CP 2019
25 YEARS

Stamford, CT, USA
September 30 – October 4 2019

Abstracts

Table of Contents

full paper published in J. Artif. Intell. Res. ***66***:*123–150 (2019)*

From Support Propagation to Belief Propagation in Constraint Programming

Gilles Pesant
Polytechnique Montréal, Montreal, Canada
CIRRELT, Université de Montréal, Montreal, Canada
gilles.pesant@polymtl.ca

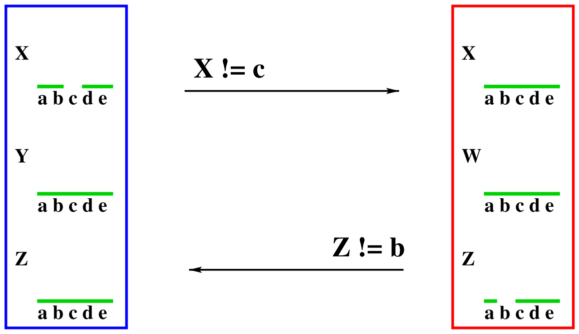

Figure 1: Standard propagation of unsupported values viewed as message passing between constraint $c_1(X, Y, Z)$ on the left and constraint $c_2(X, W, Z)$ on the right.

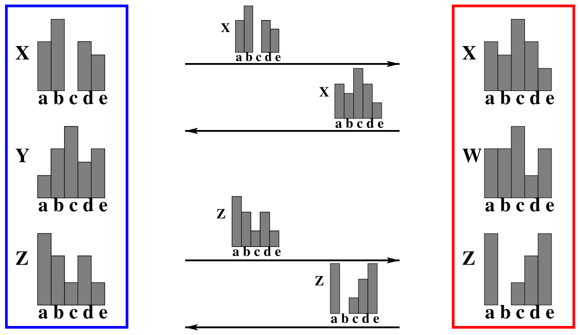

Figure 2: Exchanging frequency distributions over individual variables between constraints $c_1(X, Y, Z)$ and $c_2(X, W, Z)$.

The distinctive driving force of constraint programming to solve combinatorial problems has been a privileged access to problem structure through the high-level models it uses. From that exposed structure in the form of so-called global constraints, powerful inference algorithms have shared information between constraints by propagating it through shared variables' domains, traditionally by removing unsupported values. The paper proposes a richer propagation medium made possible by recent work on counting solutions inside constraints.

Many forms of message passing algorithms have been investigated in artificial intelligence, such as Belief Propagation (BP). Constraint Propagation can also be viewed as a message passing algorithm, announcing value deletions from domains through the constraint network and necessarily converging because some quantity, namely the size of the Cartesian product of the domains, decreases monotonically. From the perspective of message passing, the deleted values being propagated constitute rather flat information since all non-deleted values remain on an equal footing (see Fig. 1). One way to view a variable's filtered domain with respect to a constraint is as a set of variable-value pairs having non-zero frequency among its solution set — if instead we share the whole frequency distribution over individual variables (see Fig. 2) we can discriminate between values from the perspective of each constraint and, for example, use it for branching in the search tree. Since in general we don't have an explicit description of the solution set of a constraint, that frequency distribution is not readily available. An efficient implementation of belief propagation with global constraints needs to perform counting weighted by the beliefs about variables. Recent work on solution counting is bringing such computation within reach for several families of constraints.

In what I propose, beliefs (i.e. frequencies) about individual variable-value assignments are exchanged between constraints and iteratively adjusted. It generalizes standard (support) propagation and aims to converge to the true marginal distributions of the solutions over individual variables. Its advantage over standard belief propagation is that the higher-level models featuring large-arity (global) constraints do not tend to create as many cycles, which are known to be problematic for the convergence of BP.
Consider the following example to illustrate the approach:

i. $\texttt{alldifferent}(a, b, c)$
ii. $a + b + c + d = 7$
iii. $c \leq d$

three constraints with variables a, b, c, $d \in \{1, 2, 3, 4\}$. There are two solutions to that CSP: $(a = 2, b = 3, c = 1, d = 1)$ and $(a = 3, b = 2, c = 1, d = 1)$. Even if we enforce domain consistency on each constraint, no filtering occurs. To solve this CSP we would thus be left to branch on variables having identical domains.

	1	2	3	4
θ_a	0	1/2	1/2	0
θ_b	0	1/2	1/2	0
θ_c	1	0	0	0
θ_d	1	0	0	0

	1	2	3	4
θ_a^i	1/4	1/4	1/4	1/4
θ_a^{ii}	10/20	6/20	3/20	1/20
θ_b^i	1/4	1/4	1/4	1/4
θ_b^{ii}	10/20	6/20	3/20	1/20
θ_c^i	1/4	1/4	1/4	1/4
θ_c^{ii}	10/20	6/20	3/20	1/20
θ_c^{iii}	4/10	3/10	2/10	1/10
θ_d^{ii}	10/20	6/20	3/20	1/20
θ_d^{iii}	1/10	2/10	3/10	4/10

Table 1: True marginals (left) and local marginals (right).

	1	2	3	4
θ_a	.25	.25	.25	.25
θ_b	.25	.25	.25	.25
θ_c	.25	.25	.25	.25
θ_d	.25	.25	.25	.25

	1	2	3	4
θ_a	.50	.30	.15	.05
θ_b	.50	.30	.15	.05
θ_c	.62	.28	.09	.01
θ_d	.29	.34	.26	.11

	1	2	3	4
θ_a	.12	.41	.40	.07
θ_b	.12	.41	.40	.07
θ_c	.84	.15	.01	.00
θ_d	.65	.28	.06	.01

	1	2	3	4
θ_a	.01	.52	.46	.01
θ_b	.01	.52	.46	.01
θ_c	.98	.02	.00	.00
θ_d	.90	.10	.00	.00

Table 2: Initial marginals (top left) and computed marginals after 1st (top right), 5th (bottom left), and 10th (bottom right) iteration.

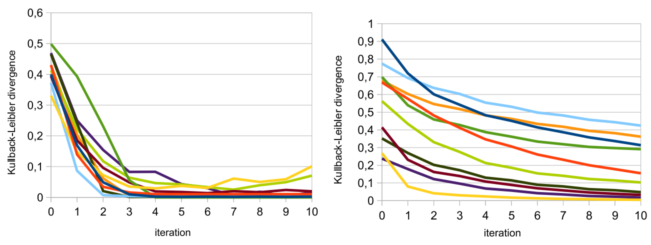

Figure 3: Average Kullback-Leibler divergence of computed variable marginals with respect to the true marginals as we iterate message passing between constraints and variables for sets of Partial Latin Square (left) and Rostering (right) instances. Each line corresponds to an instance.

Variable a takes value 2 in one of the two solutions, value 3 in one solution as well, and values 1 and 4 in no solution. If we look at the set of solutions as a multivariate discrete distribution, its projection onto a yields $\langle 0, 1, 1, 0 \rangle$ and under the assumption that either solution is equally likely the marginal probability distribution for a is then $\theta_a = \langle 0, 1/2, 1/2, 0 \rangle$. Table 1 gives on the left the true marginal for each variable and on the right the marginals local to each constraint taken individually and over its own set of solutions. Table 2 presents the evolution of the computed marginals as the number of belief propagation iterations performed increases. Note how these get closer to the true marginals in Table 1.

The paper describes the necessary architectural changes

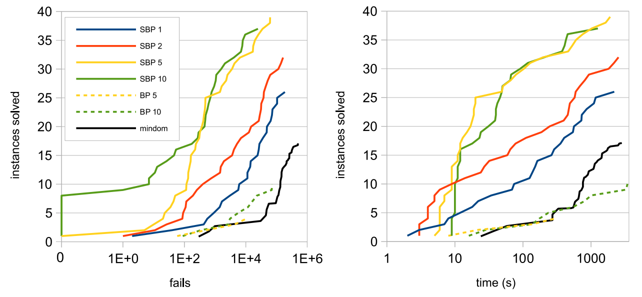

Figure 4: Number of 30×30 Partial Latin Square instances solved with respect to the number of fails (left) and computation time (right). Label "SBP k" indicates that standard propagation followed by k iterations of belief propagation are performed at each search-tree node prior to branching.

to a constraint programming solver, provides weighted-counting algorithms for a few common constraints, and conducts an empirical study of the proposal using its publicly-available implementation[1] on top of MiniCP. One finds that it provides close approximations to the true marginals (see e.g. Fig. 3) and that it significantly improves both search guidance and solving time (see e.g. Fig. 4).

[1]`https://github.com/PesantGilles/MiniCPBP`

Searching for Input Data that Exercise Maximal Errors in Floating-Point Computations[*]

Rémy Garcia, Claude Michel, Michel Rueher
Université Côte d'Azur, CNRS, I3S, France
firstname.surname@i3s.unice.fr

Overview

Floating-point computations involve errors due to rounding operations. An error characterizes the distance between the intended computations over the reals and the actual computations over the floats. Such errors impact the precision and the stability of computations and can lead to an execution path over the floats that is significantly different from the expected path over the reals. A faithfull account of these errors is mandatory to capture the actual behaviour of a program with floating-point computations.

Efficient tools exist for error analysis that rely on an over-approximation of the error in programs with floating-point computations, e.g., Fluctuat (Goubault and Putot 2011; 2006), an abstract interpreter which combines affine arithmetic and zonotopes to analyze the robustness of programs over floats, FPTaylor (Solovyev et al. 2015) which uses symbolic taylor expansions to compute tight bounds of the error, and PRECiSA (Moscato et al. 2017; Titolo et al. 2018), a more recent tool that relies on static analysis to compute a certificate of proof that can be used to formally prove the round-off error bounds of a program.

The point is that all these tools only provide *over-approximated* bounds of an error. In other words, there is no guarantee that such bounds are reachable or even that it is possible to get close to their values. Over-approximation may generate *false positives*, i.e. reports that an assertion might be violated even so in practice none of the input values reach the related case. To get rid of *false positives*, maximal errors, i.e., greatest reachable errors, must be computed. Finding a maximal error is the goal of this work.

Approach

Our approach is based on a branch-and-bound algorithm that maximizes a given error of a program with floating-point computations. The branch-and-bound algorithm is embedded in a solver over the floats (Zitoun et al. 2017; Marre and Michel 2010; Botella, Gotlieb, and Michel 2006; Michel 2002; Michel, Rueher, and Lebbah 2001) extended to constraints over the errors (Garcia et al. 2018). Our system, called FErA (**F**loating-point **Er**ror **A**nalyzer), provides not only the maximal error but also input values to reach it.

Finding the maximal error is a hard task since the operation errors are unevenly distributed. One advantage of our approach is that the branch-and-bound is an anytime algorithm: even if it is stopped before the maximal error is found, it provides input values for a reachable error.

Experiments

Preliminary experiments on benchmarks from the FPBench (Damouche et al. 2017) suite show that our approach compares well with state of the art systems. Table 1 compares results from Gappa (Daumas and Melquiond 2010), Fluctuat (Goubault and Putot 2011; 2006), Real2Float (Magron, Constantinides, and Donaldson 2017), FPTaylor (Solovyev et al. 2015), and PRECiSA (Moscato et al. 2017; Titolo et al. 2018) to FErA on a subset of the FPBench benchmarks (Moscato et al. 2017).

The two last columns detail results provided by FErA: column *filtering* gives an over-approximation computed by a single filtering while column *optimizer* provides the best reachable error e^* and over-approximation $\overline{e}$ computed in less than 10 seconds by the branch-and-bound. Bold and italic are used to rank the best and second best over-approximation among all the tools and the filtering result of FErA, while red indicates the worst result. On these benchmarks, FErA filtering classified as best four times and second two times and never provides the worst result. Despite the time limitation, FErA computes often the best over-approximation $\overline{e}$ (in green). In all cases but one, the computed reachable error e^* is in the same order of magnitude than $\overline{e}$.

Conclusion

To sum up, this work addresses a critical issue in program verification: computing input values that reach a maximal error in floating-point computations. To compute these values, we introduce an original approach based on a branch-and-bound algorithm and the constraint system for round-off error analysis from (Garcia et al. 2018). Further works include a better understanding and a tighter representation of round-off errors on elementary operations, experimentations with different search strategies dedicated to floating-point numbers from (Zitoun 2018) to improve the resolution process,

[*]This work was partially supported by ANR COVERIF (ANR-15-CE25-0002)

Table 1: Experimental results for absolute round-off error bounds (bold indicates the best approximation, italic indicates the second best, and red indicates the worst one. Green indicates the best approximation computed by the branch-and-bound.)

	Gappa	Fluctuat	Real2Float	FPTaylor	PRECiSA	FErA		
							optimizer	
						filtering	e^*	$\bar{e}$
carbonGas	2.61e-08	4.51e-08	2.21e-08	*8.06e-09*	**7.32e-09**	1.90e-08	2.53e-09	**4.88e-09**
verhulst	4.18e-16	5.51e-16	4.66e-16	**2.47e-16**	2.91e-16	*2.51e-16*	1.50e-16	**1.66e-16**
predPrey	2.04e-16	2.49e-16	2.51e-16	*1.59e-16*	1.77e-16	**1.04e-16**	7.93e-17	**9.81e-17**
rigidBody1	*2.95e-13*	3.22e-13	5.33e-13	*2.95e-13*	*2.95e-13*	**2.27e-13**	1.44e-13	**2.27e-13**
rigidBody2	3.61e-11	3.65e-11	6.48e-11	3.61e-11	*3.60e-11*	**2.27e-11**	1.62e-11	**2.27e-11**
doppler1	2.02e-13	3.90e-13	7.65e-12	**1.58e-13**	*1.99e-13*	3.63e-13	3.98e-14	1.86e-13
doppler2	3.92e-13	9.75e-13	1.57e-11	**2.89e-13**	*3.83e-13*	1.02e-12	7.88e-14	3.48e-13
doppler3	1.08e-13	1.57e-13	8.59e-12	**6.62e-14**	*1.05e-13*	1.42e-13	2.73e-14	8.85e-14
turbine1	8.40e-14	9.20e-14	2.46e-11	**1.67e-14**	*2.33e-14*	1.22e-13	4.42e-15	**1.32e-14**
turbine2	1.28e-13	1.29e-13	2.07e-12	**1.95e-14**	*3.07e-14*	1.51e-13	5.82e-15	**1.58e-14**
turbine3	3.99e+01	6.99e-14	1.70e-11	**9.64e-15**	*1.72e-14*	7.82e-14	2.68e-15	**7.74e-15**
sqroot	5.71e-16	6.83e-16	1.28e-15	*5.02e-16*	**4.29e-16**	4.79e-16	3.96e-16	4.79e-16
sine	1.13e-15	7.97e-16	6.03e-16	**4.43e-16**	5.96e-16	*4.63*e-16	2.32e-16	**4.07e-16**
sineOrder3	8.89e-16	1.15e-15	1.19e-15	*5.94e-16*	1.11e-15	**5.07e-16**	2.34e-16	**3.37e-16**

as well as devising a local search to speed up the resolution process.

References

Botella, B.; Gotlieb, A.; and Michel, C. 2006. Symbolic execution of floating-point computations. *Software Testing, Verification and Reliability* 16(2):97–121.

Damouche, N.; Martel, M.; Panchekha, P.; Qiu, C.; Sanchez-Stern, A.; and Tatlock, Z. 2017. Toward a standard benchmark format and suite for floating-point analysis. In *9th International Workshop on Numerical Software Verification (NSV2017)*, 63–77.

Daumas, M., and Melquiond, G. 2010. Certification of bounds on expressions involving rounded operators. *ACM Trans. Math. Softw.* 37(1):2:1–2:20.

Garcia, R.; Michel, C.; Pelleau, M.; and Rueher, M. 2018. Towards a constraint system for round-off error analysis of floating-point computation. In *24th International Conference on Principles and Practice of Constraint Programming :Doctoral Program*.

Goubault, E., and Putot, S. 2006. Static analysis of numerical algorithms. In *Static Analysis, 13th International Symposium, SAS 2006, Seoul, Korea, August 29-31, 2006, Proceedings*, volume 4134 of *Lecture Notes in Computer Science*, 18–34.

Goubault, E., and Putot, S. 2011. Static analysis of finite precision computations. In *12th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2011)*, 232–247.

Magron, V.; Constantinides, G. A.; and Donaldson, A. F. 2017. Certified roundoff error bounds using semidefinite programming. *ACM Trans. Math. Softw.* 43(4):34:1–34:31.

Marre, B., and Michel, C. 2010. Improving the floating point addition and subtraction constraints. In *Proceedings of the 16th international conference on Principles and practice of constraint programming (CP'10)*, LNCS 6308, 360–367.

Michel, C.; Rueher, M.; and Lebbah, Y. 2001. Solving constraints over floating-point numbers. In *7th International Conference on Principles and Practice of Constraint Programming (CP 2001)*, 524–538.

Michel, C. 2002. Exact projection functions for floating point number constraints. In *AI&M 1-2002, Seventh international symposium on Artificial Intelligence and Mathematics (7th ISAIM)*.

Moscato, M.; Titolo, L.; Dutle, A.; and Muñoz, C. A. 2017. Automatic estimation of verified floating-point round-off errors via static analysis. In *Computer Safety, Reliability, and Security*, 213–229.

Solovyev, A.; Jacobsen, C.; Rakamarić, Z.; and Gopalakrishnan, G. 2015. Rigorous estimation of floating-point round-off errors with Symbolic Taylor Expansions. In Bjørner, N., and de Boer, F., eds., *Proceedings of the 20th International Symposium on Formal Methods (FM)*, volume 9109 of *Lecture Notes in Computer Science*, 532–550. Springer.

Titolo, L.; Feliú, M. A.; Moscato, M. M.; and Muñoz, C. A. 2018. An abstract interpretation framework for the round-off error analysis of floating-point programs. In *Verification, Model Checking, and Abstract Interpretation - 19th International Conference, VMCAI 2018, Los Angeles, CA, USA, January 7-9*, 516–537.

Zitoun, H.; Michel, C.; Rueher, M.; and Michel, L. 2017. Search strategies for floating point constraint systems. In *23rd International Conference on Principles and Practice of Constraint Programming, CP 2017*, 707–722.

Zitoun, H. 2018. *Search strategies for solving constraint systems over floats for program verification*. Theses, Université Côte d'Azur.

Constraint-based Sequential Pattern Mining with Decision Diagrams (Abstract*)

Amin Hosseininasab
Tepper School of Business,
Carnegie Mellon University, USA
aminh@andrew.cmu.edu

Willem-Jan van Hoeve
Tepper School of Business,
Carnegie Mellon University, USA
vanhoeve@andrew.cmu.edu

Andre A. Cire
Dept. of Management,
University of Toronto Scarborough, Canada
andre.cire@rotman.utoronto.ca

Motivation

Sequential Pattern Mining (SPM) is a fundamental data mining task with a large array of applications in marketing, health care, finance, and bioinformatics, to name a few. Frequent patterns are used, e.g., to extract knowledge from data within decision support tools, to develop novel association rules, and to design more effective recommender systems. We refer the reader to (Fournier-Viger et al. 2017) for a recent and thorough review of SPM and its applications.

In practice, mining the entire set of frequent patterns in a database is not of interest, as the resulting number of items is typically large and may provide no significant insight to the user. It is hence desirable to restrict the mining algorithm search to smaller subsets of patterns that satisfy problem-specific constraints. For example, in online retail click-stream analysis, we may seek frequent browsing patterns from sessions where users spend at least a minimum amount of time on certain items that have specific price ranges. Such constraints limit the output of SPM and are much more effective in knowledge discovery, as compared to an arbitrary large set of frequent click-streams.

A naïve approach to impose constraints in SPM is to first collect all unconstrained frequent patterns, and then to apply a post-processing step to retain patterns that satisfy the desired constraints. This approach, however, may be expensive in terms of memory requirements and computational time, in particular when the resulting subset of constrained patterns is small in comparison to the full unconstrained set. Constraint-based sequential pattern mining (CSPM) aims at providing more efficient methods by embedding constraint reasoning within existing mining algorithms (Pei, Han, and Wang 2007; Negrevergne and Guns 2015; Aoga, Guns, and Schaus 2017). Nonetheless, while certain constraint types are relatively easy to incorporate in a mining algorithm, others of practical use are still challenging to handle in a general and effective way. This is particularly the case of non-monotone constraints representing, e.g., sums and averages of attributes.

*The full paper has been published as "A. Hosseininasab, W.-J. van Hoeve, and A. A. Cire. Constraint-Based Sequential Pattern Mining with Decision Diagrams. *Proceedings of AAAI*, pp. 1495–1502. AAAI Press, 2019."

Contributions

In this paper, we propose a novel representation of sequential databases using a multi-valued decision diagram (MDD), a graphical model that compactly encodes the sequence of items and their attributes by leveraging symmetry. The MDD representation can be augmented with constraint-specific information, so that constraint satisfaction is either guaranteed or enforced during the mining algorithm. As a proof of concept, we implement a general prefix-projection algorithm equipped with an MDD to enforce several constraint types, including complex constraints such as average ("avg") and median ("md"). To the best of our knowledge, this paper is the first to consider the "sum," "avg," and "md" constraints with arbitrary item-attribute association within the pattern mining algorithm.

To evaluate the applicability of our approach, we compare its performance against a typical generate-and-check variant, as well as a state-of-the-art constraint-based sequential pattern mining algorithm. Results on real-world benchmark databases show that our approach is competitive with or superior to these other methods in terms of efficiency as well as scalability.

References

Aoga, J. O. R.; Guns, T.; and Schaus, P. 2017. Mining Time-constrained Sequential Patterns with Constraint Programming. *Constraints* 22:548–570.

Fournier-Viger, P.; Lin, J. C.-W.; Kiran, R. U.; Koh, Y. S.; and Thomas, R. 2017. A survey of sequential pattern mining. *Data Science and Pattern Recognition* 1(1):54–77.

Negrevergne, B., and Guns, T. 2015. Constraint-based sequence mining using constraint programming. In *Proceedings of CPAIOR*, volume 9075 of *LNCS*, 288–305. Springer.

Pei, J.; Han, J.; and Wang, W. 2007. Constraint-based sequential pattern mining: the pattern-growth methods. *Journal of Intelligent Information Systems* 28(2):133–160.

Nmbr9 as a Constraint Programming Challenge

Mikael Zayenz Lagerkvist[0000−0003−2451−4834]
research@zayenz.se, https://zayenz.se

1 Introduction

Modern board games are a rich source of interesting and new challenges for combinatorial problems. The game Nmbr9 is a solitaire style puzzle game using polyominoes. The rules of the game are simple to explain, but modelling the game effectively using constraint programming is hard.

This abstract presents the game, contributes new generalized variants of the game suitable for benchmarking and testing, and describes a model for the presented variants. The question of the top possible score in the standard game is an open challenge.

2 Nmbr9

Nmbr9 (Wichmanm 2017) is a solitaire puzzle game using polyominoes that can be played by multiple people at the same time. The goal of the game is to build layers of the polyominoes in Figure 1 according to a pre-defined but unknown shuffle, and to place polyominoes representing larger numbers higher up to score points.

Rules

The game consists of polyominoes representing the numbers 0 to 9, two copies per number for each player. A common deck of 20 cards with the polyominoes are shuffled.

When a card is drawn, each player takes the corresponding part and places it in their own area. Placements are done in levels, where the first level (level 1) is directly on the table, and level n is on top of level $n - 1$.

Placement must always be done fully supported (level 1 is always fully supported, level n is supported by previously placed parts in level $n - 1$). Each level must be fully 4-connected (i.e., not diagonally) after placement. The final requirement on placements is that when placing a part on top of other parts, it must be on top of at least two different parts. For example, if a 9-part is in level n, then an 8-part can not be placed in level $n + 1$ resting solely on the 9-part.

When all cards have been drawn, the score is computed as the sum of the value for parts times their level minus one. For example, an 8-part on level 3 is worth 16, while it is worth 0 on level 1.

Variants

We define variants of Nmbr9 to get a larger sampling of problems to solve, from small trivial problems to larger intractable problems. Let $T{-}m{-}c{-}n$ be the scheme, where

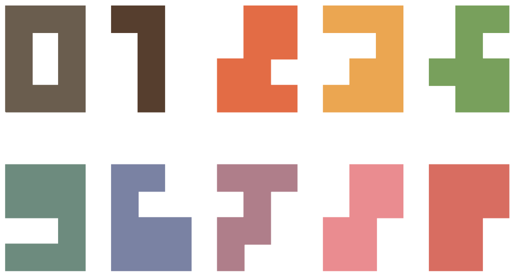

Figure 1: Parts to place representing the numbers 0-9.

- $T \in U, F, K$ Whether the draft is *unknown*, *free* to choose, or *known*. The first corresponds to a quantified problem, where the draft is $\forall$-quantified. The second models the question of the max possible score, while the last models the max possible score given a specific known shuffle.
- m The maximum value to use, from 0-9.
- c The number of available copies of each polyomino.
- k The number of cards in the deck to use, $k \leq m \cdot c$ must hold.

With this taxonomy of variants, the standard game is $U{-}9{-}2{-}20$, where all available parts are used. Reducing m, c, or k are all effective ways to make the model smaller. F variants are interesting for answering the question of the top-score possible. K variants are interesting computationally because they represent a much simpler but still hard problem, and are also interesting recreationally given a finished game, where participants wonder how they compare to the best possible result for that particular shuffle.

At Board Game Geek, a forum thread on solving the standard $F{-}9{-}2{-}20$ instance has a top score of 229 points, using a total of 7 layers (Kuenzler 2018).

3 Constraint programming model

In this section, a constraint programming model is described for Nmbr9 in the F and K variants; a quantified model for the U variant could be built as an extension on top of this. The model is based on the models in (Lagerkvist and Pesant 2008; Lagerkvist 2019), where `regular` constraints are used for placement of polyominoes in grids.

Nmbr9 is played on a conceptually infinite grid, with as many layers as needed available. The model used here requires a fixed grid. In practice, placements very seldom surpass 20 squares wide, and more than 7 layers is unlikely. The width/height size s and the number of layers $l_\top$ are parameters for the model. A core problem is keeping levels connected. This is accomplished using the technique of defining an area around parts, here as a new value. Also, parts may not be placed, which is controlled using reification of the regular expression. As an example, the regular expression for part 0 (called R_0) in its two rotations, where the value 0 is an empty square, 1 is a square occupied by the part, and 2 is an area surrounding the part becomes (including initial control variable for reified placement)

$$10^*(2^3 0^{s-4} 2 1^3 2 0^{s-5} (2 1 0 1 2 0^{s-5})^2 2 1^3 2 0^{s-4} 2^3 \mid \\ 2^4 0^{s-5} 2 1^4 2 0^{s-6} 2 1 0^2 1 2 0^{s-6} 2 1^4 2 0^{s-5} 2^4) 0^* \mid 00^*$$

Let $P = \{p_1, \ldots, p_n\}$ ($n = m \cdot c$) be the set of parts, and $v(p) \in P \to 0..m$ the value of each part. Variables $D = \langle d_1, \ldots, d_k \rangle \in P$ represent the deck of polyominoes, and $O = \langle o_1, \ldots, o_n \rangle \in \{1..n\}$ the order of the polyominoes as they occur in the deck, with values above k representing not used. The matrix G_l is the grid for level l for all parts on that level (of size $s \times s$, domain $0..n$, border is 0), and G_{lp} the grid for a part p on level l (of similar size, domain $0..2$). Variables G^1_{lp} and G^2_{lp} are Boolean matrices representing when G_{lp} is 1 and 2 respectively. Boolean variables L_{pl} represent p being placed on level l, integer variable $L_p \in 0..l_\top$ the level of p (if any, 0 if not), and Y_p a Boolean variable representing that p is placed with N_p the inverse. The following defines the constraints of the model, where p, p' is implicitly assumed to range over P with $p \neq p'$, l over $1..l_\top$, l' over $2..l_\top$, and i, j over $1..s$. Operators $\dot\vee$ and $\dot\wedge$ are used for point-wise logical operations. The notation $[M]$ is used to indicate true iff any element in the matrix M is true.

$$\texttt{global\_cardinality}(D, \langle Y_1, \ldots, Y_n \rangle) \tag{1}$$

$$\texttt{regular}(R_p, L_{pl} G_{lp}) \tag{2}$$

$$\texttt{inverse}(O, \langle D_1, \ldots, D_k, E_{k+1}, \ldots, E_n \rangle), E \in P \tag{3}$$

$$B(p, p') \leftrightarrow O(p) < O(p'),\ O(p) \leq k \leftrightarrow Y_p \tag{4}$$

$$\texttt{int\_to\_bool}(L_p, \langle N_p, L_{p1}, \ldots, L_{pl_\top} \rangle), \tag{5}$$

$$Y_p = 1 - N_p, \tag{6}$$

$$G_{lp}(i,j) = 1 \leftrightarrow G^1_{lp}(i,j), G_{lp}(i,j) = 2 \leftrightarrow G^2_{lp}(i,j) \tag{7}$$

$$G_l(i,j) = p \leftrightarrow G_{lp}(i,j) = 1 \tag{8}$$

$$\left(L_{pl} \wedge \exists_{p''|B(p'',p)} L_{p''l}\right) \to \left[G^2_{lp} \dot\wedge \left(\dot\vee_{p'|B(p',p)} G^1_{lp'}\right)\right] \tag{9}$$

$$G^1_{l'p}(i,j) \to \vee_{p'|B(p',p)} G^1_{l'-1p'}(i,j) \tag{10}$$

$$L_{pl'} \to \left(2 \leq \textstyle\sum_{p'|B(p',p)} [G^1_{lp} \dot\wedge G^1_{l-1p'}]\right) \tag{11}$$

$$S = \textstyle\sum_{p \in P} (L_p - Y_p) \cdot v(p) \tag{12}$$

Constraint 1 checks that the deck is a shuffle of parts. Constraint 2 is the core placement constraints, with $l_\top \cdot n$ regular constraints in total. Constraints 3 to 8 channel information between variables. Constraints 9 to 11 implement the requirements for connectedness and being placed supported and on top of at least two different parts. When $k = n$, then constraint 1 is equivalent to an `alldifferent`, and in constraint 3 there are no extra E variables representing fake deck placement for non-used parts.

Finally, constraint 12 sums up the score of the solution, where $L_p - Y_p$ is 0 for non-used parts ($L_p = 0$ and $Y_p = 0$), and also for parts in the first layer ($L_p = 1$ and $Y_p = 1$).

4 Implementation

The described model has been implemented using Gecode (Gecode team 2018) version 6.2.0 and C++17, and can be accessed at github.com/zayenz/cp-2019-nmbr9.

Choices are made on the deck, then the levels of parts, then the placements. Placements are made using a static variable order spiraling from the center. Implied constraints that levels require two cards before the next level can start and that the area for each level is at most the area for the level below are added. In addition, symmetry breaking on value precedence for same values in the deck and rotation symmetry on the base grid values is added.

It is intractable to find a max score for the standard game approaching the score found manually using this model and search strategy; the model has more than 2.5 million propagators, and setting up the root node takes 5 seconds. A small problem like $F{-}6{-}2{-}5$ with 3 levels and grid size 8 takes almsot 3 minutes and 140k failures to solve.

5 Conclusions

As seen, modelling Nmbr9 is surprisingly complex, mainly due to the connectedness and support requirements. This leads to a large model with not much propagation. Finding the maximum score for the standard game is a computationally hard open challenge.

Future work include: exploring how large problems can be solved with the current model; investigating better models for stronger propagation; and finding effective heuristics for the search. For search heuristics, the *propagation guided global regret* in (Lagerkvist 2019) might be very interesting.

Acknowledgments Thanks to Magnus Gedda, Magnus Rattfeldt, and Martin Butler for interesting and informative discussions on games and search methods.

References

Gecode team. 2018. Gecode, the generic constraint development environment. http://www.gecode.org/.

Kuenzler, P. 2018. Board Game Geek – Nmbr9 Forums, What is the highest possible score? https://boardgamegeek.com/article/30607221#30607221, [Accessed 2019-08-27].

Lagerkvist, M. Z., and Pesant, G. 2008. Modeling irregular shape placement problems with regular constraints. In *First Workshop on Bin Packing and Placement Constraints BPPC'08*.

Lagerkvist, M. Z. 2019. State representation and polyomino placement for the game patchwork. In *The 18th workshop on Constraint Modelling and Reformulation*.

Wichmanm, P. 2017. NMBR 9.

Scalable Approximate Model Counting via Concentrated Hashing (Extended Abstract)[*]

Kuldeep S. Meel[1] **and S. Akshay**[2]
[1] School of Computing, National University of Singapore, Singapore
[2] Dept of CSE, Indian Institute of Technology, Bombay

Given a Boolean formula F in conjunctive normal form (CNF), the problem of model counting, also referred to as $\#SAT$, is to compute the number of models of F. Model counting is a fundamental problem in computer science with a wide variety of applications ranging from probabilistic reasoning, network reliability, quantified information leakage, and the like.

In his seminal paper, Valiant showed that $\#\mathsf{SAT}$ is $\#\mathsf{P}$-complete, where $\#\mathsf{P}$ is the set of counting problems associated with NP decision problems (Valiant 1979). Given the computational intractability of #SAT, researchers have focused on approximate variants. Stockmeyer presented a randomized hashing-based technique that can compute (ε, δ) approximation within the polynomial time, in $|F|, \varepsilon, \delta$, given access to a NP oracle where $|F|$ is the size of formula, ε is the error tolerance bound and δ is the confidence. The computational intractability of NP dissuaded development of algorithmic implementations of Stockmeyer's hashing-based techniques and no practical tools for approximate counting existed until the 2000's (Gomes, Sabharwal, and Selman 2006). By extending Stockmeyer's framework, Chakraborty, Meel, and Vardi demonstrate a scalable (ε, δ)-counting algorithm, ApproxMC (Chakraborty, Meel, and Vardi 2013). Subsequently, several new algorithmic ideas have been incorporated to demonstrate the scalability of ApproxMC; the current version of ApproxMC is called ApproxMC3 (Chakraborty, Meel, and Vardi 2016; Soos and Meel 2019). Recent years have seen a surge of interest in the design of hashing-based techniques for approximate counting (Ermon et al. 2013a; 2013b; Chakraborty et al. 2014; Ivrii et al. 2015; Meel et al. 2016; Belle, Van den Broeck, and Passerini 2015; Chakraborty et al. 2016). We refer the reader to (Meel 2017) for detailed commentary on hashing-based techniques for approximate counting.

The core theoretical idea of the hashing-based framework is to employ 2-universal hash functions to partition the solution space, denoted by $sol(F)$ for a formula F, into *roughly equal small* cells, wherein a cell is called *small* if it has solutions less than or equal to a pre-computed threshold, thresh. A NP oracle is employed to check if a cell is small by enumerating solutions one-by-one until either there are no more solutions or we have already enumerated $\mathsf{thresh} + 1$ solutions.

A standard family of 2-universal hash functions employed for this is the H_{xor} family comprising of functions expressed as conjunction of XOR constraints. In particular, viewing the set of variables Y of the formula F as a vector of dimension $n \times 1$, one can represent the hash function $h : \{0,1\}^n \mapsto \{0,1\}^m$ as $h(Y) = \boldsymbol{A}Y + \boldsymbol{b}$ where $\boldsymbol{A}$ is a $m \times n$ matrix while $\boldsymbol{b}$ is $m \times 1$ 0-1 vector and each entry of $\boldsymbol{A}$ and $\boldsymbol{b}$ is either 0 or 1. Now, the solutions of F in a given cell α are the solutions of the formula $F \wedge (\boldsymbol{A}Y + \boldsymbol{b} = \alpha)$. As the input formula F is in CNF, this formula is a conjunction of CNF and XOR-constraints, also called an CNF-XOR formula. Given a hash function h and a cell α, the random variable of interest, denoted by $|\mathsf{Cell}_{\langle F,h,\alpha \rangle}|$ is the number of solutions of F that h maps to cell α.

The practical implementation of these techniques employ a SAT solver to perform NP oracle calls. The performance of SAT solvers, however, degrades with increase in the number of variables in XOR constraints (also called their *width*) and therefore recent efforts have focused on design of *sparse* hash functions where each entry is chosen with $p \ll 1/2$ (p is also referred to as density) (Gomes et al. 2007; Ermon et al. 2013b; Ivrii et al. 2015; Asteris and Dimakis 2016; Achlioptas, Hammoudeh, and Theodoropoulos 2018; Achlioptas and Theodoropoulos 2017). The primary theoretical challenge is that 2-universality has been crucial to obtain (ε, δ)-guarantees, and sparse hash functions are not 2-universal. In fact, despite intense theoretical and practical interest in the design of sparse hash functions, the practical implementation of all the prior constructions have had to sacrifice theoretical guarantees.

Given the applications of counting to critical domains such as network reliability, the loss of theoretical guarantees limits the applications of approximate model counters. Therefore, in this context, the main challenge is: *Is it possible to construct sparse hash functions and design algorithmic frameworks to achieve runtime performance improvement without losing theoretical guarantees?*

In this paper, we address this challenge. We introduce a

[*]The corresponding full version of the paper is under submission. Given that ApproxMC was introduced in CP 2013; we believe that this is appropriate venue for announcing the result.

new family of hash functions, denoted by $\mathcal{H}_{1.1}^{Rennes}$, which consists of hash functions of the form $\boldsymbol{A}X + \boldsymbol{b}$, where every entry of $\boldsymbol{A}[i]$ is set to 1 with $p_i = \mathcal{O}(\frac{\log_2 i}{i})$. The construction of the new family marks a significant departure from prior families in the behavior of the density dependent on rows of the matrix $\boldsymbol{A}$. We believe $\mathcal{H}_{1.1}^{Rennes}$ is of independent interest and can be substituted for 2-universal hash functions in several applications of hashing.

In order to do so we develop a new family of hash functions, called *concentrated hash functions* and use them instead of 2-universal hash functions to develop an algorithm, called SparseScalMC, for approximate model counting with (ε, δ)-guarantees. We design sparse hash functions belong to this concentrated hash family and employ them in SparseScalMC to achieve dramatic runtime improvements. we use the above concentrated hash family to develop a new approximate model counting algorithm SparseScalMC, building on the existing state-of-the-art algorithm ApproxMC3. A comprehensive experimental evaluation on 1896 benchmarks with computational effort over 20,000 computational hours demonstrates that usage of $\mathcal{H}_{1.1}^{Rennes}$ in SparseScalMC leads to up to $10\times$ speedup in runtime over ApproxMC3. It is worth viewing the runtime improvement in the context of prior work where significant slowdown was observed. To the best of our knowledge, *this work is the first study to demonstrate runtime improvement through sparse hash functions without loss of $(\varepsilon, \delta)-$guarantees, demonstrating the tightness of our bounds in practice.*

References

Achlioptas, D., and Theodoropoulos, P. 2017. Probabilistic model counting with short xors. In *International Conference on Theory and Applications of Satisfiability Testing*, 3–19. Springer.

Achlioptas, D.; Hammoudeh, Z.; and Theodoropoulos, P. 2018. Fast and flexible probabilistic model counting. In *International Conference on Theory and Applications of Satisfiability Testing*, 148–164. Springer.

Asteris, M., and Dimakis, A. G. 2016. Ldpc codes for discrete integration. Technical report, Technical report, UT Austin.

Belle, V.; Van den Broeck, G.; and Passerini, A. 2015. Hashing-based approximate probabilistic inference in hybrid domains. In *Proc. of UAI*.

Chakraborty, S.; Fremont, D. J.; Meel, K. S.; Seshia, S. A.; and Vardi, M. Y. 2014. Distribution-aware sampling and weighted model counting for SAT. In *Proc. of AAAI*, 1722–1730.

Chakraborty, S.; Meel, K. S.; Mistry, R.; and Vardi, M. Y. 2016. Approximate probabilistic inference via word-level counting. In *Proc. of AAAI*.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2013. A scalable approximate model counter. In *Proc. of CP*, 200–216.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2016. Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls. In *Proc. of IJCAI*.

Ermon, S.; Gomes, C.; Sabharwal, A.; and Selman, B. 2013a. Embed and project: Discrete sampling with universal hashing. In *Proc. of NIPS*, 2085–2093.

Ermon, S.; Gomes, C. P.; Sabharwal, A.; and Selman, B. 2013b. Taming the curse of dimensionality: Discrete integration by hashing and optimization. In *Proc. of ICML*, 334–342.

Gomes, C. P.; Hoffmann, J.; Sabharwal, A.; and Selman, B. 2007. Short xors for model counting: from theory to practice. In *Proc. of SAT*, 100–106.

Gomes, C. P.; Sabharwal, A.; and Selman, B. 2006. Model counting: A new strategy for obtaining good bounds. In *Proc. of AAAI*, volume 21, 54–61.

Ivrii, A.; Malik, S.; Meel, K. S.; and Vardi, M. Y. 2015. On computing minimal independent support and its applications to sampling and counting. *Constraints* 1–18.

Meel, K. S.; Vardi, M.; Chakraborty, S.; Fremont, D. J.; Seshia, S. A.; Fried, D.; Ivrii, A.; and Malik, S. 2016. Constrained sampling and counting: Universal hashing meets sat solving. In *Proc. of Beyond NP Workshop*.

Meel, K. S. 2017. *Constrained Counting and Sampling: Bridging the Gap between Theory and Practice*. Ph.D. Dissertation, Rice University.

Soos, M., and Meel, K. S. 2019. Bird: Engineering an efficient cnf-xor sat solver and its applications to approximate model counting. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)(1 2019)*.

Valiant, L. 1979. The complexity of enumeration and reliability problems. *SIAM Journal on Computing* 8(3):410–421.

Constrained Logistic Regression to Avoid Undesirable Predictions

David Bergman[1a], **Mohsen Emadikhiav**[1b], **Serdar Kadioglu**[2*]
[1]Department of Operations and Information Management, School of Business, University of Connecticut
2100 Hillside Rd., Storrs, CT 06269
[a]david.bergman@uconn.edu, [b]mohsen@uconn.edu
[2]Fidelity Investments
245 Summer St., Boston, MA 02210
serdar.kadioglu@fmr.com

Abstract

In predictive modeling, it is often observed that for some of the observations in the dataset, the resulting predictions contradict with what practitioners expect to see. For instance, a binary classifier may predict a person with generally high quality financial metrics to default on a loan, or a soccer player may be predicted to suffer from a certain type of muscle injury even though a kinesiologist might recognize the athelete as low risk. Two possible reasons for model misclassification in contrast to simple expert understanding is lack of enough data to expose patterns, or overfitting to patterns within an available data set. Predictive model typically tries to minimize a loss function with the aim of high accuracy while not taking into account practical aspects of a setting. This paper addresses this shortcoming, and investigates how expert opinion can be incorporated into predictive models in order to boost their performance and practical interpretability, in addition to increasing the chance of use in the real-world so practitioners and experts have more trust in what they typically view as black-box predictions.

In this research, we focus on logistic regression, since it is one of the most important supervised binary classification models in machine learning and aim to answer the following questions:

1. How can we include domain knowledge in parameter estimation for the logistic regression?
2. When domain knowledge is not available, how can we acquire such knowledge using available data?
3. How does including such additional information impact the performance of logistic regression models?

To answer the first question, we introduce a new variant of logistic regression where a set of rules (domain knowledge) are injected into the model as side constraints. By adding such constraints, we want to enforce the model to learn its parameters while avoiding undesireable predictions.

We illustrate how the injected constraints can be linearized. Therefore, given the convex cost function in logistic regression, we can establish the convexity of optimization model used to learn the parameters of the model. The optimal solution satisfies the well-known Karush-Kuhn-Tucker (KKT) conditions, and the model can be solved optimally with gradient-decent-based algorithms.

Acquiring domain knowledge may be hard or it may not be available. To address the second question, we present a framework to gain such knowledge by utilizing decision-tree binary classification. The idea is to select some leaf nodes and extract rules by following their trajectory to the root node of the decision tree, focusing on low-level nodes which thereby has less complex rules (increasing generalizability), consist of relatively large sample sizes (representative), and are pure in the sense that they consist mostly of a single class label (descriptive). To reduce the risk of overfitting, our framework incorporates an iterative train-and-check approach where in each iteration the most violated constraints are added to the model and the most helpful rules are selected from a cross-validation process.

Regarding the third question, we test our proposed framework on a bank marketing dataset (Moro, Cortez, and Rita 2014). The dataset contains over 41,000 observations of individuals consisting of 20 attributes. The dependent variable is whether or not a customer has subscribed to term deposit. Our results indicate that, for a standard logistic regression model, the precision and recall are 81% and 51%, respectively. Our constrained model results achieves 79% accuracy and 61% recall. Our preliminary analysis showcases the potential benefits of including additional information to estimate model parameters for logistic regression.

Constrained predictive models have been previously presented in the literature. For example, Zafar et al. (2015) introduced fairness constraints for fair classification in logistic regression and support vector machines, Anand and Reddy (2011) present constrained logistic regression for discriminative pattern mining, and James, Paulson, and Rusmevichientong (2012) introduce constrained lasso . To our knowledge, we are the first to study including domain knowledge constraints to avoid unwanted outcomes for a classification method. We also present a framework to generate constraints using decision trees when domain knowledge is not available and gradually add them to the logistic regression model to boost performance.

[*]Author names are listed in alphabetical order.

References

Anand, R., and Reddy, C. K. 2011. Constrained logistic regression for discriminative pattern mining. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 92–107. Springer.

James, G. M.; Paulson, C.; and Rusmevichientong, P. 2012. The constrained lasso. In *Refereed Conference Proceedings*, volume 31, 4945–4950. Citeseer.

Moro, S.; Cortez, P.; and Rita, P. 2014. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems* 62:22–31.

Zafar, M. B.; Valera, I.; Rodriguez, M. G.; and Gummadi, K. P. 2015. Fairness constraints: Mechanisms for fair classification. *arXiv preprint arXiv:1507.05259*.

Incremental Approach to Interpretable Classification Rule Learning[*]

Bishwamittra Ghosh and **Kuldeep S. Meel**
School of Computing
National University of Singapore

The recent advances in the machine learning techniques have led autonomous decision making systems be adopted in wide range of domains to perform data-driven decision making. As such the domains range from movie recommendations, ad predictions to legal, medical, and judicial. The diversity of domains mandate different criteria for the machine learning techniques. For domains such as movie recommendations and ad predictions, accuracy is usually the primary objective but for safety critical domains (Otte 2013) such as medical and legal, interpretability, privacy, and fairness (Barocas, Hardt, and Narayanan 2017) are of paramount importance.

It has been long observed that the interpretable techniques are typically trusted and adopted by decision makers as interpretability provides them understanding of reasoning behind a tool's decision making (Ribeiro, Singh, and Guestrin 2016). At this point, it is important to acknowledge that formalizing interpretability is a major challenge (Doshi-Velez and Kim 2017) and we do not claim to have final word on this. In this context, it is worth noting that for several domains such as medical domain, which was the motivation for our investigation, decision rules with small number of rules tend to be most interpretable (Letham et al. 2015).

Since the problem of rule learning is known to be in NP-hard, the earliest efforts focused on heuristic approaches that sought to combine heuristically chosen optimization functions with greedy algorithmic techniques. Recently, there has been surge of effort to achieve balance between accuracy and rule size via principled objective functions and usage of combinatorial optimization techniques such as linear programming (LP) relaxations, sub-modular optimization, or Bayesian methods (Bertsimas, Chang, and Rudin 2012; Marchand and Shawe-Taylor 2002; Malioutov and Varshney 2013; Wang et al. 2015). Motivated by the success of MaxSAT solving over the past decade, Malioutov and Meel proposed a MaxSAT-based approach, called MLIC (Maliotov and Meel 2018), that provides a precise control of accuracy vs. interpretability. The said approach was shown to provide interpretable Boolean formulas without significant loss of accuracy compared to the state of the art classifiers. MLIC, however, has poor scalability in terms of training time and times out for most instances beyond hundreds of samples. In this context, we ask: *Can we design a MaxSAT-based framework to efficiently construct interpretable rules without loss of accuracy and scaling to large real-world instances?*

The primary contribution of this paper is an affirmative answer to the above question. We first investigate the reason for poor scalability of MLIC and attribute it to large size (i.e., number of clauses) of MaxSAT queries constructed by MLIC. In particular, for training data of n samples over m boolean features, MLIC constructs a formula of size $\mathcal{O}(n \cdot m \cdot k)$ to construct a $k-$clause Boolean formula. We empirically observe that the performance of MaxSAT solvers has worse than quadratic degradation in runtime with increase in the size of query. This leads us to propose a novel incremental framework, called IMLI, for learning interpretable rules using MaxSAT. In contrast to MLIC, IMLI makes p queries to MaxSAT solvers with each query of the size $\mathcal{O}(\frac{n}{p} \cdot m \cdot k)$. IMLI relies on first splitting the data into p batches and then incrementally learning rules on the p batches in a linear order such that rule learned for the i-th batch not only uses the current batch but regularizes itself with respect to the rules learned from the first $i-1$ batches.

We now discuss briefly about formal logic theory and MaxSAT. A CNF (Conjunctive Normal Form) formula on a set of Boolean variables is a conjunction of clauses where each clause is a disjunction of literals. Here a literal is either a variable or its complement. Given a CNF formula, the SAT (satisfiability) problem finds an assignment to the variables that satisfies all the clauses in the formula, wherein a clause is satisfied when at least one literal in that clause is satisfied. MaxSAT is an optimization analogue to SAT, where the goal is to find an assignment that satisfies most of the clauses in the formula. In this problem, we consider a weighted variant of a CNF formula where each clause is given a positive weight. Based on the weight, there are two types of clauses in a formula: a hard clause, where the weight is ∞ and a soft clause, where the weight $\mathbb{R}^+$. To learn rules incrementally over batches of the dataset, we consider a partial weighted MaxSAT formula, where the goal is to find an optimal as-

[*]This is an extended abstract of our paper: IMLI: An Incremental Framework for MaxSAT-Based Learning of Interpretable Classification Rule, AIES-19.

signment that satisfies all the hard clauses and most of the soft clauses such that the total weight of the satisfied soft clauses is maximized.

In this paper, we reduce the learning problem as an optimization problem, where we optimize both the interpretability and the prediction accuracy of a rule. We consider a standard binary classification problem on a dataset with binary features. Features with categorical and real-valued features can be converted to binary features by applying standard discretization techniques as in (Maliotov and Meel 2018). Let $\mathbb{1}\{true\} = 1$ and $\mathbb{1}\{false\} = 0$. To learn a k-clause CNF rule from the dataset, we consider two types of boolean decision variables: feature variable $b_i^j = \mathbb{1}\{j\text{-th feature is selected in } i\text{-th clause}\}$ and noise variable $\eta_q = \mathbb{1}\{\text{sample } q \text{ is misclassified}\}$. In our proposed incremental approach, we first split the original dataset into fixed number p of batches. Given a training set ($\mathbf{X} \in \{0,1\}^{n\times m}, \mathbf{y} \in \{0,1\}^n$) in the τ-th batch, we consider the following optimization function.

$$\min \sum_{i,j} b_i^j \cdot I(b_i^j) + \lambda \sum_q \eta_q$$

where indicator function $I(\cdot)$ is defined as follows.

$$I(b_i^j) = \begin{cases} -1 & \text{if } b_i^j = 1 \text{ in the } (\tau-1)\text{-th batch } (\tau \neq 1) \\ 1 & \text{otherwise} \end{cases}$$

The first term in the objective function tries to keep the assignment of all feature variables in the previous batch except in the first batch, where the preference is given on the sparsity (i.e., interpretability) of the rule. The second term in the objective function emphases on minimizing the prediction error. λ is the data fidelity parameter balancing the trade-off between the sparsity and the prediction accuracy of the learned rule. Higher value of λ guarantees less prediction error while sacrificing the sparsity of $\mathcal{R}$ by adding more literals in $\mathcal{R}$, and vice versa.

We now discuss how to construct the MaxSAT formula for learning rule in a batch. We construct soft clauses to encode the objective function and hard clauses to encode the constraints for all samples, that is, a positive labeled sample must satisfy the learned rule and a negative labeled sample must dissatisfy the rule, otherwise the sample is detected as a classification noise. The weight of the soft clause is derived from the coefficients in the objective function. The clauses in the MaxSAT formula are defined as follows:

$$S_j^i := \begin{cases} b_i^j & \text{if } b_i^j = 1 \text{ in the } (\tau-1)\text{-th batch}(\tau \neq 1) \\ \neg b_i^j & \text{otherwise} \end{cases},$$

$$\mathsf{wt}(S_j^i) = 1;$$

$$E_q := \neg\eta_q, \qquad \mathsf{wt}(S_j^i) = \lambda;$$

$$H_q := \neg\eta_q \rightarrow \Big(y_q \leftrightarrow \bigwedge_{i=1}^{k} \mathbf{X}_q \circ \mathbf{b}_i\Big), \qquad \mathsf{wt}(H_q) = \infty.$$

In the hard clause H_q, $\mathbf{X}_q$ is the q-th row of input matrix $\mathbf{X}$, y_q is the q-th element of $\mathbf{y}$, and $\mathbf{b}_i = \{b_i^j \mid j \in \{1,\ldots,m\}\}$. Between two vectors $\mathbf{u}$ and $\mathbf{v}$ over boolean variables or constants (i.e., $0, 1$), we refer $\mathbf{u} \circ \mathbf{v}$ to represent the inner product of $\mathbf{u}$ and $\mathbf{v}$, i.e., $\mathbf{u} \circ \mathbf{v} = \bigvee_i u_i \wedge v_i$, where u_i and v_i denote a variable/constant at the i-th index of $\mathbf{u}$ and $\mathbf{v}$ respectively.

Once we construct all soft and hard clauses, the MaxSAT query Q is the conjunction of all clauses.

$$Q := \bigwedge_{q=1}^{n} E_q \wedge \bigwedge_{i=1,j=1}^{i=k,j=m} S_j^i \wedge \bigwedge_{q=1}^{n} H_q$$

Our learned rule consists of features that are assigned 1 in the optimal solution of Q by an off-the-shelf MaxSAT solver.

We conduct a comprehensive experimental study over a large set of benchmarks and show that IMLI significantly improves upon the runtime performance of MLIC by achieving a speedup of up to three orders of magnitude. Furthermore, the rules learned by IMLI are significantly small and easy to interpret compared to that of the state-of-the-art classifiers such as RIPPER and MLIC. We think IMLI highlights the promise of MaxSAT-based approach and opens up several interesting research directions at the intersection of AI and SAT/SMT community. In particular, it would be an interesting direction of future research if the MaxSAT solvers can be designed to take advantage of incrementality of IMLI.

References

Barocas, S.; Hardt, M.; and Narayanan, A. 2017. Fairness and machine learning. *NIPS Tutorial*.

Bertsimas, D.; Chang, A.; and Rudin, C. 2012. An integer optimization approach to associative classification. In *Proc. of NIPS*.

Doshi-Velez, F., and Kim, B. 2017. Towards a rigorous science of interpretable machine learning.

Letham, B.; Rudin, C.; McCormick, T. H.; Madigan, D.; et al. 2015. Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. *The Annals of Applied Statistics*.

Maliotov, D., and Meel, K. S. 2018. Mlic: A maxsat-based framework for learning interpretable classification rules. In *Proc. of CP*.

Malioutov, D. M., and Varshney, K. R. 2013. Exact rule learning via boolean compressed sensing. In *Proc. of ICML*.

Marchand, M., and Shawe-Taylor, J. 2002. The set covering machine. *Journal of Machine Learning Research* (Dec).

Otte, C. 2013. Safe and interpretable machine learning: a methodological review. In *Computational intelligence in intelligent data analysis*.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. Why should i trust you?: Explaining the predictions of any classifier. In *Proc. of KDD*.

Wang, T.; Rudin, C.; Doshi-Velez, F.; Liu, Y.; Klampfl, E.; and MacNeille, P. 2015. Or's of and's for interpretable classification, with application to context-aware recommender systems.

A Software & Hardware based SAT Solving System with FPGA

Anping He,[1] Lvying Yu,[2] Yuqing Liang,[1] Pengfei Li,[1] Yongcong Wang,[1] Jinzhao Wu[3]
[1]School of Information Science and Engineering, Lanzhou University,Lanzhou, China
[2]Xinnuo Pusi Technology(Wuhan) Co. ,Ltd, Wuhan, China
[3]Guangxi Key Lab of Hybrid Computation and IC Design Analysis, Guangxi University for Nationalities, Nanning, China
{heap,yuly16,liangyq18,lipf17,wangyc16}@lzu.edu.cn, gxmdwjzh@aliyun.com

Introduction

The Boolean satisfiability problem (SAT) is the first proved NP problem and has been proved by Cook as an NPC problem (Cook 1971), so any NP problem can be approximated to the SAT problem in polynomial time. Many practical problems can be solved by solving the SAT problem. Solving the SAT problem is to judge whether there exists a set of variable assignments that make the Boolean expression true. The speed of solving a SAT problem depends on the performance of the SAT solver, so it is especially important to develop a faster solver. At present, the SAT problem has been applied in many fields, such as artificial intelligence, text processing, and integrated circuit design and verification (En and Srensson 2004).

In this article, we introduce a FPGA based asynchronous SAT solver (FaSATer), which is composed of software and FPGA chip. To speed up the SAT problem, the design of the solver improves the classical DPLL algorithm (Davis and Putnam 1960) by adding stochastic strategies and the multiple parallel strategy and using asynchronous methodology. Compared with synchronous circuit, the asynchronous circuit uses the asynchronous controller's handshake signal instead of the clock signal to control the whole circuit in the large-scale integrated circuit design. It can avoid the clock offset and power consumption and excessive delay faced by synchronous circuit design. Today, some products based on asynchronous design methodology have been born. For example, the recent TrueNorth chip from IBM (Akopyan et al. 2015) and Intel's Loihi chip(Davies et al. 2018). In order to fully utilize the digital features of the FPGA, this design uses an instance-based method to independently develop and configure for each SAT instance.

Architecture

In the field of SAT solving, the standard form of Boolean expressions is the conjunction paradigm, so the conjunction paradigm file (CNF) is set as entry of solver and needs to be parsed. In this study, the parser is implemented by JAVA programming language. It is responsible for parsing CNF files into Verilog files which are suitable for the hardware platform of the FPGA. The work of hardware part is to map Verilog files parsed by software onto the FPGA chip, which is correspond to the AND, OR and NOT gates in hardware circuit one by one, then a SAT problem is solved by using the hardware circuit.

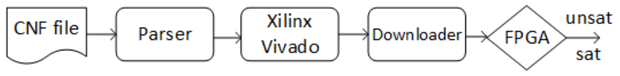

Figure 1: The tool chain of the asynchronous FaSATer.

The tool chain of the asynchronous solver is shown in Fig. 1. The FaSATer works under the mechanism of hardware and software cooperation. The specific workflow is divided into three steps:

- **Parse the CNF file:** Parse the CNF file into hardware circuit module described by Verilog language through a software-implemented tool.
- **Import to vivado:** Import the parsed Verilog file into vivado platform of Xilinx, and then synthesis, implementation and generate bitstream.
- **Download to FPGA:** Downloaded the bitstream to the FPGA development board through the downloader and verify the solution. By observing the LED lamp on the development board to finnd out whether the problem is sat or not.

Main Technology

Template

The software part is responsible for parsing the CNF file to generate the required hardware language description of the circuit file. Therefore, the first step is to generate the template of the configuration file, which records the relevant circuit information, including the port information of all modules, the connection information between ports and the connection information between modules. It consists of three parts: **module definition**, **module instance** and **module connection**. The definition of module describes the number of variables and the type of module. Module instances provide the names of different numbers of instances according to the type of module. In the third part, each sub-element includes the type and name of the connection port, the port

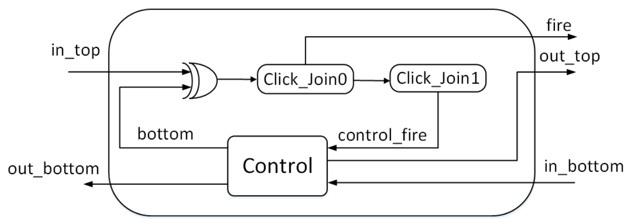

Figure 2: The structure of bi-chain node.

type, the wire connected to the port, and the name, type and port name of another module connected to the port. Using this method, the module connection describes the connection of different modules in the whole circuit.

Different SAT instance only affect the number of variables and clauses in the template. Therefore, based on the characteristics of the template, Verilog generator can be easily implemented.

Asynchronous bi-chain

The asynchronous circuit has no clock, but adopts handshake to realize communication. The asynchronous controller is the key component of the asynchronous circuit, which implements the handshake protocol. In FaSATer, the *click Join* controller (Gilla and Swetha 2018) is used. The controller has only one request input port, one request output port and supports a mechanism to manage data processing.

The bi-direction chain structure, named bi-chain for short, is implemented by the asynchronous circuit. The structure of the module is shown in Fig. 2. The module is composed of *click Join* which carries a request to activate the variable assignment module and control unit. The control unit determines the request signal is transmitted forward or backward.

In addition, the control unit can also activate the asynchronous bi-chain by receiving the backward request signal from the forward bi-chain node (in_bottom), and then the control unit will activate the variable assignment module to re-assign the variables.

In order to demonstrate the advantages of asynchronous bi-chain, the performance of synchronous clock-based and asynchronous bi-chain is compared in Table 1. The synchronous bi-chain uses the highest clock frequency of 200MHz of Xilinx's Artix-7 core board. It can be seen from the table that the data transfer efficiency of the asynchronous bi-chain is much higher. Compared with the synchronous bi-chain, the average efficiency of the asynchronous FIFO is improved by about 44.2%.

Table 1: Comparison of synchronous and asynchronous bi-chain

Node Number	Data Width	Sync	Async	Promotion
2*8	16bit	46.491ns	36.971ns	25.7%
	32bit	46.675ns	34.080ns	37.0%
2*32	16bit	166.467ns	115.762ns	43.8%
	32bit	166.705ns	151.517ns	10.0%
2*64	16bit	326.464ns	236.707ns	37.9%
	32bit	326.582ns	216.031ns	51.2%

Conclusion

In this study, a software and hardware-based SAT solver based on asynchronous methodology is implemented on the FPGA hardware platform. In the design, three methods are adopted to speed up the solving of the SAT problem and improve the performance of the solver:

- The classical DPLL algorithm is used as the basic algorithm, and it is improved by adding random assignment strategy, random ranking strategy and multiple parallel strategy.
- Use an asynchronous controller instead of clock to avoid the problem of clock offset and slow speed caused by clock signal control.
- Make full use of the flexibility of software and the circuit characteristics of hardware.

The correctness of the FaSATer is verified by multiple solutions with multiple instances. Experiments show that the FaSATer designed in this research has the characteristics of fast solution speed and is suitable for solving hard SAT problems. When solving the SAT problem of 133 clauses containing 42 variables, the solution time is only 2.989 ms.

Acknowledgment

This work is Supported by the National Natural Science Foundation of China under Grant No. 61772006, the Science and Technology Program of Guangxi under Grant No. AB17129012, the Science and Technology Major Project of Guangxi under Grant No. AA17204096, the Special Fund for Scientific and Technological Bases and Talents of Guangxi under Grant No. AD16380076, and the Special Fund for Bagui Scholars of Guangxi.

References

Akopyan, F.; Sawada, J.; Cassidy, A.; Alvarez-Icaza, R.; Arthur, J.; Merolla, P.; Imam, N.; Nakamura, Y.; Datta, P.; and Nam, G. J. 2015. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34(10):1537–1557.

Cook, S. A. 1971. The complexity of theoremproving procedures. *Proc.annual Acm Symp.on Theory of Computing* 4(4):153–165.

Davies, M.; Srinivasa, N.; Lin, T. H.; Chinya, G.; and Hong, W. 2018. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38(1):82–99.

Davis, M., and Putnam, H. 1960. A computing procedure for quantification theory. *Journal of the Acm* 7(3):201–215.

En, N., and Srensson, N. 2004. An extensible sat-solver. *Proc.int.conf.on Theory & Applicationsof Satisfiability Testing* 2919:502–518.

Gilla, M., and Swetha. 2018. Silicon compilation and test for dataflow implementations in gasp and click. *Dissertations and Theses*.

Augmenting the Power of MaxSat Resolution [*]

Javier Larrosa, Emma Rollon
Universitat Politecnica de Catalunya
Barcelona, Spain

Abstract

The refutation power of SAT and MaxSAT resolution is challenged by problems like the *soft* and *hard Pigeon Hole Problem* PHP for which short refutations do not exist. In this paper we augment the MaxSAT resolution proof system with an *extension* rule. The new proof system MaxResE is sound and complete, and more powerful than plain MaxSAT resolution, since it can refute the soft and hard PHP in polynomial time. We show that MaxResE refutations actually subtract lower bounds from the objective function encoded by the formulas. The resulting formula is the *residual* after the lower bound extraction. We experimentally show that the residual of the soft PHP (once its necessary cost of 1 has been efficiently subtracted with MaxResE) is a concise, easy to solve, satisfiable problem.

Introduction

The MaxSAT resolution proof system (Larrosa and Heras 2005; Bonet, Levy, and Manyà 2007; Larrosa, Heras, and de Givry 2008) generalizes SAT resolution (Robinson 1965) allowing it to reason with both *hard* and *soft* clauses. A refutation under this proof system corresponds to a subtraction of a lower bound of the optimum from the formula. The remaining formula is called *residual* since it is what remains after the extraction. Although MaxSAT is a more general language than SAT, it is known that it does not improve over problems such as the hard and soft *Pigeon Hole Problems* PHP for which no short refutations exist.

In this paper we augment the MaxSAT resolution proof system with an *extension* rule. Roughly, the new rule allows to reason in the realm of negative weights, which is not possible using just resolution. We show that the new proof system **MaxResE** is sound and complete. The potential of MaxResE is demonstrated by the fact that it can produce short refutations for the hard and soft PHP.

We conjecture that a potential application of MaxResE is to extract from a formula lower bounds that would not be obtained efficiently otherwise, and solve the residual with an off-the-shelf solver. We demonstrate this idea with the soft PHP: The size of the original formula is $O(m^3)$ with m being the number of holes. We show that a lower bound of 1 can be extracted in $O(m^3)$ inference steps producing a residual formula of size $O(m^4)$. We show experimentally that the satisfiability of the residual can be proved in time linear on its size (i.e, $O(m^4)$).

[*]This work was funded partially by the Spanish MINECO, MCIU, AEI and FEDER (EU) through projects TIN2015-69175-C4-3-R and RTI2018-094403-B-C33

The MaxResE Proof System

MaxSAT Resolution

MaxSAT resolution (Larrosa and Heras 2005; Larrosa, Heras, and de Givry 2008) is the generalization of standard resolution (Robinson 1965) to MaxSAT. It is written as,

$$\frac{(x \vee A, v) \quad (\neg x \vee B, w)}{\begin{array}{c}(A \vee B, m) \\ (x \vee A, v-m) \quad (\neg x \vee B, w-m) \\ (x \vee A \vee \neg B, m) \quad (\neg x \vee B \vee \neg A, m)\end{array}}$$

where $m = \min\{v, w\}$. When A (resp. B) is empty, $\neg A$ (resp. $\neg B$) is constant true, so $x \vee \neg A \vee B$ (resp. $x \vee A \vee \neg B$) is tautological. In MaxSAT resolution the clauses at the top are *replaced* by the clauses at the bottom in the formula.

Extension

Here we present a new MaxSAT proof system, called **MaxResE**. The difference with respect to MaxRes is that we remove the condition of weights being strictly positive. Negative weights may appear during proofs with the application of the following new inference rule, called *extension*:

$$\frac{}{(C, -u) \quad (x \vee C, u) \quad (\neg x \vee C, u)}$$

where u is an arbitrary natural number and x is an arbitrary variable not in C. Note that the rule also applies with $C = \square$. The extension rule adds a triplet of fresh clauses that cancel each other into the formula.

Proofs and Refutations

A proof under the MaxResE proof system is a sequence $\mathcal{F}_0 \vdash \mathcal{F}_1 \vdash \ldots \vdash \mathcal{F}_e$ where $\mathcal{F}_0$ is the original formula and each $\mathcal{F}_i$ is obtained by applying *resolution to clauses with positive weight* or *extension* to $\mathcal{F}_{i-1}$.

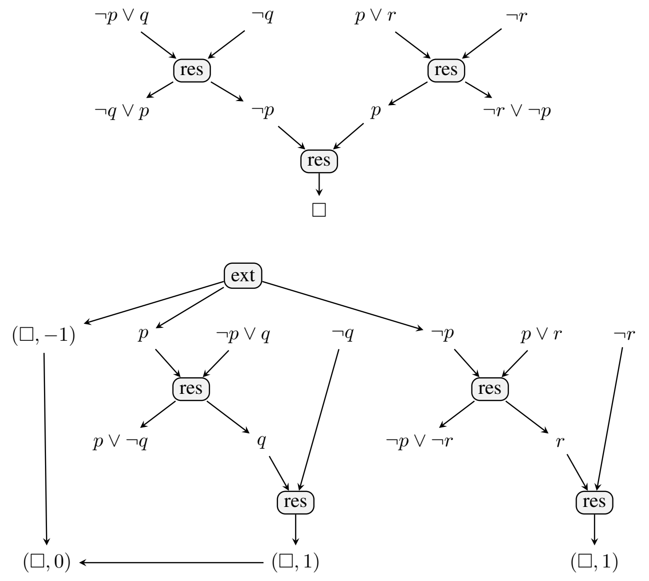

Figure 1: DAG representation of a MaxRes refutation (top) and a MaxResE refutation (bottom). All clauses have cost 1 unless otherwise indicated.

Let $\mathcal{F}_0$ be a formula not containing the empty clause and such that $MaxSAT(\mathcal{F}_0) > 0$. A MaxResE *refutation* is a proof $\mathcal{F}_0 \vdash^* \{(\Box, k)\} \cup \mathcal{F}_e^\delta$ with $k > 0$ and *all weights of* $\mathcal{F}_e^\delta$ *being positive*.

The Soft Pigeon Hole Problem

In the (m holes) *Pigeon Hole Problem* PHP the goal is to assign $m+1$ pigeons to m holes without any pair of pigeons sharing their hole. In the usual encoding there is a boolean variable x_{ij} (with $1 \leq i \leq m+1$, and $1 \leq j \leq m$) associated to pigeon i being in hole j. There are two groups of clauses. For each pigeon i, we have the clause,

$$P_i = \{x_{i1} \vee x_{i2} \vee \ldots \vee x_{im}\}$$

indicating that the pigeon must be assigned to a hole. For each hole j we have the set of clauses,

$$H_j = \{\neg x_{ij} \vee \neg x_{i'j} \mid 1 \leq i < i' \leq m+1\}$$

indicating that the hole is occupied by at most one pigeon. Let PHP be the union of all these sets of clauses,

$$PHP = \cup_{1 \leq i \leq m+1} P_i \ \cup_{1 \leq j \leq m} H_j$$

Note that $|PHP| = O(m^3)$

In the *soft* PHP the goal is to find the assignment that falsifies the minimum number of clauses. In MaxSAT language it is encoded as,

$$PHP_{soft} = \{(C, 1) \mid C \in PHP\}$$

Although the solution to this problem is obvious ($MaxSAT(PHP_{soft}) = 1$), it is known that there is no short MaxRes refutation for it (Bonet, Levy, and Manyà 2007).

Theorem 1. *Consider the encoding of the soft PHP,*

$$PHP_{soft} = \{(C, 1) \mid C \in PHP\}$$

and let m *be the number of holes. There is a MaxResE refutation* $PHP_{soft} \vdash^* \{(\Box, 1)\} \cup PHP_{soft}^\delta$ *of length* $O(m^3)$ *where,*

$$PHP_{soft}^\delta = \cup_{1 \leq i \leq m+1} \mathcal{P}_i^\delta \ \cup_{1 \leq j \leq m} \mathcal{H}_j^\delta$$

$$\mathcal{P}_i^\delta = \{(\neg x_{ij} \vee \neg(x_{ij+1} \vee \ldots \vee x_{im}), 1) \mid 1 \leq j < m\}$$

$$\begin{aligned} \mathcal{H}_j^\delta =& \{(\neg x_{ij} \vee \neg x_{i'j} \vee \neg(x_{i+1j} \vee \ldots \vee x_{i'-1j}), 1) \mid \\ & 1 \leq i < i' - 1 \leq m\} \\ & \cup \{(x_{1j} \vee \ldots \vee x_{m+1j}, 1)\} \end{aligned}$$

Note that $|PHP_{soft}|$ is $O(m^3)$ and $|PHP_{soft}^\delta|$ is $O(m^4)$.

Conclusions

In this paper we have extended the MaxRes proof system from (Larrosa, Heras, and de Givry 2008) and (Bonet, Levy, and Manyà 2007). The new proof system, called MaxResE, is stronger since it can produce short refutations for the hard and soft pigeon hole problem. We have also shown that it generalizes the recently proposed dual rail encoding (Bonet et al. 2018) and it is closely related to optimal soft arc-consistency (Cooper et al. 2010).

References

Bonet, M. L.; Buss, S.; Ignatiev, A.; Marques-Silva, J.; and Morgado, A. 2018. Maxsat resolution with the dual rail encoding. In McIlraith, S. A., and Weinberger, K. Q., eds., *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, 6565–6572. AAAI Press.

Bonet, M. L.; Levy, J.; and Manyà, F. 2007. Resolution for max-sat. *Artif. Intell.* 171(8-9):606–618.

Cooper, M. C.; de Givry, S.; Sánchez-Fibla, M.; Schiex, T.; Zytnicki, M.; and Werner, T. 2010. Soft arc consistency revisited. *Artif. Intell.* 174(7-8):449–478.

Larrosa, J., and Heras, F. 2005. Resolution in max-sat and its relation to local consistency in weighted csps. In Kaelbling, L. P., and Saffiotti, A., eds., *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, 193–198. Professional Book Center.

Larrosa, J.; Heras, F.; and de Givry, S. 2008. A logical approach to efficient max-sat solving. *Artif. Intell.* 172(2-3):204–233.

Robinson, J. A. 1965. A machine-oriented logic based on the resolution principle. *J. ACM* 12(1):23–41.

Parallel Hybrid Best-First Search for Cost Function Networks

Abdelkader Beldjilali, David Allouche, Simon de Givry
INRA, MIA Toulouse, UR-875
31320 Castanet-Tolosan, France

Introduction

Cost Function Networks (CFNs), also known as Weighted Constraint Satisfaction Problems (Meseguer, Rossi, and Schiex 2006) is a mathematical model which has been derived from Constraint Satisfaction Problems by replacing constraints with cost functions. In a CFN, we are given a set of variables with an associated finite domain and a set of local cost functions. Each cost function involves some variables and associates a non-negative integer cost to each of the possible combinations of values they may take. The usual problem considered is to assign all variables in a way that minimizes the sum of all costs. This problem is NP-hard, and exact methods usually rely on Branch and Bound (B&B) algorithms exploring a binary search tree with propagation at each node in order to improve the problem lower bound and prune domain values with a forbidden cost (Cooper et al. 2010). Several B&B search methods have been developed in the CFN solver `toulbar2`[1].

In this work we describe a first parallel version of Hybrid Best-First Search (HBFS) (Allouche et al. 2015) and give a preliminary empirical evaluation on combinatorial optimization problems from uncapacitated warehouse location, computational protein design, and genetics. This last section includes solving time and speed-up comparisons between our approach within `toulbar2` and other parallel approaches available in IBM Ilog `cplex` and `daoopt` (Otten and Dechter 2017) (i.e., respectively a Mixed Integer Programming and an AND/OR search Graphical Model solver).

Parallel HBFS

The sequential version of HBFS (Allouche et al. 2015) is a B&B method for CFNs that combines Best-First Search (BFS) and Depth-First Search (DFS). Like BFS, HBFS provides an anytime global lower bound on the optimum, while also providing anytime upper bounds, like DFS. Hence, it provides feedback on the progress of search and solution quality in the form of an optimality gap. Besides, it exhibits highly dynamic behavior that allows it to perform on par with methods like Limited Discrepancy Search (LDS) and frequent restarting in terms of quickly finding good solutions. As in BFS, HBFS maintains a frontier of open search nodes. It expands each open node using DFS with a limit on its number of backtracks. Each bounded DFS returns a new list of open nodes to be inserted in the BFS frontier.

The parallel version of HBFS is based on the Master-Worker parallel paradigm (Ralphs et al. 2018) where the *Master* is in charge of the open node frontier and dispatches the current best (with minimum lower bound) open node plus the current best solution found so far to the next available *Worker*. The Worker performs a bounded DFS starting from the received node and returns to the Master the resulting list of open nodes (see Fig. 1, with a DFS limit of 3 backtracks). Each open node is associated to a corresponding lower bound and a vector of search decisions. The Worker also returns the best solution found during its limited search if any. Only the Master has a global view of the whole search and reports optimality gaps until the proof of optimality is reached (when the current best frontier lower bound, including active Worker starting nodes, is equal or greater than the cost of the best solution found so far).

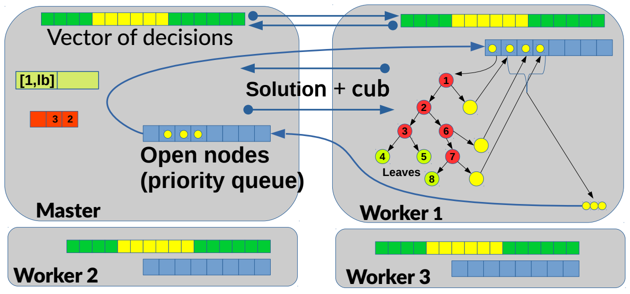

Figure 1: Parallel HBFS using the Master-Worker paradigm.

Instance	n	d	Time (sec.)		Speed-up
			`HBFS`	`HBFS-24`	
capmo1	200	100	10.92	**5.14**	2.12
capmo2	200	100	**1.80**	2.04	0.88
capmo3	200	100	6.09	**3.73**	1.63
capmo4	200	100	4.36	**3.21**	1.36
capmo5	200	100	2.69	**2.58**	1.04

Table 1: Warehouse benchmark (Larrosa et al. 2005) with n, number of variables, and d, maximum domain size.

Experimental Results

We implemented parallel HBFS inside `toulbar2` using the MPI library. Experiments were performed on 24-core

[1]`https://github.com/toulbar2/toulbar2`

servers (Intel Xeon E5-2680/87 at 2.50/3GHz and 256 GB) and the GenoToul cluster (64-core nodes of Intel Xeon E5-2683 at 2.10GHz).

Instance	n	d	Time (sec.)		Speed-up
			`HBFS`	`HBFS-24`	
1xaw	107	412	721.43	**568.50**	1.27
3lf9	120	416	**407.28**	407.92	1
5dbl	130	384	**122.84**	171.82	0.71
5e10	133	400	**147.73**	198.23	0.75
5e0z	136	420	**148.26**	193.41	0.77
5eqz	138	434	3,366.11	**1,049.11**	3.21
1dvo	152	389	622.03	**463.29**	1.34
4bxp	170	439	**327.46**	395.16	0.83
1is1	185	431	-	**2,545.82**	-
2gee	188	397	**797.22**	863.64	0.92
5jdd	263	406	-	**2,758.98**	-
3r8q	271	418	1,605.30	**1,294.97**	1.24
1f00	282	430	-	**2,140.40**	-

Table 2: Computational Protein Design (CPD) benchmark (Ouali et al. 2017). A '–' indicates that the corresponding method failed to prove optimality in less than 1 hour.

Instance	n	d	`cplex`	`cplex-10`	`HBFS`	`HBFS-10`	Speed-up
1UBI	13	198	-	-	1,023	**214.02**	4.78
2DHC	14	198	-	-	8.2	**5.83**	1.41
2DRI	37	186	-	-	135.5	**30.00**	4.52
1CDL	40	186	-	-	392.6	**54.95**	7.14
1CM1	42	186	-	6,177	6.6	**6.11**	1.08
1BRS	44	194	-	-	555.3	**107.86**	5.15
1GVP	52	182	-	-	596.1	**185.75**	3.21
1RIS	56	182	-	-	129.7	**36.23**	3.58
3CHY	74	66	-	5,259	88.7	**20.71**	4.28

Table 3: Another CPD benchmark (Allouche et al. 2014). A '–' indicates that the corresponding method failed to prove optimality in less than $9,000$ seconds. Only instances solved in more than 5 sec. by any successful method are reported.

	pedigree19	pedigree31	pedigree44	pedigree51
n	793	1,183	811	1,152
d	5	5	4	5
`cplex`	790	59.30	6.35	36.23
`//10`	191 (4.14)	9.00 (6.59)	2.48 (2.56)	9.43 (3.84)
`//30`	**75** (10.53)	7.17 (8.27)	2.69 (2.36)	**5.34** (6.78)
`daoopt`	375,110	16,238	95,830	101,788
`//20`	27,281 (13.75)	1,055 (15.39)	6,739 (14.22)	6,406 (15.89)
`//100`	7,492 (50.07)	201 (80.79)	1,799 (53.27)	1,578 (64.50)
`HBFS`	3,126	4.34	39.72	1,608
`//10`	434.27 (7.20)	1.51 (2.87)	6.08 (6.53)	179.22 (8.97)
`//20`	227.02 (13.77)	1.39 (3.12)	3.18 (12.49)	72.30 (22.24)
`//100`	119.43 (26.17)	**0.97** (4.47)	**1.64** (24.22)	31.40 (51.21)

Table 4: Linkage benchmark (Favier et al. 2011) with different number of cores (speed-up in parentheses).

We report in Table 1 and Table 2 solving times to find and prove optimality on Warehouse and CPD benchmarks for the sequential and 24-core parallel versions of HBFS. Parallel HBFS solved three more CPD instances within the 1-hour time-out. The maximum speed-up was 2.12 (resp. 3.21) for Warehouse (resp. CPD), which is rather limited compared to the number of cores used. Experiments on difficult instances of another CPD benchmark using only 10 cores resulted in better speed-ups (up to 7.14 on 1CDL, see Table 3). Moreover our CFN approach was much faster than `cplex`. In Table 4, we compared `cplex`, `daoopt`, and HBFS on the Linkage benchmark. We report `daoopt` time from (Otten and Dechter 2017), obtained on a cluster of dual 2.67 GHz Intel Xeon X5650 6-core CPUs and 24 GB of RAM. Here, the sequential version of HBFS is dominated by `cplex` on three instances among four. But, the parallel version of HBFS got better relative speed-ups than `cplex` when the number of cores increases. We found `daoopt` got very good speed-ups on these instances but still was far from `cplex` in terms of CPU times.

Conclusions

Parallel HBFS is a first parallel approach for HBFS. It provides interesting results on several instances, outperforming in some cases state of the art solvers like `cplex` and `daoopt`. Even if the scalability of our approach must be subject of deeper investigation, due to the minimal size of the information shared between the Master and the Workers, the approach is very likely compliant with a larger number of cores. We found that the speed-up was very instance dependent, and must be also investigated.

As future work, we will take into account the structure of CFNs by parallelizing Backtrack with Tree Decomposition (BTD-HBFS) (Allouche et al. 2015). The resulting parallel method could replace LDS inside a parallel large neighborhood search strategy (Ouali et al. 2017) offering better anytime lower and upper bounds.

Acknowledgments This work has been partially funded by the french Agence nationale de la Recherche, reference ANR-16-C40-0028. We are grateful to the genotoul bioinformatics platform Toulouse Midi-Pyrenees (Bioinfo Genotoul) for providing computing and storage resources.

References

Allouche, D.; André, I.; Barbe, S.; Davies, J.; de Givry, S.; Katsirelos, G.; O'Sullivan, B.; Prestwich, S.; Schiex, T.; and Traoré, S. 2014. Computational protein design as an optimization problem. *Artificial Intelligence* 212:59–79.

Allouche, D.; de Givry, S.; Katsirelos, G.; Schiex, T.; and Zytnicki, M. 2015. Anytime Hybrid Best-First Search with Tree Decomposition for Weighted CSP. In *Proc. of CP*, 12–28.

Cooper, M.; de Givry, S.; Sanchez, M.; Schiex, T.; Zytnicki, M.; and Werner, T. 2010. Soft arc consistency revisited. *AI* 174:449–478.

Favier, A.; Givry, S.; Legarra, A.; and Schiex, T. 2011. Pairwise decomposition for combinatorial optim. in graphical models. In *Proc. of IJCAI*, 2126–2132.

Larrosa, J.; de Givry, S.; Heras, F.; and Zytnicki, M. 2005. Existential arc consistency: getting closer to full arc consistency in weighted CSPs. In *Proc. of IJCAI*, 84–89.

Meseguer, P.; Rossi, F.; and Schiex, T. 2006. Soft constraints processing. In *Handbook of Constraint Programming*. Elsevier. chapter 9, 279–326.

Otten, L., and Dechter, R. 2017. And/or branch-and-bound on a computational grid. *JAIR* 59:351–435.

Ouali, A.; Allouche, D.; de Givry, S.; Loudni, S.; Lebbah, Y.; Eckhardt, F.; and Loukil, L. 2017. Iterative Decomposition Guided Variable Neighborhood Search for Graphical Model Energy Minimization. In *Proc. of UAI-17*, 550–559.

Ralphs, T.; Shinano, Y.; Berthold, T.; and Koch, T. 2018. *Handbook of Parallel Constraint Reasoning*. Springer. chapter Parallel Solvers for Mixed Integer Linear Optimization, 283–336.

Constraint Programming and Hybrid Decomposition Approaches to Discretizable Distance Geometry Problems

Moira MacNeil,[1] **Merve Bodur**[2]
Department of Mechanical and Industrial Engineering, University of Toronto
[1] m.macneil@mail.utoronto.ca, [2] bodur@mie.utoronto.ca

Given an integer dimension K and a simple, undirected graph G with positive edge weights, the Distance Geometry Problem (DGP) aims to find a realization function mapping each vertex of G to a coordinate in a K-dimensional space such that the distance between pairs of vertex coordinates is equal to the corresponding edge weights in G.

The DGP has wide application areas, including astronomy, where we position stars relative to each other, robotics, where the distances are arm lengths and we are trying to determine a set of positions within reach of a robot (Lavor et al. 2017; Liberti et al. 2014; Mucherino et al. 2012). In molecular geometry, Nuclear Magnetic Resonance spectroscopy is used to image molecules, usually proteins, in two dimensions, but finding their three dimensional structure is key for determining their functional properties. In this case, we are positioning the atoms in three-dimensional Euclidean space (Lavor et al. 2017). In wireless sensor localization, the network has components with fixed positions, such as routers, and we wish to determine the positions of mobile wireless sensors, such as smartphones (Liberti et al. 2014). Other applications include statics and graph rigidity, graph drawing, and clock synchronization (Lavor et al. 2017; Liberti et al. 2014; Mucherino et al. 2012).

The so-called discretization assumptions reduce the search space of the realization from a continuous space to a finite discrete one. The solution space of these Discretizable Distance Geometry Problems (DDGPs) is often represented as a binary tree structure that may be searched using a branch and prune (BP) algorithm. Given a *discretization vertex order* in G, the BP algorithm constructs a binary tree where the nodes at a layer provide all possible coordinates of the vertex corresponding to that layer.

We focus on the *Discretization Vertex Order Problem* (DVOP) (Lavor et al. 2012) which determines for a simple, connected, undirected graph G and a dimension K if there exists a total vertex ordering that satisfies:

1. the first K vertices in the order form a clique in the input graph, G, and
2. the following vertices each have at least K adjacent vertices in G as predecessors in the order.

We refer to this as a DVOP order. Given a DVOP order, the BP algorithm solves the DDGP by fixing the coordinates of the first K vertices in the order and enumerating the possible realizations of the remaining vertices. In the BP search tree, branching on a vertex with exactly K predecessors in the order yields at most two child nodes which are called *double vertices*. Otherwise if the vertex has more than K predecessors it has at most one child. The number of double vertices can be used as a measure of the size of the BP tree.

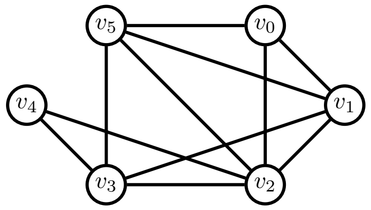

Figure 1: A graph instance which is feasible for DVOP with $K = 2$.

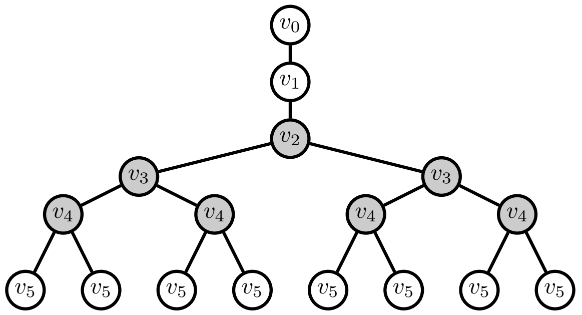

Figure 2: A BP tree for DVOP order $(v_0, v_1, v_2, v_3, v_4, v_5)$ of the graph in Figure 1.

Figure 2 and Figure 3 show two BP trees for feasible DVOP orders for the graph given in Figure 1 with $K = 2$. The tree in Figure 2 is for a DVOP order which has three double vertices, v_2, v_3, v_4, thus the tree has at most 17 nodes. However, Figure 3 depicts the BP tree for a DVOP order which has two double vertices, v_2, v_4, so the tree has at most 9 nodes. Thus, an order with fewer doubles yields a smaller BP tree. The focus of this work is finding *optimal BP trees*

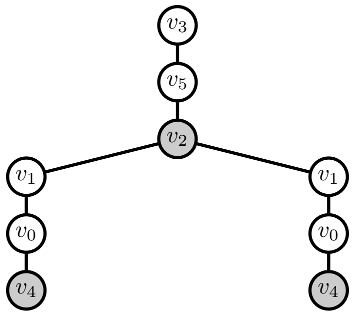

Figure 3: A BP tree for DVOP order $(v_3, v_5, v_2, v_1, v_0, v_4)$ of the graph in Figure 1.

for DVOPs. More specifically, we aim to find a discretization vertex order in G that yields a BP tree with the least number of branches, and thus the minimum number of doubles.

Our contributions are as follows:

- We propose three novel constraint programming formulations that all significantly outperform the state-of-the-art cutting plane algorithm for this problem.
- Motivated by the difficulty in solving instances with a large input graph, we develop two hybrid constraint programming-integer programming decomposition algorithms
- We strengthen the hybrid decomposition algorithms with a set of valid inequalities, which further improve the solvability of the problem.
- We conduct computational experiments on two types of instances: randomly generated graphs and synthetic minimal instances and present the results.

References

Lavor, C.; Lee, J.; Lee-St. John, A.; Liberti, L.; Mucherino, A.; and Sviridenko, M. 2012. Discretization orders for distance geometry problems. *Optimization Letters* 6(4):783–796.

Lavor, C.; Liberti, L.; Lodwick, W. A.; and da Costa, T. M. 2017. *An Introduction to Distance Geometry applied to Molecular Geometry*. Springer.

Liberti, L.; Lavor, C.; Maculan, N.; and Mucherino, A. 2014. Euclidean distance geometry and applications. *SIAM Review* 56(1):3–69.

Mucherino, A.; Lavor, C.; Liberti, L.; and Maculan, N. 2012. *Distance geometry: theory, methods, and applications*. Springer Science & Business Media.

Arc Consistency Revisited[*]

Ruiwei Wang and **Roland H. C. Yap**
School of Computing, National University of Singapore, Singapore
{ruiwei,ryap}@comp.nus.edu.sg

Extended Abstract

Binary constraint is a general representation for constraints and is used in Constraint Satisfaction Problems (CSPs) to model/solve any discrete combinatorial problem. Historically, work on constraint satisfaction began with binary CSPs, problems with at most two variables per constraint and many algorithms have been proposed to maintain Arc Consistency (AC) on binary constraints. The seminal work of Mackworth (Mackworth 1977) proposed a basic local consistency, arc consistency, which has been the main reasoning technique used in constraint solvers for CSPs. However, many problems are more naturally modelled with non-binary constraints (constraints with arity > 2). Several well-known binary encoding methods can be used to transform non-binary CSPs to binary CSPs. Non-binary CSPs can also be solved directly which would require non-binary constraint solvers, Generalized Arc Consistency (GAC) is the natural extension of AC for non-binary constraints. In more recent times, research has focused on non-binary constraints and efficient GAC algorithms using clever algorithms, representations and data structures (Ullmann 2007; Lecoutre 2011; Lecoutre, Likitvivatanavong, and Yap 2012; Perez and Régin 2014; Cheng and Yap 2010; Verhaeghe, Lecoutre, and Schaus 2018; Xia and Yap 2013; Katsirelos and Walsh 2007; Jefferson and Nightingale 2013; Gharbi et al. 2014; Wang et al. 2016; Demeulenaere et al. 2016; Verhaeghe, Lecoutre, and Schaus 2017; Verhaeghe et al. 2017). Improvements in GAC algorithms have led to a "folklore belief" that AC algorithms on the binary encoding of a non-binary CSP do not compete with GAC. It has also spurred major developments in GAC algorithms.

We first show with experimental comparisons of binary encoding with state-of-art GAC algorithms reasons why binary encoding with existing AC algorithms are outperformed by GAC on non-binary constraints. Our experiments investigate two well-known binary encodings, CSP instances from the dual and hidden variable encodings. On standard benchmarks (instances from the XCSP3 website), transforming non-binary CSPs to binary CSPs with the dual encoding leads to very large CSPs which can run out of memory. The hidden variable encoding does not suffer from this space issue. However, comparing specialized AC algorithms designed for the hidden variable encoding, namely, HAC (Mamoulis and Stergiou 2001) with state-of-art GAC algorithms such as CT (Demeulenaere et al. 2016), shows that on average CT significantly outperforms HAC. This explains the folklore belief that non-binary CSPs are best solved in their non-binary form with a GAC algorithm. Our experiments show that there are two reasons: (i) CT is faster than HAC when the search nodes are similar; and (ii) the hidden variable encoding can affect the search heuristic, and the search space can be smaller or larger (but many instances have more search nodes).

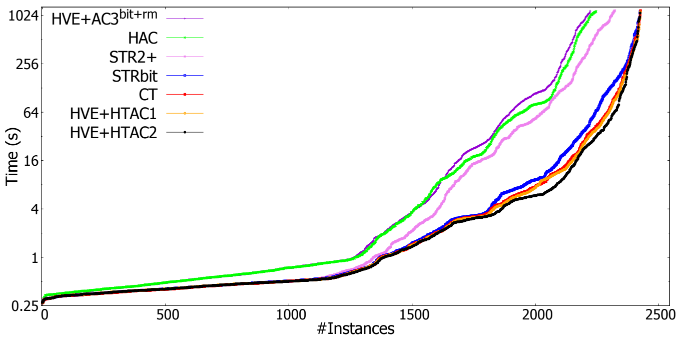

Figure 1: Runtime Distribution: HTAC, HAC, HVE+AC3^{bit+rm}, CT, STRbit, STR2+

We propose new algorithms to enforce AC on binary encoded instances which address these two drawbacks: (i) a more efficient propagator for hidden variable binary constraints; and (ii) control the interaction between the search heuristic and the binary encoded model. Preliminary experiments show that our AC algorithm (HTAC) can be much faster than state-of-the-art AC algorithms applied to binary encodings of non-binary CSPs. It is also competitive (or better) than GAC algorithms, CT and STRbit (Wang et al. 2016), run on the original instance.

A summary of our results on 2431 non-binary CSP instances across many series from the XCSP3 problems is given in Figure 1. The graph show the runtime distribution of the problem instances solved by the algorithms de-

[*]The full paper appears in (Wang and Yap 2019).

scribed below. It compares non-binary CSP instances transformed through the hidden variable encoding (HVE) into binary CSPs solved with the following AC algorithms: HTAC (our new algorithm(s)), AC3^{bit+rm} (Lecoutre, Hemery, and others 2007), HAC. While for the original non-binary instance, we solve them using recent GAC algorithms: CT, STRbit and STR2+ (Lecoutre 2011). The graph shows that the HTAC algorithms are competitive with state-of-art GAC algorithms and perhaps slightly outperform the CT algorithm.

This result is surprising and is contrary to the "folklore" on AC vs GAC algorithms. Contrary to popular belief, AC algorithms on the binary encoding (hidden variable encoding) of a non-binary CSP instance can be just as good as using GAC algorithms on the non-binary CSP instance. This suggests that it may be interesting to take revisit AC algorithms. We believe our results can lead to a revival of AC algorithms since binary constraints and resulting algorithms are simpler than the non-binary ones. For example, many stronger consistencies were proposed to handle binary constraints and these have been more extensively studied in the case of binary constraints. Many fundamental works studying properties of CSPs are often also studied primarily in the binary case.

Acknowledgements

We acknowledge the support of grant MOE2015-T2-1-117.

References

[Cheng and Yap 2010] Cheng, K., and Yap, R. H. C. 2010. An MDD-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. *Constraints* 15(2):265–304.

[Demeulenaere et al. 2016] Demeulenaere, J.; Hartert, R.; Lecoutre, C.; Perez, G.; Perron, L.; Régin, J.-C.; and Schaus, P. 2016. Compact-table: Efficiently filtering table constraints with reversible sparse bit-sets. In *International Conference on Principles and Practice of Constraint Programming*, 207–223. Springer.

[Gharbi et al. 2014] Gharbi, N.; Hemery, F.; Lecoutre, C.; and Roussel, O. 2014. Sliced table constraints: Combining compression and tabular reduction. In *Proceedings of the 11th International Conference on Integration of Artificial Intelligence and Operations Research techniques in Constraint Programming*, 120–135.

[Jefferson and Nightingale 2013] Jefferson, C., and Nightingale, P. 2013. Extending simple tabular reduction with short supports. In *Proceedings of the 23rd International Joint Conferences on Artificial Intelligence*, 573–579.

[Katsirelos and Walsh 2007] Katsirelos, G., and Walsh, T. 2007. A compression algorithm for large arity extensional constraints. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, 379–393.

[Lecoutre, Hemery, and others 2007] Lecoutre, C.; Hemery, F.; et al. 2007. A study of residual supports in arc consistency. In *IJCAI*, volume 7, 125–130.

[Lecoutre, Likitvivatanavong, and Yap 2012] Lecoutre, C.; Likitvivatanavong, C.; and Yap, R. H. C. 2012. A path-optimal GAC algorithm for table constraints. In *Proceedings of the 20th European Conference on Artificial Intelligence*, 510–515.

[Lecoutre 2011] Lecoutre, C. 2011. STR2: optimized simple tabular reduction for table constraints. *Constraints* 16(4):341–371.

[Mackworth 1977] Mackworth, A. K. 1977. Consistency in networks of relations. *Artificial intelligence* 8(1):99–118.

[Mamoulis and Stergiou 2001] Mamoulis, N., and Stergiou, K. 2001. Solving non-binary csps using the hidden variable encoding. In *International Conference on Principles and Practice of Constraint Programming*, 168–182. Springer.

[Perez and Régin 2014] Perez, G., and Régin, J.-C. 2014. Improving GAC-4 for table and MDD constraints. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming*, 606–621.

[Ullmann 2007] Ullmann, J. R. 2007. Partition search for non-binary constraint satisfaction. *Information Sciences* 177:3639–3678.

[Verhaeghe et al. 2017] Verhaeghe, H.; Lecoutre, C.; Deville, Y.; and Schaus, P. 2017. Extending compact-table to basic smart tables. In *International Conference on Principles and Practice of Constraint Programming*, 297–307. Springer.

[Verhaeghe, Lecoutre, and Schaus 2017] Verhaeghe, H.; Lecoutre, C.; and Schaus, P. 2017. Extending compact-table to negative and short tables. In *AAAI*, 3951–3957.

[Verhaeghe, Lecoutre, and Schaus 2018] Verhaeghe, H.; Lecoutre, C.; and Schaus, P. 2018. Compact-mdd: Efficiently filtering (s) mdd constraints with reversible sparse bit-sets. In *IJCAI*, 1383–1389.

[Wang and Yap 2019] Wang, R., and Yap, R. H. C. 2019. Extending simple tabular reduction with short supports. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, 599–615.

[Wang et al. 2016] Wang, R.; Xia, W.; Yap, R. H.; and Li, Z. 2016. Optimizing simple tabular reduction with a bitwise representation. In *IJCAI*, 787–795.

[Xia and Yap 2013] Xia, W., and Yap, R. H. C. 2013. Optimizing STR algorithms with tuple compression. In *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming*, 724–732.