph of the decisions

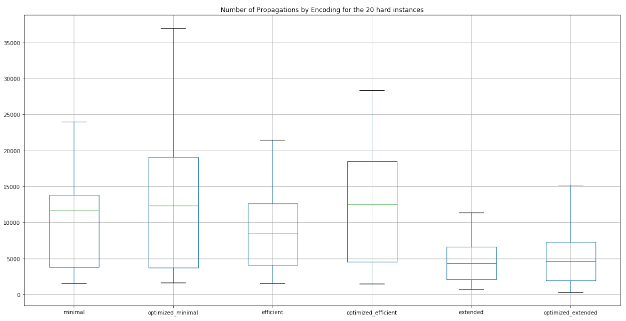

Figure 4: Graph of the propagations

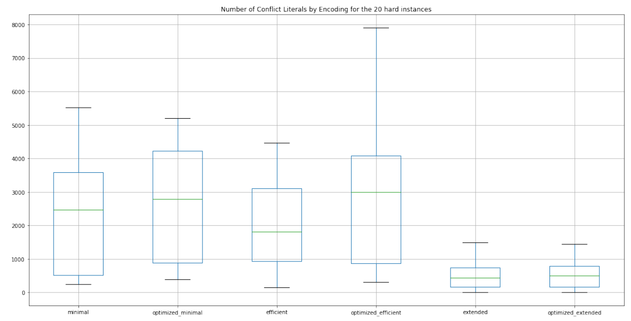

Figure 5: Graph of the conflict literals

the number of conflicts, restarts, decisions, propagations and conflict literals.

The size of the selected dataset is small, only 20 hard instances have been used. In order to validate the hypothesis with more certainty, we could have used a larger dataset with 17 givens, still choosing from hard instances [S+92] [SML96].

The selection of the performance metrics was based in the available report of the chosen tool [SE05]. A more recent SAT Solver [Soo16] reports more metrics, based on the same basic indicators we have used in this project, but examining them during the execution and in relation to each other.

The evaluation of a larger dataset in combination with the extended indicators would provide more concrete results for this measurement.

Interpretation

The experimental results confirm our hypothesis. The 5 chosen metrics show improved performance, as the numbers of the metrics are lower in the more efficient encodings. Especially the group of the two extended encodings which contains the more complete set of clauses, outperforms the groups of minimal and efficient encodings. The minimal and efficient encodings compared to each other, do not have large differences in the observed metrics. The introduction of the optimization step to eliminate the literals and the clauses, did not improve much the performance of the SAT solver compared to the execution without these steps.

Conclusion summary

It was expected that the introduction of more constraints will decrease the search space and improve the performance metrics we have chosen.

The dataset and tools used can be improved to increase the certainty of the hypothesis. Additional heuristics can be used to create a wide range of different improvement steps.

References

[S+92] Bart Selman, Hector J Levesque, David G Mitchell, et al. “A New Method for Solving Hard Satisfiability Problems.” In: AAAI. Vol. 92. 1992, pp. 440–446.

[SML96] Bart Selman, David G Mitchell, and Hector J Levesque. “Generating hard satisfiability problems”. In: Artificial intelligence 81.1-2 (1996), pp. 17–29.

[SE05] Niklas Sorensson and Niklas Een. “Minisat v1. 13-a sat solver with conflict-clause minimization”. In: SAT 2005.53 (2005), pp. 1–2.

[KJ06] Gihwon Kwon and Himanshu Jain. “Optimized CNF encoding for sudoku puzzles”. In: Proc. 13th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR2006). 2006, pp. 1–5.

[Soo16] Mate Soos. “The CryptoMiniSat 5 set of solvers at SAT Competition 2016”. In: SAT COMPETITION 2016 (2016), p. 28.