Learning SAT Encodings for Individual Constraints

Felix Ulrich-Oltean[0000−0001−5162−5826]
Supervisors: Peter Nightingale and James Cussens

Department of Computer Science
University of York, UK, YO10 5DD
https://www.cs.york.ac.uk

Abstract. One popular strategy for solving Constraint Satisfaction Problems (CSPs) is to encode them into Boolean (SAT) formulae and then solve them using a SAT solver. We illustrate the process of encoding to SAT, discuss the notion of a "best" encoding and explore how to make fine-grained choices when encoding unseen problem instances. We present initial results which support our intention to train a machine learning algorithm to select encodings for individual constraints in a CSP instance.

Keywords: Constraint Programming · Combinatorial Satisfaction · SAT Encodings · Portfolio Systems · Pseudo-Boolean Constraints

1 Introduction

Constraint Satisfaction Problems (CSPs) can model many interesting and important real-life problems ranging from formal verification of hardware systems to allocation of resources, multi-criteria scheduling and layout optimisation. One popular and increasingly powerful way of solving a CSP is by translating (encoding) it into an instance of the Boolean Satisfiability Problem (SAT) – a propositional formula for which a satisfiying assignment of Boolean variables can either be found or not.

The overall goal of this project is to predict the "best" SAT encoding choice for each constraint in a given CSP instance. We begin by investigating which features of a CSP favour certain encodings and how

those features may be efficiently extracted or calculated. The intention is to train a machine learning model to make the choice of encoding and to evaluate this approach against choosing the accepted state-of-the-art encoding for each constraint class.

This paper is an introduction to my PhD topic and consists of:

- an overview of some key concepts and relevant literature covering CSPs, SAT and portfolio ML systems in this arena (§ 2),
- a description of some early experiments to establish the viability of the research (§ 3),
- an outline of the intended next steps, and an assessment of the challenges to be addressed in the course of this research (§ 4)

2 Background

2.1 Constraint Satisfaction Problems

A CSP consists of a set of variables and their associated domains, along with a set of constraints specifying what combinations of values are allowed for subsets of the variables. A classic example often cited is the Sudoku puzzle, traditionally with 81 variables, each with integer domain $[1 \dots 9]$; the constraints in Sudoku demand that no variables in the same row, column or mini-square can be assigned the same value.

Although constraints can be defined by explicitly enumerating the allowed combinations of values, it is often useful to express the constraints logically, for example requiring that a set of variables be assigned distinct values, as in the previous paragraph. Many languages exist which give the user the ability to describe CSPs as a high-level, human-readable *model*; some examples are OPL [13] , the MiniZinc language and its lower-level "compiled" version FlatZinc [9], XCSP3 with its associated API JvCSP3 [3] and the Essence Prime language used by Savile Row [10] along with its higher level relative Essence [5].

2.2 Savile Row and Essence Prime

Savile Row is a constraint modelling assistant [10] which takes a problem description in Essence Prime and tailors an output model to any one of several backend solvers, carrying out various reformulations, both to improve the model where possible and to take advantage of the target

Listing 1.1. Sudoku model in Essence Prime

```
1  letting RNG be domain int(1..9)
2  given clues: matrix indexed by [RNG,RNG] of int(0..9)
3  find M: matrix indexed by [RNG,RNG] of RNG
4  such that
5  forAll row,col: RNG .
6    clues[row,col]!=0 -> M[row,col]=clues[row,col],
7  forAll row: RNG . allDiff(M[row,..]),
8  forAll col: RNG . allDiff(M[..,col]),
9  forAll i,j: int(1,4,7) .
10   allDiff([M[k,l] | k: int(i..i+2), l: int(j..j+2)])
```

solver's strengths. Designed as a research tool, it provides many options for how the translation is done. Its output means a user can benefit from advances in solvers of different flavours, e.g. CP, SMT, SAT. Of particular interest is the ability to manually choose which SAT encoding is used for different classes of constraint in the model.

Listing 1.1 shows an example of how Sudoku could be modelled for Savile Row using Essence Prime[1]. Note that the `given` declaration allows instance-specific values to be set outside of the model file which describes the problem class – in this case, the `clues` matrix holding the initial values on the board (line 2). Lines 5 and 6 ensure that the given numbers are maintained in the final grid; lines 7-10 enforce the uniqueness of numbers in the three contexts of Sudoku (row, column, mini-square).

2.3 Encoding CSP to SAT

Any (finite-domain) CSP can be translated into the Boolean Satisfiability Problem (SAT). The simplicity of SAT, especially in CNF form[2] has led to a panoply of SAT-solvers with impressive and ever-improving

[1] This example is adapted from the Essence Prime manual [11].

[2] Boolean formulae in Conjunctive Normal Form are structured as a conjunction of clauses; each clause is a disjunction of literals; each literal is a Boolean variable or its negation.

performance. We can continue to benefit from these improvements with our strategy of solving CSPs by encoding to SAT. Of course other types of constraint solver are also improving but it has been shown that translating to SAT can give the best performance for some problem instances. In fact, PicatSAT [15] (which uses a SAT-solver) was the winner of the XCSP2019 constraint solver competition [1].

Many schemes exist for encoding CSPs to SAT; of particular interest in this project are encodings for constraints with arbitrary arity, such as cardinality constraints, pseudo-Boolean constraints and at-most-one constraints. For each type of constraint there are often several alternative encodings with competing advantages in terms of the size of the formula produced, propagation strength or suitability for being combined with other constraints.

To illustrate the concept, we consider the pseudo-Boolean sum constraint (PBC), whose scope is the Boolean variables $x_1 \dots x_n$ with associated weights $q_1 \dots q_n$ and which requires that $\sum x_i q_i \leq K$. This type of constraint is very common in many settings including resource allocation and timetabling.

One example of a PBC encoding is the Generalized Totalizer [7], which uses a tree to represent the possible totals of the terms on the left side of the inequality. Fig. 1 illustrates this structure. Notice that each term is represented as a leaf node with a variable whose name carries the weight of the term. Each non-leaf node has a Boolean variable for every possible non-zero total, culminating at the root with all the possible totals (excepting any totals that exceed K, all of which are represented by a value of $K + 1$ and shown in bold in Fig. 1).

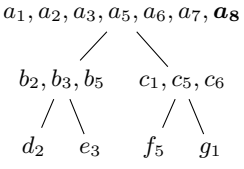

Fig. 1. Generalized Totalizer tree for $2x_1 + 3x_2 + 5x_3 + x_4 \leq 7$

In the SAT formula, clauses are introduced to "turn on" a variable in the parent node if the appropriate child variables are on, for example

$(\neg d_2 \vee \neg e_3 \vee b_5)$ ensures that b_5 is on if d_2 and e_3 are on. The overall PBC is satisfied if the $K+1$ variable remains "off"; in our example the clause would be $(\neg a_8)$.

Alongside the encoding presented above for PBCs, some alternatives include: BDD (based on binary decision diagrams), sequential weight counters (based on a set of logical adders) and polynomial watchdog (using the binary representation of coefficients). Bofill et al. describe and evaluate the relative merits of the encodings for different problem types in [4].

2.4 Selection

The application of machine learning algorithms to inform decisions in the process of solving CSPs is well established, not least in relation to the use of SAT solvers. Portfolio solvers such as SATZilla [14] choose the solver based on features of the SAT problem instance. The Proteus portfolio solver [6] uses three stages: first it considers whether to use a specialised CSP solver or to translate the problem to SAT, then chooses which SAT solver to employ, and finally which encoding to use. A comprehensive survey of portfolio approaches is given by Kotthoff in [8]. Soos et al. [12] use supervised learning techniques at a much lower level inside a SAT solver in order to support decision making in "tasks such as branching, clause memory management, and restarting" – their stated intention being to shed light on the inner workings of SAT solvers which are often used as *black boxes*.

3 First Steps

The initial aim is to find out how much difference the choice of encoding makes to solving times. When new encodings are studied and published, the benchmark problem instances used to gauge performance are selected in many different ways but usually authors are keen to show where their contribution has a positive impact on performance. We intend to study the performance of encodings in an objective way to answer empirically the question: *For a given type of constraint, is one encoding always the best choice or does it depend on features of the instance?*

We have begun by considering small CSPs involving a single class of constraint, intending to extend this in order to identify

- features of the instance which affect performance
- whether the best-performing encoding changes as those features vary, and
- whether any such findings persist when other constraints are present in a CSP.

The first constraint class considered is the PBC – this constraint has several encodings available (Savile Row provides support for five in the main release and more in development). It is also fairly straightforward to create an instance generator where we can vary many features.

The first model we wrote for testing just contained two such PBCs which were mutually exclusive, giving an unsatisfiable problem so as to exhaust the search space. We created problem instances with different numbers of variables and different configurations of coefficients (weights) for the variables, varying the amount of clustering and the size of the coefficients relative to the upper bound K.

In Fig. 2 we see the solving times for different amounts of clustering; $c = 0$ means the coefficients are randomly spread, whereas $c = 1$ means that each decision variable x_i has the same weight as some other variables, i.e. all weights occur in clusters of shared values. As more clustering occurs, the relative performance of the encodings varies significantly.

These initial findings suggest that there could be overall performance gains to be made by selecting the encoding based on features of the problem instance.

4 Future Direction

4.1 Intentions

We plan to conduct similar investigations with other constraints in order to build up a profile of performance. We will then investigate whether the presence of other constraints in a CSP affects the ranking of encodings by performance, working towards an understanding of which combinations of choices perform best.

When it comes to designing a translation to SAT, there is a tension between encoding size and propagation strength. Further considerations may include the time complexity of the encoding algorithm. There are also situations where an encoding can combine two types of

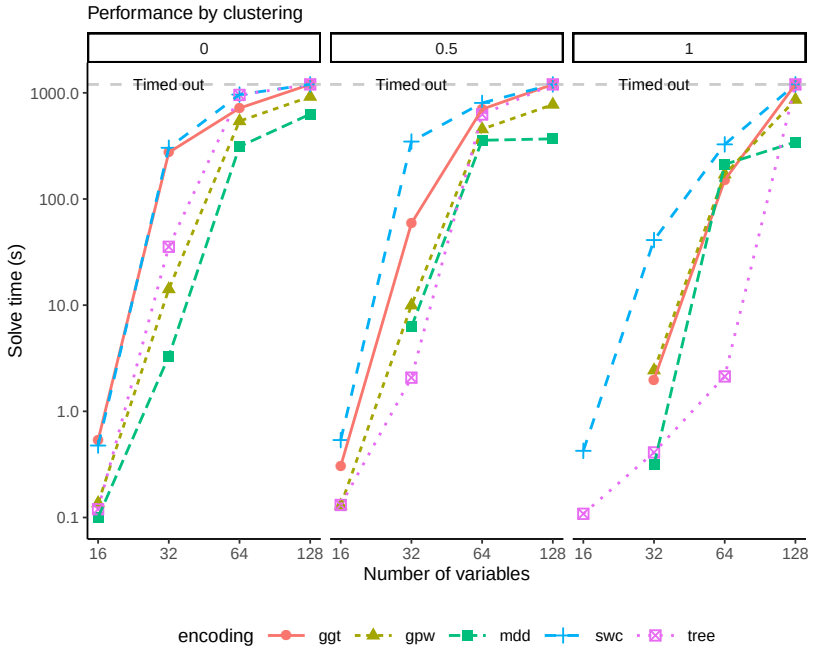

Fig. 2. Timings for different amounts of clustering

constraints very efficiently, for example the combination of At-Most-One with Pseudo-boolean in [2].

We envisage the machine learning algorithm combining all factors outlined above to decide the encoding for each constraint. One hope is that these investigations yield useful insights that can contribute to the understanding and development of SAT encodings.

4.2 Challenges

In theory it is possible to investigate countless variations in CSPs, but there are clear resource constraints during a PhD programme. It is also important to source meaningful, realistic and relevant (industrial) problem instances. Good quality data is vital in training ML systems. Yet another challenge is the fact that each individual experiment is

time-consuming so gathering enough experimental data to train on may be difficult.

Acknowledgments

This work is supported by grant EP/R513386/1 from the UK Engineering and Physical Sciences Research Council. I am grateful to my supervisors for their support and guidance, and in particular to Peter Nightingale for the initial idea.

References

1. 2019 XCSP3 Competition (2019), http://www.cril.univ-artois.fr/XCSP19/
2. Ansótegui, C., Bofill, M., Coll, J., Dang, N., Esteban, J.L., Miguel, I., Nightingale, P., Salamon, A.Z., Suy, J., Villaret, M.: Automatic detection of at-most-one and exactly-one relations for improved SAT encodings of pseudo-boolean constraints. In: International Conference on Principles and Practice of Constraint Programming. pp. 20–36. Springer (2019). https://doi.org/10.1007/978-3-030-30048-7
3. Audemard, G., Boussemart, F., Lecoutre, C., Piette, C., Roussel, O.: XCSP3 and its ecosystem. Constraints (Feb 2020). https://doi.org/10.1007/s10601-019-09307-9
4. Bofill, M., Coll, J., Suy, J., Villaret, M.: SAT encodings of pseudo-boolean constraints with at-most-one relations. In: International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 112–128. Springer (2019). https://doi.org/10.1007/978-3-030-19212-9
5. Frisch, A.M.: The Design of ESSENCE: A Constraint Language for Specifying Combinatorial Problems. In: Proceedings of the Twentieth International Joint Conference on Artificial Intelligence. pp. 80–87. Hyderabad (2007)
6. Hurley, B., Kotthoff, L., Malitsky, Y., O'Sullivan, B.: Proteus: A Hierarchical Portfolio of Solvers and Transformations. In: Simonis, H. (ed.) Integration of AI and OR Techniques in Constraint Programming. pp. 301–317. Lecture Notes in Computer Science, Springer International Publishing, Cham (2014). https://doi.org/10.1007/978-3-319-07046-9
7. Joshi, S., Martins, R., Manquinho, V.: Generalized totalizer encoding for pseudo-boolean constraints. In: Pesant, G. (ed.) Principles and Practice of Constraint Programming. pp. 200–209. Springer International Publishing, Cham (2015), https://arxiv.org/pdf/1507.05920

8. Kotthoff, L.: Algorithm selection for combinatorial search problems: A survey. In: Bessiere, C., De Raedt, L., Kotthoff, L., Nijssen, S., O'Sullivan, B., Pedreschi, D. (eds.) Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary Approach, pp. 149–190. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-50137-6
9. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: MiniZinc: Towards a Standard CP Modelling Language. In: Bessière, C. (ed.) Principles and Practice of Constraint Programming – CP 2007. pp. 529–543. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74970-7
10. Nightingale, P., Akgün, Ö., Gent, I.P., Jefferson, C., Miguel, I., Spracklen, P.: Automatically improving constraint models in Savile Row. Artificial Intelligence **251**, 35–61 (Oct 2017). https://doi.org/10.1016/j.artint.2017.07.001
11. Nightingale, P., Rendl, A.: Essence' Description. ArXiv **abs/1601.02865** (2016)
12. Soos, M., Kulkarni, R., Meel, K.S.: CrystalBall: Gazing in the black box of SAT solving. In: International Conference on Theory and Applications of Satisfiability. vol. 22, pp. 371–387 (2019). https://doi.org/10.1007/978-3-030-24258-9
13. Van Hentenryck, P.: The OPL Optimization Programming Language. MIT Press, Cambridge, MA, USA (1999)
14. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla: Portfolio-based algorithm selection for SAT. Journal of artificial intelligence research **32**, 565–606 (2008). https://doi.org/10.1613/jair.2490
15. Zhou, N.F., Kjellerstrand, H.: The Picat-SAT Compiler. In: Gavanelli, M., Reppy, J. (eds.) Practical Aspects of Declarative Languages. pp. 48–62. Lecture Notes in Computer Science, Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-28228-2_4