A Theory Learning SAT-Solver for Sudokus

Project Report

September 30, 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 (CNV) encodings involve hundreds of thousands of variables.

A much smaller problem commonly found for entertainment in newspapers or magazines are Sudoku riddles. The classic Sudoku involves 9x9 cells arranged in matrix - some of them blank some of them prefilled - and 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 constain each number exactly once
- each square block (eight on the edges, one in the center) of nine cells must contain each number exactly once

Sudokus are a commonly studied paradigm for SAT-solvers as well as constraint programming. Constraint programming sacrifices generality for efficienty by tackles 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 "Satisfiablity-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-solvers T-box 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 T-box 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 T-box. In this way we train the SAT-solver 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.

2 Formal Framework and Preliminaries

2.1 Sudoku Encodings and DIMACS

2.2 The Minisat-Solver

2.3 The Algorithm

2.4 Measures of Performance

3 Results

4 Conclusion

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 —= P(x,y) with P'(x,y) =——= P(x,y) then A is also a model for Q(x,y,z)=P'(x,y)vR(z), that is, A —= 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 constaints. 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: