

A Theory Learning SAT-Solver for Sudokus

Project Report

October 3, 2017

1 Introduction

Due to the expressiveness of propositional logic, SAT-solvers allow the automated solution of a great variety of problems. Although the SAT-problem is the paradigmatic NP-complete problem typical complexity of SAT-problems is usually benign. In addition, recent years have seen an immense improvement in the efficiency of SAT-solvers. Thus it is no possible to use SAT-solvers to tackle problems whose conjunctive normal form (CNF) encodings involve hundreds of thousands of variables.

A much smaller problem commonly found for entertainment in newspapers or magazines are Sudoku riddles. Sudokus are also a commonly studied paradigm for SAT-solvers as well as constraint programming. Constraint programming sacrifices generality for efficiency by tackling by tailoring solvers for hard problems using various constraints that go beyond the mere axioms of propositional logic. A common approach in constraint programming are “Satisfiability-Modulo-Theories” (SMT)-solvers. They supplement a SAT-solver by a specialised theory(T)-solver exploiting both the high degree of optimization of SAT-solvers and the domain specific efficiency of T-solvers.

For example in the DPLL(T)-algorithm a DPLL-based procedure is used to SAT-check a formula F starting from an empty T-box. If the formula is satisfiable it is passed to a specialised T-solver that checks whether it is T-consistent. If not, $\neg F$ is added to the SAT-solver’s axioms and so forth.

Both SAT and SMT approaches have their merits: SAT is a very general approach that requires minimal Human preprocessing while SMT can be much more efficient for specific domains.

In this report we investigate the middle ground between the two approaches for the domain of 9x9 Sudokus. Namely, we do not use a T-solver to add to the axioms but check the clauses learnt by the SAT-solver on specific Sudokus for global validity on Sudokus. If a valid clause is found, it is added to the axioms.

Based on previous results we hypothesize that in this way it is possible to train SAT-solvers to become more efficient on the current domain. We attempt to keep the procedure as general as possible so that it can be applied to any domain, not only Sudokus.

The rest of this report is organised as follows: the results of previous work that led us to our hypothesis are summarised in section 3; we present our methodology including the choice of dataset, encodings, SAT-solver and performance metrics in section 4; in section 5 we propose an algorithm for SAT-theory learning including clause pruning, validity checking and validity pruning and various training protocols; section 6 summarizes our findings; finally, in section 7 we present our conclusions and suggest avenues for further research.

2 Preliminaries

Given a set of propositional variables, we call the variables and their negations *literals*. A *clause* s is a set of literals interpreted as a disjunction. We call a set S of clauses interpreted as a conjunction a *SAT-problem* (in CNF). A function ϕ that maps some problem to a *SAT-problem* is called an *encoding*. As far as possible we will treat a SAT-solver as a blackbox that tells us whether a SAT-problem in CNF is satisfiable and returns a satisfying assignment of the propositional variables. A SAT-problem is *proper* if it admits exactly one satisfying assignment. Given a set of SAT-problems $D = \{S_1, \dots, S_n\}$ with $\cap_{i=1}^n S_i \neq \emptyset$ in CNF we refer to D as the *domain* of the problem, $A = \cap_{i=1}^n S_i$ as the *axioms* of D and to $C_i = S_i \setminus \cap_{i=1}^n S_i$ as the *claims* or *assertions* of problem i . For $c \in C$, if the SAT-solver returns “unsatisfiable” for $S = A \cup c$ we call $\neg c$ *valid* and a *theorem*. An axiom $a \in A$ is *redundant* iff it is a theorem for any domain $D \setminus_a$ with axioms $A \setminus_a = A \setminus a$. For any theorem, we call its minimal valid subclauses the *kernels* of the theorem.

In this study D_{min} , D_{eff} , and D_{ext} are the sets of proper SAT-problems obtained by applying the corresponding encoding functions $\phi_{minimal}$, $\phi_{efficient}$ and $\phi_{extended}$ to a dataset of Sudokus to be specified below. The encodings furnish different sets of axioms A_{min} , A_{eff} and A_{ext} . In addition, each individual SAT-problem is characterised by claims corresponding to the Sudoku’s givens. We henceforth use Sudoku to refer to the SAT-problem corresponding to the Sudoku as well when the context is clear.

3 Related Work

Redundant axioms for Sudoku are mainly considered in constraint programming approaches. E.g. Demoen and de la Banda (2014, 11) find that: “Redundant constraints are very often good for the performance of CP systems, and indeed, all solvers we checked perform much slower (about a factor 2000) with a minimal set of big constraints.” Simonis (2005) suggests several such redundant constraints and finds they boost performance of CP propagation schemes.

As opposed to CP-systems SAT-solvers usually only have very limited propagation schemes such as unit propagation. However, the literature on Sudoku as a SAT-problem considers redundant axioms from a different perspective. A large part of it is devoted to optimizing CNF-encodings for Sudoku with respect

to SAT-solver performance. Lynce and Ouaknine (2006) suggest and test two encodings: a *minimal* and an *extended* encoding. The extended encoding is nothing else than an encoding containing redundant axioms. They compare the encodings efficiency with respect to SAT-solvers’ ability to solve Sudokus without search. They find among other things that substituting the minimal by the extended encoding increases the likelihood of solving a Sudoku without search using only unit propagation from 1% to 69%. In addition, Kwon and Jain (2006) compared these encodings with respect to runtime over various Sudoku-sizes and found that the extended encoding scales much better with increased problem size.

This apparent convergence in two strains of literature led us to hypothesize that redundant axioms might be beneficial to SAT-solver performance in spite of their limited inference tools. We thus wanted to find out which redundant axioms would be most helpful in strengthening inference and minimizing search. One way to go about this is to look for redundant axioms that the SAT-solver actually *learns* during the search process.

4 Methodology

We decided to test our hypothesis experimentally. In order to do this, it was necessary to choose a suitable dataset, SAT-solver and a meaningful metric of performance as well as a CNF-encoding of Sudoku.

4.1 Dataset

The domain of all proper Sudokus has a cardinality of more than $6.67 * 10^{21}$. Therefore it is necessary to work with a subdomain. We used the collection of minimum (i.e. 17 givens) proper sudokus assembled by Gordon Royle (*Minimal Sudoku Dataset*, n.d.). A minimal Sudoku is a Sudoku from which no given can be removed leaving it a proper Sudoku. The dataset consists of 49151 such sudokus.

We used 10000 sudokus to measure the performance of the SAT solver across different encodings and different base clauses. We used 5000 sudokus to train our SAT solver to learn new clauses which are valid for sudokus.

4.2 Sudoku Encodings and DIMACS

We use the three different types of encoding functions specified in (Lynce & Ouaknine, 2006) and (Kwon & Jain, 2006) for our experiments: the minimal encoding, the efficient encoding, and the extended encoding.

The difference in the encodings is their axioms. Namely, $A_{min} \subset A_{eff} \subset A_{ext}$. Since the minimal encoding is a complete axiomatization of Sudoku, all of the additional axioms in the other encodings are redundant.

The encodings all have to encode the following four constraints:

- each cell has to be filled with a number from 1-9

- each row must contain each number exactly once
- each column must contain each number exactly once
- each square block (eight on the edges, one in the center) of nine cells must contain each number exactly once

We follow Kwon and Jain (2006) in representing them. The constraints are encoded using the same sets of propositional variables: in an 9×9 Sudoku puzzle each cell must contain a number between 1 and 9. Thus, each cell is associated with 9 propositional variables which can be represented by 3-tuples (r, c, v) as variables. Then we get $V = \{(r, c, v) | 1 \leq r, c, v \leq 9\}$ as the set of propositional variables

The difference in the encodings stems from the way they encode the constraints. The following conjunctions refer to each cell, row, column or block having at least one number from 1 to n (i.e. *definedness*) or at most one number from 1 to n (i.e. *uniqueness*). A subscript d to denote definedness constraints and subscript u to denote the uniqueness constraints.

$$\begin{aligned}
Cell_d &= \bigwedge_{r=1}^n \bigwedge_{c=1}^n \bigwedge_{v=1}^n (r, c, v) \\
Cell_u &= \bigwedge_{r=1}^n \bigwedge_{c=1}^n \bigwedge_{v_i=1}^{n-1} \bigvee_{v_j=v_i+1}^n \neg(r, c, v_i) \vee \neg(r, c, v_j) \\
Row_d &= \bigwedge_{r=1}^n \bigwedge_{v=1}^n \bigvee_{c=1}^n (r, c, v) \\
Row_u &= \bigwedge_{r=1}^n \bigwedge_{v=1}^n \bigwedge_{c_i=1}^{n-1} \bigwedge_{c_j=c_i+1}^n \neg(r, c_i, v) \vee \neg(r, c_j, v) \\
Col_d &= \bigwedge_{c=1}^n \bigwedge_{v=1}^n \bigvee_{r=1}^n (r, c, v) \\
Col_u &= \bigwedge_{c=1}^n \bigwedge_{v=1}^n \bigwedge_{r_i=1}^{n-1} \bigwedge_{r_j=r_i+1}^n \neg(r_i, c, v) \vee \neg(r_j, c, v) \\
Block_d &= \bigwedge_{r_{offs}=1}^{\sqrt{n}} \bigwedge_{c_{offs}=1}^{\sqrt{n}} \bigwedge_{v=1}^n \bigvee_{r=1}^{\sqrt{n}} \bigvee_{c=1}^{\sqrt{n}} (r_{offs} * \sqrt{n} + r, c_{offs} * \sqrt{n} + c, v) \\
Block_u &= \bigwedge_{r_{offs}=1}^{\sqrt{n}} \bigwedge_{c_{offs}=1}^{\sqrt{n}} \bigwedge_{v=1}^n \bigwedge_{r=1}^{\sqrt{n}} \bigwedge_{c=r+1}^{\sqrt{n}} \\
&\quad \neg(r_{offs} * \sqrt{n} + (r \bmod \sqrt{n}), c_{offs} * \sqrt{n} + (r \bmod \sqrt{n}), v) \\
&\quad \vee \neg(r_{offs} * \sqrt{n} + (r \bmod \sqrt{n}), c_{offs} * \sqrt{n} + (c \bmod \sqrt{n}), v)
\end{aligned}$$

In addition one more conjunction is needed to represent givens. Each given contains a pre-assigned number and thus the givens form a conjunction of unit

clauses. Let $G_i = \{(r_1, c_1, v_1), \dots, (r_k, c_k, v_k)\}$ be the set of givens of Sudoku S_i . Then we have:

$$Assigned = \bigwedge_{i=1}^k (r, c, v), \text{ where } (r_j, c_j, v_j) \in G_i$$

The differences in the three mentioned encodings then consist in which of the definedness and uniqueness conditions they include (Kwon & Jain, 2006):

$$\begin{aligned}\phi_{minimal} &= Cell_d \wedge Row_u \wedge Col_u \wedge Block_u \wedge Assigned \\ \phi_{efficient} &= Cell_d \wedge Cell_u \wedge Row_u \wedge Col_u \wedge Block_u \wedge Assigned \\ \phi_{extended} &= Cell_d \wedge Cell_u \wedge Row_d \wedge Row_u \wedge Col_d \wedge Col_u \wedge \\ &\quad Block_d \wedge Block_u \wedge Assigned\end{aligned}$$

Note that $\phi_{minimal}$ is a complete axiomatization of Sudoku and the additional axioms of the other encodings are thus redundant.

The encodings are represented in the DIMACS to pass them to the SAT-solver. It has the following format:

At the beginning of the file there may be one or more comment lines. Comment lines start with a 'c'. They are followed by:

p FORMAT VARIABLES CLAUSES

FORMAT is for programs to detect which format is to be expected. In our case this is "cnf"

VARIABLES is the number of unique variables in the expression

CLAUSES is the number of clauses in the expression

This line is followed by the information in SAT-problem. Unique variables are enumerated from 1 to n. A negation is represented as '-'. Each line represents one disjunction and is delimited by "0". Example: $(A \vee \neg B \vee C) \wedge (B \vee D \vee E) \wedge (D \vee F)$ is represented in DIMACS as:

```
c This is a comment
c This is another comment
p cnf 6 3
1 -2 3 0
2 4 5 0
4 6 0
```

4.3 The Minisat-Solver

The SAT-problems in DIMACS format are then passed on to a SAT-solver. For our experiment we use a SAT solver called Minisat. Minisat implements the DPLL algorithm and additionally implements conflict-driven learning and

non-chronological backtracking. As Minisat has an upper limit on the number of learned clauses allowed it employs an activity heuristic when deciding which learned clauses to keep when it. The same heuristic is applied when doing backjumping. The heuristic essentially records which clauses and literals have been in recent conflicts and favors them over "stale" clauses and literals. For more information see (een2003extensible).

For our purposes we needed to record the learned clause for a given conflict. To do this we needed to edit the Minisat code such that when encountering a conflict the deduced learned clause is printed out. By printing out the learned clause straight away we avoided the upper limit of Minisat learned clauses.

4.4 Measures of Performance

In addition we had Minisat output the number of *decisions* per Sudoku. A decision consists in choosing a propositional variable and tentatively assigning it a truth value. The solver then checks whether the problem is satisfiable given the decision. Decisions occur when the SAT-solver exhausts its inference capabilities and resorts to search. Therefore the number of decisions is a good metric of the proportions of search vs inference the SAT-solver used in solving a SAT-problem. A low number of decisions indicates heavy use of inference, a high number indicates a lot of search. Given our hypothesis that redundant axioms can help inference, we would expect the number of decisions to decrease when more redundant axioms are added.

Overview: The project is about the effects of redundant constraints and the process of learning redundant constraints in Sudoku.

Problem description: Definition: A redundant constraint is a constraint that is valid given the other constraints. That is, if A is a model for constraint/clause $P(x,y)$, s.t. $A \models P(x,y)$ with $P'(x,y) \models P(x,y)$ then A is also a model for $Q(x,y,z)=P'(x,y) \vee R(z)$, that is, $A \models Q(x,y,z)$. $Q(x,y,z)$ is then a redundant clause.

Many SAT solvers use clausal learning in the process of finding a model for the problem. Clausal learning adds clauses to the set of formulas. The clauses added do not affect the model of the problem but makes explicit which valuations are not possible. If the clause which is learned is valid given the original restraints (it is redundant), then it could have been derived to begin with and added to the set of formulas and the SAT solver would not have to explore the tree that lead to the learned clause. These redundant clauses become more important when dealing with a class of problems which differ only in few formulas. Learning clauses which pertain to the class of problems, not the individual problems, could save time as it reduces the search space and tells us more about the structure of the problem. In particular, it tells us what clauses could be added to the initial problem description and further speed up the SAT solver by adding redundant clauses (<http://www.cs.cmu.edu/~hjain/papers/sudoku-as-SAT.pdf> [Sudoku as SAT]).

Checking validity of a clause is the VAL problem which is Co-NP. In other words, checking whether the negation is satisfiable will answer the VAL problem.

If the negation of a clause is not satisfiable, the clause must be valid.

Consider this example: Let's say the foreign policy of the country can be described by the following clauses:

$P' + Q'$ $Q' + R'$ $R + P$

In addition the king wants $Q + R$

The first three clauses then describe the class of problem and the king's demands are a particular instance of the problem. When running a SAT solver on this problem it might try splitting by trying Q . Then we are forced to choose P' and R' but then we cannot satisfy $R + P$. The clauses of the class of problem imply Q' . We can therefore add Q' to the problem class description and avoid splitting by Q in future runs of another problem instance.

Plan: Our plan is to explore the learned clauses when using a SAT solvers to solve Sudoku puzzles. More specifically, our plan is to find out which redundant constraints in Sudoku make SAT-solvers faster (here we should investigate the clauses from the extended encoding and also the constraints in <http://4c.ucc.ie/~hsimonis/sudoku.pdf> [Sudoku as constraint problem]) and especially we are interested in which of those redundant constraints are the most commonly learned.

We look at two different encodings of Sudoku puzzles. One called the minimal encoding (a misnomer, see <https://lirias.kuleuven.be/bitstream/123456789/353637/1/sudokutplp.pdf> [redundant sudoku rules]) and the other one the extended encoding. Both are taken from <http://www.cs.cmu.edu/~hjain/papers/sudoku-as-SAT.pdf> [Sudoku as SAT]. Most results point towards faster execution of SAT solvers, given more redundant constraints. Except in the case when the redundant constraints overflow the memory of the machine running the SAT solver. In those cases a minimal representation of the problem might be more feasible. We therefore conjecture that:

"For low numbers of variables (e.g. 3x3 Sudokus) SAT-performance can be increased by adding redundant constraints. For high numbers of variables (e.g. 81x81 Sudoku) SAT-performance suffers from adding redundant constraints."

"It is possible to train a SAT solver to perform faster on (a class of Sudokus) by letting it add the most commonly learned valid clauses to the constraints of the (class of) Sudokus."

Data: As a data set I would suggest the 50000 minimal sudokus found here: <http://staffhome.ecm.uwa.edu.au/~00013890/sudokumin.php> Another dataset that might be useful is this website <http://www.menneske.no/sudoku/> This is also what's used in most of the other papers.

Code:

5 The learning Algorithm

5.1 Pseudocode

5.2 Clause Pruning

5.3 Validity Checking

5.4 Validity Pruning

5.5 Training

6 Results

7 Conclusion

References

- Demoen, B., & de la Banda, M. G. (2014). Redundant sudoku rules. *Theory and Practice of Logic Programming*, 14(3), 363–377.
- Kwon, G., & Jain, H. (2006). Optimized cnf encoding for sudoku puzzles. In *Proc. 13th international conference on logic for programming artificial intelligence and reasoning (lpar2006)* (pp. 1–5).
- Lynce, I., & Ouaknine, J. (2006). Sudoku as a sat problem. In *Isaim. Minimal sudoku dataset*. (n.d.). <http://staffhome.ecm.uwa.edu.au/00013890/sudoku17>. (Accessed: 2017-09-25)
- Simonis, H. (2005). Sudoku as a constraint problem. *Proc. 4th Int. Works. Modelling and Reformulating Constraint Satisfaction Problems*, 13–27.