Proceedings Paper:
Akgun, Ozgur, Gent, Ian Philip, Jefferson, Christopher Anthony et al. (3 more authors) (2018) Automatic discovery and exploitation of promising subproblems for tabulation. In: Hooker, John, (ed.) Principles and Practice of Constraint Programming:24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings. Lecture Notes in Computer Science . Springer , Netherlands , pp. 3-12.

https://doi.org/10.1007/978-3-319-98334-9_1

Automatic Discovery and Exploitation of Promising Subproblems for Tabulation

Özgür Akgün, Ian P. Gent, Christopher Jefferson,
Ian Miguel, Peter Nightingale, András Z. Salamon

School of Computer Science, University of St Andrews, St Andrews, UK
{ozgur.akgun, ian.gent, caj21, ijm, pwn1, Andras.Salamon}@st-andrews.ac.uk

Abstract. The performance of a constraint model can often be improved by converting a subproblem into a single table constraint. In this paper we study heuristics for identifying promising subproblems. We propose a small set of heuristics to identify common cases such as expressions that will propagate weakly. The process of discovering promising subproblems and tabulating them is entirely automated in the tool Savile Row. A cache is implemented to avoid tabulating equivalent subproblems many times. We give a simple algorithm to generate table constraints directly from a constraint expression in Savile Row. We demonstrate good performance on the benchmark problems used in earlier work on tabulation, and also for several new problem classes.

1 Introduction

In order to improve the performance of a constraint model, a common step is to reformulate the expression of a subset of the problem constraints, either to strengthen the inferences made during search by the constraint solver by increasing constraint propagation, or to maintain the level of propagation while reducing the cost of propagating the constraints. One such method is *tabulation*: to aggregate a set of constraint expressions into a single table constraint [16,11,14], which explicitly lists the allowed tuples of values for the decision variables involved. This allows us to exploit efficient table constraint propagators that enforce generalised arc consistency [3], typically a stronger level of inference than is achieved for a logically equivalent collection of separate constraints. Successful examples of this approach where the reformulation has been performed by hand include Black Hole patience [9] and Steel Mill Slab Design [8].

Recently, Dekker et al. [6] presented a method for the partial automation of tabulation. In their approach a predicate (a Boolean function) expressed in the MiniZinc language [17] may be annotated to be converted automatically into a table constraint. In the same vein, the IBM ILOG CPLEX Optimization Studio software supports `strong` annotations to indicate that the solver should find a precomputed table constraint corresponding to a specified set of variables; the resulting table constraint is then added to the model as an implied constraint [12]. The Propia library performed a similar step for an annotated goal

in ECLiPSe [13]. In all of these approaches, the crucial first step of identifying promising parts of a given model for tabulation is left to the human modeller.

In this work we present an entirely automatic tabulation method situated in the automated constraint modelling tool Savile Row [18,21,19]. A set of heuristics is employed to identify in an Essence Prime [20] model candidate sets of constraints for tabulation, which are then tabulated automatically. In order to demonstrate the effectiveness of our approach, we first examine the same four case studies used by Dekker et al. to demonstrate the utility of tabulation from manual model annotations. We show that our automated approach can identify the same opportunities to improve the model by tabulation. We also study four additional problem classes that show that our tabulation heuristics remain effective on a wider range of problems.

Preliminaries A *constraint satisfaction problem* (CSP) is defined as a set of variables X, a function that maps each variable to its domain, $D : X \rightarrow 2^{\mathbb{Z}}$ where each domain is a finite set, and a set of constraints C. A *constraint* $c \in C$ is a relation over a subset of the variables X. The *scope* of a constraint c, named scope(c), is the sequence of variables that c constrains. The scope has an order and may contain a decision variable more than once. During a systematic search for a solution to a CSP, values are progressively removed from the domains D. A *literal* is a variable-value pair (written $x \mapsto v$). A literal $x \mapsto v$ is *valid* iff $v \in D(x)$. For a constraint c we use r for the size of scope(c). A constraint c is Generalised Arc Consistent (GAC) if and only if there exists a support containing every valid literal of every variable in scope(c). GAC is established by identifying all literals $x \mapsto v$ for which no support exists and removing v from the domain of x. A *support* of constraint c is a set of literals containing exactly one literal for each variable in scope(c), such that c is satisfied by the assignment represented by these literals. In a table constraint the set of supports are explicitly listed.

2 Identifying Promising Subproblems for Tabulation

We have designed four heuristics to identify cases where expert modellers might experiment with tabulation to improve the performance of a CP solver. The heuristics and tabulation operate on the abstract syntax tree (AST) of a model, once all problem class parameters have been substituted in, all quantifiers and comprehensions have been unrolled, and matrices of variables have been replaced by individual variables. Tabulation is applied before common subexpression elimination and general flattening. Details of the tailoring process of Savile Row are given elsewhere [19]. Our heuristics are:

Duplicate Variables identifies a constraint containing at most 10 distinct variables, with at least one variable occurring more than once in the scope.

Large AST identifies a constraint where the number of nodes in the AST is greater than 5 times the number of distinct decision variables in scope.

Weak Propagation identifies a constraint c_1 that is likely to propagate weakly (i.e. less than GAC), such that there is another constraint c_2 that propagates strongly, with at least one variable in the scope of both c_1 and c_2.

Identical Scopes identifies sets of two or more constraints whose scopes contain the same set of decision variables.

Each of the four heuristics is based on a simple rationale regarding either propagation strength or propagation speed of the constraint(s). The Duplicate Variables heuristic identifies constraints that are likely to propagate weakly even when the target solver has a strong propagator for the constraint type. In most cases a GAC propagator will enforce GAC only when there are no duplicate variables. For example, enforcing GAC on the Global Cardinality Constraint (GCC) is known to be NP-hard with duplicate variables [4], therefore Régin's polynomial-time GAC propagator [22] achieves GAC only when there are no duplicate variables. The replacement table constraint will not have duplicate variables in scope and will therefore achieve GAC.

The Large AST heuristic identifies constraints that are not compactly represented in the AST. A typical example would be an element constraint $M[x] = y$ with a large constant matrix M. The rationale behind it is that a table propagator may be more efficient while achieving the same or stronger propagation.

The Weak Propagation heuristic is intended to catch cases where the weak propagation of one constraint is hindering strong propagation of another. For example, suppose we have the constraints allDifferent(x_1, x_2, x_3) and $x_1 = 10x_4 + x_5$, a GAC propagator is used for allDifferent, and a bound consistency propagator is used for sum equality. Tabulating the sum equality constraint and therefore potentially pruning more values from x_1 may strengthen propagation of the allDifferent onto x_2 and x_3. To implement the Weak Propagation heuristic we need to define which constraint expressions are expected to propagate strongly. The definition is recursive on the AST representing the expression. Each type of AST node is defined to be either weak, or strong iff all its children are strong. At the leaves of the AST, constants and references to variables are defined to be strong. For example, the allDifferent constraint often has a GAC propagator so it is defined to be strong iff all its children are strong. The constraint allDifferent(x_1, x_2, x_3) is strong. Sums are defined to be weak because they are often implemented with bound consistency propagators. The constraint allDifferent$(x_1 - x_2, x_3 - x_4, x_5 - x_6)$ is therefore defined to be weak. Its eventual representation in the CP solver is unlikely to enforce GAC.

Finally we consider the Identical Scopes heuristic. It is well known that multiple constraints on the same scope may not propagate strongly together, even if each constraint individually does propagate strongly. The Identical Scopes heuristic is intended to collect such sets of constraints into a single table constraint that may propagate more strongly and also may be faster.

Each heuristic fires on at least one of the case studies in Sections 3 and 4. We discuss the expressions that trigger the heuristics, and the benefits of tabulation.

Caching We use caches to avoid generating identical tables many times for similar constraints. To store or retrieve a table for an expression e, we first place e into a normal form: the expression is simplified and placed into negation normal form [19, Sec. 3.3]. Then all associative and commutative k-ary expressions (such as sums) and commutative binary operators (e.g. $=$) within e are sorted. Alpha-

betical order is used because it will group together references to the same matrix (all else being equal) and place references to different matrices in a consistent order regardless of the indices. The expression is traversed in left-first order to collect a sequence of decision variables (without duplication), and the variables in the sequence are then renamed to a canonical sequence of names to create e'. Thus the actual variable names in e do not affect e', only their relative positions. e' and the variable domains together are used as a key to store and retrieve tables in the caches. We have a persistent cache stored on disk containing tables, and two memory caches: the first contains tables, and the second stores cases where tabulation failed because the tabulator reached one of its limits. When an expression is identified by a heuristic to tabulate, we look it up in the memory caches then the disk cache. In our experiments we disabled the disk cache because it would cause timings to change depending on the order of processes.

Generating Tables Given a boolean expression e to tabulate, we first sort e and collect a list of its variables (without duplicates) in the order used by the cache. This ensures that the columns of the table are in the right order for it to be stored in the cache. A table is generated by depth-first search with a static variable ordering and d-way branching. At each node the expression is simplified [19]; if it evaluates to *false* then the search backtracks. At each leaf that evaluates to *true*, we store the assignment as a tuple in the table. In some cases a heuristic will identify a constraint that is simply too large to be tabulated. To deal with these cases we limit the depth-first search to generate at most 10,000 tuples, and to fail and backtrack at most 100,000 times.

3 Experimental Evaluation: Baseline

Tabulation (whether performed manually or with tool support) is a well-established technique. Therefore, instead of examining whether tabulation is effective, we consider whether we can automatically identify subproblems that can be usefully tabulated. Our first four case studies are the four problems presented by Dekker et al. [6]. In each case we show that our heuristics can automatically identify the same subproblems that Dekker et al. identified by hand, to then yield comparable performance improvements.

Black Hole Black Hole is a patience card game where cards are played one by one into the 'black hole' from seventeen face-up fans of three cards. All cards can be seen at all times. A card may be played into the 'black hole' if it is adjacent in rank to the previous card. Black Hole was modelled for a variety of solvers by Gent et al. [9] and a table constraint was used in the CP model. We use the simplest and most declarative model of Dekker et al. [6] where two variables `a` and `b` represent adjacent cards iff `|a-b| % 13 in {1,12}`. The adjacency constraint triggers the Weak Propagation heuristic because it overlaps with an allDifferent constraint. No other constraint triggers any heuristic, so our set of constraints to tabulate exactly match those identified by hand, first by Gent et al. and later by Dekker et al.

Block Party Metacube Problem The Block Party Metacube Problem is a puzzle in which eight small cubes are arranged into a larger *metacube*, such that the visible faces on each of the six sides of the metacube form a "party". Each small cube has a symbol at each corner of each of its faces (24 symbols per cube in total), and each symbol has three attributes, with each attribute in turn taking one of four values. To form a valid party (the *party constraint*), the four small cubes forming a visible face of the large cube must be arranged so that the four symbols in the middle of the visible face are either all different, or all the same, for each of the three attributes. We use the model and instances of Dekker et al. [6].

Dekker et al. tabulated a channelling constraint linking cubes and icons. The Duplicate Variables heuristic identifies the same channelling constraint and it is successfully tabulated. The party constraint as a whole triggers the Identical Scopes heuristic, and the Duplicate Variables heuristic is triggered by each of the four conjuncts of the party constraint, however these constraints are not tabulated because the tabulator reaches a limit. Overall our system tabulates exactly the same set of constraints as Dekker et al.

Handball Tournament Scheduling Handball Tournament Scheduling requires scheduling matches of a tournament, while respecting the rules governing the tournament, and minimising a cost function related to the availability of venues. We use the simplified 7+7 team model and 20 instances (all of the same size) used by Dekker et al. [6] with a standard decomposition of the *regular* constraint because Savile Row and Minion do not currently implement *regular*.

Dekker et al. experimented with tabulating two types of subproblem, the second of which provided a significant performance improvement. The second type of subproblem is a part of the objective function that calculates the cost of one row of the schedule. The Large AST heuristic triggers for this type of constraint, however one of the limits described in Section 2 prevents these constraints being tabulated. It seems that fixed limits may be too coarse, and a more sophisticated cost-benefit calculation may be required. The Large AST heuristic also triggers for a small number of element constraints containing constant matrices. Tabulation creates a unary table which is absorbed into the variable's domain.

JP Encoding Problem The JP Encoding problem was introduced in the MiniZinc Challenge 2014. In brief, the problem is to find the most likely encoding of each byte of a stream of Japanese text where multiple encodings may be mixed. The encodings considered are ASCII, EUC-JP, SJIS, UTF-8 or unknown (with a large penalty). Once again our model closely follows that of Dekker et al. [6]. We use all 10 instances in the MiniZinc benchmark repository. The instances are from 100 to 1900 bytes in length. Each byte has four variables: the encoding, a 'byte status' variable that combines the encoding with the byte's position within a multibyte character, a 'char start' variable indicating whether the byte begins a new multibyte character, and the score which contributes to the objective.

Dekker et al. tabulate three subproblems. The first connects two adjacent status variables, and the Identical Scopes heuristic triggers on this. The second

links status, encoding, and char start, and we found that the Identical Scopes heuristic separately links status to encoding, and status to char start. The encoding and char start variables are both functionally defined by status so no propagation is lost with two binary table constraints compared to one ternary table. Thirdly Dekker et al. tabulate the constraint linking the score to the encoding. The Duplicate Variables heuristic triggers on this. In summary, the heuristics identify almost the same set of constraints to tabulate as Dekker et al. did manually, and all identified constraints are successfully tabulated.

4 Experimental Evaluation: New Case Studies

In this section we present four case studies that were not featured in Dekker et al. [6]. In each case we briefly describe the model and discuss the expressions that trigger our heuristics. We evaluate tabulation with three CP solvers:
Minion-Static Minion 1.8 [10], ascending value and static variable orderings.
Minion-Conflict Same as the above with Conflict variable ordering [15].
Chuffed Current version of the learning CP solver Chuffed [5] with free search.

Each reported time is the median of five runs on a 64-core AMD Opteron 6376 (32 processes in parallel, 6 hour time limit). Times include the time taken by Savile Row to tailor the instance and (if activated) to tabulate. Software, models and parameter files for the experiment are available online [2], with some additional analysis of experimental results. The results are plotted in Figure 1.
Sports Scheduling Completion The Sports Scheduling problem is to construct a schedule of $n(n-1)/2$ games among n teams where each team plays every other team once with some other constraints. In Sports Scheduling Completion we start with a partial schedule. 10 instances were generated with $n = 12$ and 10 slots assigned uniformly at random. Trivially unsatisfiable instances were excluded. Each game between a pair of teams is represented as a pair of variables `a` and `b` and also a single variable `c`, with the channelling constraint `n*(a-1)+b=c`. The Weak Propagation heuristic identifies the channelling constraint, and tabulating it proves to be highly beneficial for the two Minion configurations. With Chuffed the picture is mixed. Some instances are slowed by tabulation, particularly the easiest four, while some of the more difficult instances benefit from it. Van Hentenryck et al. manually tabulated the same constraint in their OPL model of Sports Scheduling [23].
Langford's Problem Langford's problem (CSPLib problem 24 [1]) with parameters n and k is to find a sequence of length nk which contains k copies of each number in the set $\{1, \ldots, n\}$. The sequence must satisfy the constraint that if the first occurrence of x is at position p, then the other occurrences appear at $p + (x + 1)i$, for $i \in \{1, \ldots, k-1\}$. We model Langfords as an $n \times k$ 2D matrix P, where row i represents the positions of the k occurrences of i. The constraints are $P[i, j] = P[i, j-1] + i + 1$ and all the positions $P[i, j]$ are different. We also break the symmetry that the entire sequence can be reversed by requiring $(P[1, 1] - 1) \leq (nk - P[1, k])$. We use all 80 instances where $n \in \{2 \ldots 17\}$ and $k \in \{2 \ldots 6\}$. The Weak Propagation heuristic triggers on the

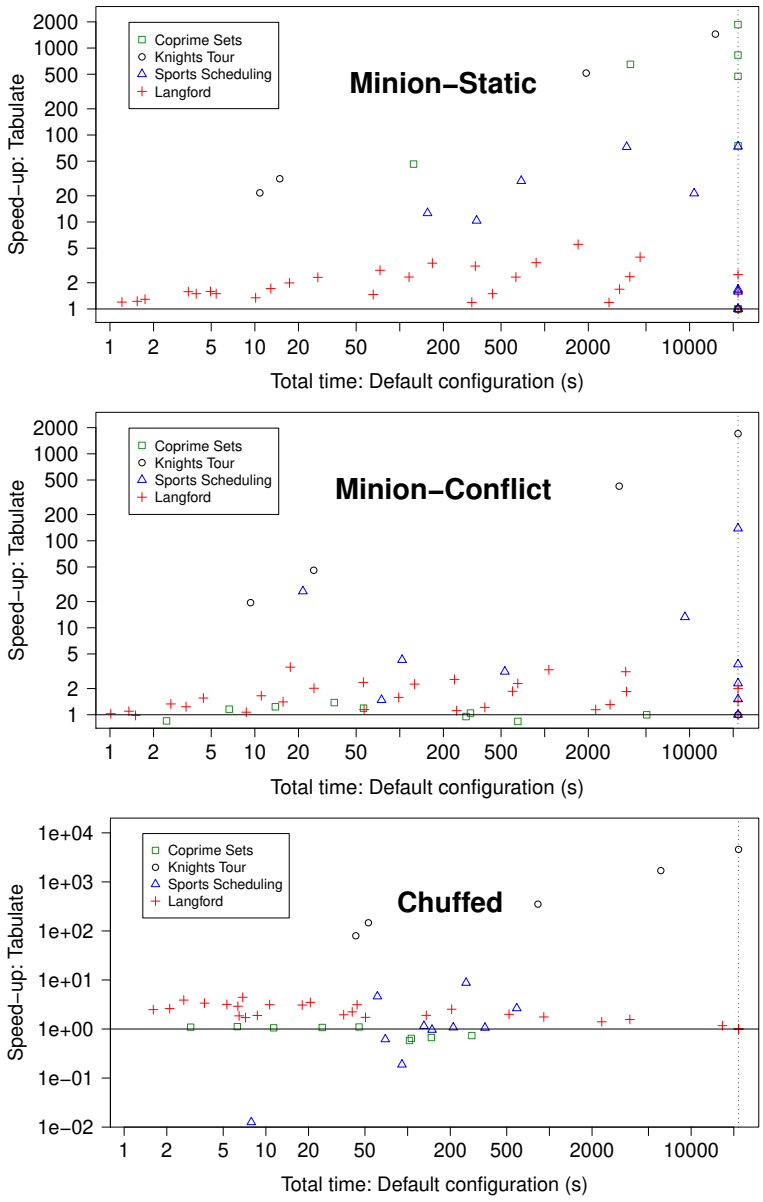

Fig. 1. Tabulate vs Default, total time with Minion solver and static variable ordering (top), Minion solver and Conflict variable ordering (middle), and Chuffed solver with free search (bottom). The x-axis indicates time taken by the default configuration (including both Savile Row and the solver). The y-axis indicates the speed-up obtained by tabulation. Instances that time out are reported as if they completed in 6 hours. The dotted line indicates the time limit of 6 hours; points appearing on the line timed out with the default configuration.

$P[i,j] = P[i,j-1] + i + 1$ constraints (because they overlap with the global allDifferent). Tabulation of these constraints improves propagation and results in improvements for all three solvers.

Coprime Sets Erdös and Sárközy [7] studied a range of problems involving coprime sets. A pair of numbers a and b are coprime if there is no integer $n > 1$ which is a factor of both a and b. The Coprime Sets problem of size k is to find the smallest m such that there is a set of k numbers in $\{m/2 \dots m\}$ that are pairwise coprime. In our model the set is represented as a sequence of integer variables. Each pair of variables a and b has a set of coprime constraints: $\forall d \in \{2 \dots m\}\,(a \not\equiv 0 \pmod{d}) \vee (b \not\equiv 0 \pmod{d})$. Adjacent variables are ordered to break symmetry. We use the instances $k \in \{8 \dots 16\}$. The Identical Scopes heuristic triggers on the coprime constraints (and any symmetry breaking constraint) for each pair of variables. All the original constraints are tabulated.

Static variable ordering follows the sequence from smallest to largest number, so would appear to be a natural choice. However, Minion-Static performs poorly compared to the other two solvers. In this case, tabulation makes the model more robust to the poor variable ordering, speeding it up by over 1000 times in some cases. Tabulation provides no benefit for the other two solvers that already solve the instances relatively well.

Knight's Tour Problem The Knight's Tour Problem on an $n \times n$ chessboard is to visit every square of the board exactly once while making only knight's moves. We use a model where the location of the knight is encoded as a single integer ($nx+y$), we start at location (0,0) and search for a sequence of n^2 distinct locations. We use instances $n \in \{6 \dots 10\}$. The knight's move constraint contains two location variables and uses integer division and modulo to obtain the x and y coordinates. The coordinates are used multiple times in the expression. Identical common subexpression elimination (CSE) substantially improves the model by adding auxiliary variables for the x and y coordinates among others. The default configuration includes identical CSE. The knight's move constraint triggers the Duplicate Variables, Large AST and Weak Propagation heuristics. Tabulation produces a quite different model with no auxiliary variables, much stronger propagation and far better performance with all three solvers.

5 Conclusions

In this paper we have demonstrated that a small set of heuristics can successfully and automatically identify promising subproblems in a constraint model for tabulation, and that these opportunities can be effectively exploited through an automated tabulation method incorporated into the automated constraint modelling system Savile Row. Our heuristics identify the same tabulation opportunities as recent work by Dekker et al. using manual annotations of a MiniZinc model [6]. In addition we have presented four new case studies demonstrating the efficacy of our heuristics and automated tabulation.

Acknowledgements We thank EPSRC for grants EP/P015638/1 and EP/P-026842/1. Dr Jefferson holds a Royal Society University Research Fellowship.

References

1. CSPLib: A problem library for constraints. http://www.csplib.org (1999)
2. Özgür Akgün, Gent, I.P., Jefferson, C., Miguel, I., Nightingale, P., Salamon, A.Z.: Tabulation experimental software and additional results (2018). https://doi.org/10.5281/zenodo.1290656, https://github.com/stacs-cp/cp2018-tabulation
3. Bessiere, C.: Constraint propagation. In: Handbook of Constraint Programming, pp. 29–83. Elsevier (2006)
4. Bessière, C., Hebrard, E., Hnich, B., Walsh, T.: The complexity of reasoning with global constraints. Constraints **12**(2), 239–259 (2007). https://doi.org/10.1007/s10601-006-9007-3
5. Chu, G., Stuckey, P.J., Schutt, A., Ehlers, T., Gange, G., Francis, K.: Chuffed (2018), available from https://github.com/chuffed/chuffed/
6. Dekker, J.J., Björdal, G., Carlsson, M., Flener, P., Monette, J.N.: Auto-tabling for subproblem presolving in MiniZinc. Constraints **22**(4), 512–529 (2017). https://doi.org/10.1007/s10601-017-9270-5
7. Erdös, P., Sárközy, A.: On sets of coprime integers in intervals. Hardy-Ramanujan Journal **16**, 1–20 (1993), https://hal.archives-ouvertes.fr/hal-01108688
8. Gargani, A., Refalo, P.: An efficient model and strategy for the steel mill slab design problem. In: Proceedings of CP 2007. LNCS, vol. 4741, pp. 77–89. Springer (2007). https://doi.org/10.1007/978-3-540-74970-7_8
9. Gent, I.P., Jefferson, C., Kelsey, T., Lynce, I., Miguel, I., Nightingale, P., Smith, B.M., Tarim, S.A.: Search in the patience game 'Black Hole'. AI Communications **20**(3), 211–226 (2007), https://content.iospress.com/articles/ai-communications/aic405
10. Gent, I.P., Jefferson, C., Miguel, I.: Minion: A fast scalable constraint solver. In: Proceedings of ECAI 2006. pp. 98–102. IOS Press (2006), http://ebooks.iospress.nl/volumearticle/2658
11. Gent, I.P., Jefferson, C., Miguel, I., Nightingale, P.: Data structures for generalised arc consistency for extensional constraints. In: Proceedings of AAAI 2007. pp. 191–197. AAAI Press (2007), http://www.aaai.org/Papers/AAAI/2007/AAAI07-029.pdf
12. IBM Knowledge Center: The strong constraint (2017), https://www.ibm.com/support/knowledgecenter/SSSA5P_12.8.0/ilog.odms.ide.help/OPL_Studio/opllang_quickref/topics/tlr_oplsch_strong.html
13. Le Provost, T., Wallace, M.: Domain independent propagation. In: Proceedings of FGCS: International Conference on Fifth Generation Computer Systems, pp. 1004–1011. IOS Press (1992), http://www.webmail.eclipseclp.org/reports/corefgcs.pdf
14. Lecoutre, C.: STR2: optimized simple tabular reduction for table constraints. Constraints **16**(4), 341–371 (2011). https://doi.org/10.1007/s10601-011-9107-6
15. Lecoutre, C., Sais, L., Tabary, S., Vidal, V.: Last conflict based reasoning. In: Proceedings of ECAI 2006. pp. 133–137. IOS Press (2006), http://ebooks.iospress.nl/volumearticle/2665
16. Mohr, R., Masini, G.: Good old discrete relaxation. In: Proceedings of ECAI 1988. pp. 651–656. Pitman Publishing (1988)
17. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: MiniZinc: Towards a standard CP modelling language. In: Proceedings of CP 2007. LNCS, vol. 4741, pp. 529–543. Springer (2007). https://doi.org/10.1007/978-3-540-74970-7_38

18. Nightingale, P., Akgün, Ö., Gent, I.P., Jefferson, C., Miguel, I.: Automatically improving constraint models in Savile Row through associative-commutative common subexpression elimination. In: 20th International Conference on Principles and Practice of Constraint Programming (CP 2014). pp. 590–605. Springer (2014). https://doi.org/10.1007/978-3-319-10428-7_43
19. Nightingale, P., Akgün, O., Gent, I.P., Jefferson, C., Miguel, I., Spracklen, P.: Automatically improving constraint models in Savile Row. Artificial Intelligence **251**, 35–61 (2017). https://doi.org/10.1016/j.artint.2017.07.001
20. Nightingale, P., Rendl, A.: Essence' description (2016), arXiv:1601.02865 [cs.AI]
21. Nightingale, P., Spracklen, P., Miguel, I.: Automatically improving SAT encoding of constraint problems through common subexpression elimination in Savile Row. In: Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming (CP 2015). pp. 330–340. Springer (2015). https://doi.org/10.1007/978-3-319-23219-5_23
22. Régin, J.C.: Generalized arc consistency for global cardinality constraint. In: Proceedings of AAAI 1996. pp. 209–215. AAAI Press (1996), http://www.aaai.org/Papers/AAAI/1996/AAAI96-031.pdf
23. Van Hentenryck, P., Michel, L., Perron, L., Régin, J.: Constraint programming in OPL. In: Proceedings of PPDP 1999: International Conference on Principles and Practice of Declarative Programming. LNCS, vol. 1702, pp. 98–116. Springer (1999). https://doi.org/10.1007/10704567_6