Automatic Detection of At-Most-One and Exactly-One Relations for Improved SAT Encodings of Pseudo-Boolean Constraints*

Carlos Ansótegui[1], Miquel Bofill[2], Jordi Coll[2], Nguyen Dang[3], Juan Luis Esteban[4], Ian Miguel[3], Peter Nightingale[5], András Z. Salamon[3], Josep Suy[2], and Mateu Villaret[2]

[1] University of Lleida, Lleida, Spain
carlos@diei.udl.cat
[2] University of Girona, Girona, Spain
{miquel.bofill,jordi.coll,josep.suy,mateu.villaret}@imae.udg.edu
[3] University of St Andrews, St Andrews, United Kingdom
{nttd,ijm,Andras.Salamon}@st-andrews.ac.uk
[4] Technical University of Catalonia, Barcelona, Spain
esteban@cs.upc.edu
[5] University of York, York, United Kingdom
peter.nightingale@york.ac.uk

Abstract. Pseudo-Boolean (PB) constraints often have a critical role in constraint satisfaction and optimisation problems. Encoding PB constraints to SAT has proven to be an efficient approach in many applications, however care must be taken to encode them compactly and with good propagation properties. It has been shown that at-most-one (AMO) and exactly-one (EO) relations over subsets of the variables can be exploited in various encodings of PB constraints, improving their compactness and solving performance. In this paper we detect AMO and EO relations completely automatically and exploit them to improve SAT encodings that are based on Multi-Valued Decision Diagrams (MDDs). Our experiments show substantial reductions in encoding size and dramatic improvements in solving time thanks to automatic AMO and EO detection.

Keywords: Automatic CSP reformulation · SAT · pseudo-Boolean · at-most-one constraint.

1 Introduction

Solving constraint satisfaction and optimisation problems often requires dealing with Pseudo-Boolean (PB) constraints, either explicitly stated in the original

* Work supported by grants TIN2015-66293-R, TIN2016-76573-C2-1/2-P (MINECO/FEDER, UE), Ayudas para Contratos Predoctorales 2016 (grant number BES2016-076867, funded by MINECO and co-funded by FSE), RTI2018-095609-B-I00 (MICINN/FEDER, UE), and EPSRC EP/P015638/1.

model or as a product of some reformulation process. A successful approach to solving constraint problems is by translation to SAT and the use of SAT solvers. Example tools that support this method include MiniZinc [24, 18], Picat [28], and Savile Row [25]. Ideally, such encodings would be compact (in terms of the number of clauses and additional variables) and would have good propagation properties.

In this paper we focus on efficiently translating PB constraints to SAT within Savile Row, which produces a reformulated SAT model from an input constraint model in the Essence Prime language [26]. There exist several approaches for compactly encoding PB constraints to SAT based on different representations, such as Decision Diagrams [13, 2], Sequential Weight Counters [17], Generalised Totalisers [19], and Polynomial Watchdog schemes [5].

There are also attempts to exploit collateral constraints to shrink these encodings further [1, 8]. In particular, in [8], it is shown how to use existing At-Most-One (AMO) and Exactly-One (EO) relations on subsets of the variables of a PB constraint to obtain very compact decision diagram-based representations. In that work, the authors provide empirical evidence of the utility of using this technique in several scheduling problems. Specifically, they provide specialised SAT Modulo Theories (SMT) encodings exploiting AMO and EO relations. However, these relations are found by hand and are not always obvious.

In this work we propose a technique for exploiting such collateral constraints when encoding PB constraints to SAT in a fully automatic manner. By collateral constraints we mean constraints that are derived from the entire model in some way. They may appear directly in the model, or they may be implied by constraints in the model. One can then use a declarative constraint modelling language and forget about collateral constraints when posting PB constraints. The proposed system is able to automatically identify AMO and EO relations and to take them into account when encoding PB constraints. In particular, we use the approach described in [3] to detect sets of Boolean variables in a SAT formula that model finite-domain variables, which essentially corresponds to detecting the AMO (i.e., cardinality constraints with $\leq$ operator and $k = 1$) and At-Least-One (ALO) relations among a set of Boolean variables. Later, in [7], a method to detect arbitrary cardinality constraints ($k \geq 1$) was introduced. To the best of our knowledge, [7] is the first attempt to apply in practice reformulation techniques through the automatic detection of cardinality constraints. They reformulate the input SAT formula by erasing the clauses entailed by the cardinality constraints detected so far. In our work, we tackle a different goal since our aim is to use the automatically detected cardinality constraints to improve the encoding of more general constraints, specifically PB constraints.

The proposed techniques are embedded in Savile Row. In preparing the SAT encoding Savile Row employs the propagation facilities of the constraint solver Minion [15] in order to identify AMOs, plus a syntactic technique for identifying At-Least-One (ALO) relations (which together with AMOs comprise EO relations). The use of propagation techniques to obtain semantic information has already been used in other scenarios. For example, in [11] unit propagation was

used to deduce sub-clauses from implication graphs, and also unit propagation was used in [14] to detect redundant clauses in SAT formulas.

We apply the technique to several problem classes and highlight the characteristics of each regarding the automatically found AMO and EO relations. Our experiments show dramatic improvements of encoding size and solving time.

2 Preliminaries

Essence Prime is typical of solver-independent constraint modelling languages in providing integer and Boolean variable types, as well as multidimensional matrices of these types. It supports arbitrarily nested arithmetic and logical constraint expressions, as well as a suite of global constraints. Savile Row is able to translate any Essence Prime model into SAT, which we define here.

A *Boolean variable* is a variable than can take truth values 0 (false) and 1 (true). A *literal* is a Boolean variable x or its negation $\neg x$. A *clause* is a disjunction of literals. A *propositional formula in conjunctive normal form* (CNF) is a conjunction of clauses. Any propositional formula can be transformed into CNF.

A CNF formula represents a Boolean function, i.e. a function of the form $f : \{0,1\}^n \to \{0,1\}$. An *assignment* is a mapping of Boolean variables to truth values, which can also be seen as a set of literals (e.g., $\{x = 1, y = 0, z = 0\}$ is usually denoted $\{x, \neg y, \neg z\}$). A *satisfying assignment* of a Boolean function f is an assignment that makes the function evaluate to 1. In particular, an assignment A satisfies a CNF formula F if at least one literal l of each clause in F belongs to A. Such an assignment is called a *model* of the formula.

SAT is the problem of determining if there exists a satisfying assignment for a given propositional formula. Given two formulas F and G, we say that G is a logical consequence of F, written $F \models G$, iff every model of F is also a model of G. We say that two Boolean functions F and G are logically equivalent, denoted $F \equiv G$, if $F \models G$ and $G \models F$.

Unit propagation (UP) is the core deduction mechanism in modern SAT solvers: whenever each literal of a clause but one is false, the remaining literal must be set to true in order to satisfy the clause. We say that G is a logical consequence of F by UP, written $F \models_{UP} G$, iff $F \wedge \neg G$ can be determined to be unsatisfiable by UP.

Savile Row encodes integer variables to provide SAT literals for $(x = a)$ and $(x \leq a)$ for each integer variable x and value a. Each constraint type is then encoded using these SAT literals, as described in [25]. For this work we have added the MDD encoding of PB constraints as defined below.

Definition 1. *A* pseudo-Boolean *(PB) constraint is a Boolean function of the form* $\sum_{i=1}^{n} q_i l_i \diamond K$ *where* K *and the* q_i *are integer constants,* l_i *are literals, and* $\diamond \in \{<, \leq, =, \geq, >\}$.

Definition 2. *An* at-most-one *(AMO) constraint is a Boolean function of the form* $\sum_{i=1}^{n} l_i \leq 1$*, where all* l_i *are literals.*

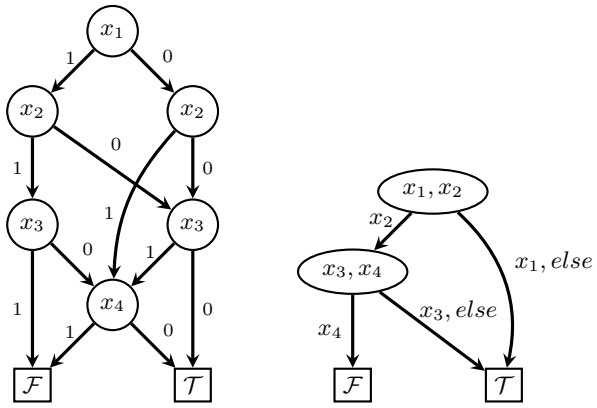

Fig. 1. Left: BDD for $P = 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 7$; Right: MDD for P, assuming $AMO(x_1, x_2)$ and $AMO(x_3, x_4)$, where each x_i branch means choosing $x_i = 1$, and the *else* branches mean choosing $x_i = 0$ for all x_i in the corresponding source node.

Definition 3. *An* at-least-one *(ALO) constraint is a Boolean function of the form* $\sum_{i=1}^{n} l_i \geq 1$*, where all* l_i *are literals.*

Definition 4. *An* exactly-one *(EO) constraint is a Boolean function of the form* $\sum_{i=1}^{n} l_i = 1$*, where all* l_i *are literals.*

One of the best methods to encode PB constraints to SAT is to use Binary Decision Diagrams (BDDs) [13]. In [2] an even more efficient encoding is given for PB constraints where all coefficients, literals and K are positive and the relational operator is $\leq$. Such a constraint has the important property of being *monotonic decreasing*, i.e. any model remains a model after flipping inputs from 1 to 0. In [8] it is shown how the encoding can be dramatically reduced in size in the presence of AMO constraints over subsets of the variables. The improved encoding is based on Multi-Valued Decision Diagrams (MDDs) and is intended also for monotonic decreasing PB constraints. Figure 1 shows an example of this situation. The number of nodes and edges in the second diagram is substantially reduced, and the number of clauses and variables needed to encode the diagram is reduced accordingly. The input of this encoding is a PB constraint, and a partition of its literals, where each part must satisfy an AMO constraint. We will refer to each of the parts as an AMO group.

An interesting particular case occurs when there are not only AMO constraints, but EO constraints over subsets of the variables in the PB constraint. In this case, the number of variables can be reduced [8]: by subtracting the same integer from all the coefficients of a set of variables in an EO relation, as well as from K, we can make at least one coefficient become zero, and then remove the zero-coefficient terms. The result of reducing the set of variables with an EO relation is also an AMO group.

3 Background: AMO and ALO Detection

In this section we present the approach described in [3] to *semantically* detect AMO and ALO constraints in a SAT formula F. The idea is to compute for each literal in F which other literals are entailed by unit propagation (UP). Then an undirected graph $G = (V, E)$ is constructed, where all vertices $u \in V$ are literals of F and an edge $(u, v) \in E$ iff $F \wedge u \models_{UP} \neg v$, i.e. $F \wedge u \wedge \neg v$ can be determined to be unsatisfiable by UP. In other words, if $(u, v) \in E$ then $F \models (\neg u \vee \neg v)$, therefore there is an AMO constraint between literals u and v. We refer to these AMO constraints between two literals as *mutexes*. Accordingly, we refer to the graph G as the *UP-mutex graph* of F.

Recall that a clique of a graph $G = (V, E)$ is a subset of vertices of G such that every pair of vertices u, v are adjacent, i.e. $(u, v) \in E$. Therefore, every clique $C = (V', E')$ in the UP-mutex graph of a SAT formula F corresponds to an AMO $A = \sum_{v \in V'} v \leq 1$ such that $F \models A$. By construction, we know that there is a mutex between all pairs of literals $u, v \in V'$, hence $F \models u + v \leq 1$ and so $F \models \sum_{v \in V'} v \leq 1$. Thus we can identify all the AMO constraints in a SAT formula F that can be detected by UP by finding the cliques in the UP-mutex graph of F.

In [7] the authors propose an approach to detect cardinality constraints (Boolean functions of the form $\sum_{i=1}^{n} l_i \leq k$ where all l_i are literals and $k \geq 1$ is an integer) which generalize AMO constraints. As pointed out by the authors, this methodology is particularly useful for $k > 2$, compared to other approaches for detecting cardinality constraints.

Given a set of literals L of a formula F we can also automatically detect whether $F \models_{UP} \vee_{l \in L} l$, i.e. F entails by UP an ALO constraint on L, by testing whether $F \wedge \bigwedge_{l \in L} \neg l$ is unsatisfiable by UP.

There are two key details in the procedure we have described to semantically detect the AMO constraints in a SAT formula F. First of all, how do we detect the mutexes, i.e. the level of local consistency (power of propagation) we use to find them. Notice that by enforcing stronger consistency than UP we may identify more mutexes and consequently more AMO constraints. Second, how do we detect the cliques in the UP-mutex graph. Depending on the goal of the particular application, the challenge is to properly address these two key details. In the following section, we adapt this procedure to our context by replacing the SAT formula F with a CSP instance, replacing unit propagation with the propagation of the constraint solver Minion [15].

4 AMO and EO Relations in Savile Row

In this section we describe our approach and how it is integrated into Savile Row. As part of this process we must deal with sum constraints that contain integer terms, negative coefficients, and any comparator $\diamond \in \{<, \leq, =, \neq, \geq, >\}$. The end result is a monotonic decreasing PB constraint and a partition of its literals into AMO groups. This is achieved by a sequence of reformulations, where the AMO

groups will arise either from the decomposition of an integer variable, or from the detection of a clique of mutexes in the mutex graph. As described in [25] Savile Row performs two tailoring processes, the first of which uses the constraint solver Minion [15] to filter variable domains, and the second produces output for the desired solver (SAT in this case). Our approach adds mutex detection to Minion, and finds AMO and EO groups during the second tailoring process.

4.1 Mutex Inference

The mutex inference step is performed on Minion's CSP representation of the problem at hand. This representation contains integer constraints that will be transformed into PB constraints later. These integer constraints are of this form $\sum_{i=1}^{n} q_i e_i \diamond K$. An expression e_i may be an integer variable, a Boolean literal, or $(x_i \diamond k_i)$ where x_i is an integer variable or a Boolean literal. Next, any Boolean expressions of the form $(x_i \diamond k_i)$ are replaced with a new Boolean variable b_i and the constraint $b_i \leftrightarrow (x_i \diamond k_i)$ is added to the model. By adding the b_i variables, the mutex detection algorithm is able to see the mutex between $x < 5$ and $x \geq 5$ for example.

Minion is called to perform domain filtering [25] and to find mutexes between literals of Boolean variables. For each Boolean variable b in the CSP, each value of b is assigned in turn and the propagation loop of Minion is called. Consequences of the assignment are propagated through the entire constraint model, including integer variables and global constraints. All assignments of other Boolean variables (to either 0 or 1) by propagation are recorded in the mutex graph G.

Mutex inference is very similar to [3] (described in Section 3) with the SAT formula replaced by the CSP, and unit propagation replaced by Minion's propagation algorithms. Comparing propagation power is not straightforward because it depends on the SAT encoding on the one hand, and fine details of propagators on the other. However, there is one key advantage to using the CSP representation: we avoid generating the (potentially very large) encoding of the problem instance without considering AMO and EO relations. See, for example, the Nurse Scheduling Problem (Section 5.3) where the encoding that uses AMO and EO relations is ten times smaller than the one without.

4.2 Normalisation

To use the MDD encoding referred to in Section 2 we must have *monotonic decreasing* PB constraints in $\leq$ form. Reformulations are required both before and after the AMO and EO groups are constructed. In the first step, all PB and sum constraints are rearranged into the form $\sum_{i=1}^{n} q_i e_i \leq K$ with arithmetic transformations [13].

Terms $q_i e_i$ where e_i is integer are dealt with as follows. Let $q = q_i$ and $e = e_i$. First, if $q < 0$, then $q \leftarrow -q$ and $e \leftarrow -e$. Second, if the smallest possible value c of e is less than 0, then $e \leftarrow e + c$ and K is adjusted by adding qc. Finally, the term qe with n possible values becomes an AMO group of $n-1$ terms containing

$e = k_i$ by enumerating all values k_i except the smallest value, and K is adjusted accordingly.

At this point, all expressions e_i in the constraint are Boolean. All terms $q_i e_i$ where $q_i < 0$ are made positive by replacing with $q_i(1 - \neg e_i)$, then multiplying out and subtracting the constant from both sides. The constraint is now a monotonic decreasing $\leq$ PB constraint, suitable for encoding to SAT via an MDD as described in Section 2. However, the next steps may require inverting the polarity of some Boolean expressions e_i in order to match the detected AMOs, losing the normal form. In this case, the normal form will be restored after making the polarities match.

4.3 AMO and EO Detection

For each PB constraint, we take the subgraph $G' = (V', E')$ of the mutex graph G where V' is a set containing both literals of all Boolean variables in the constraint. The algorithm has a list of vertices L, initially containing all vertices in V'. L is sorted by descending degree in G'. A clique cover is constructed by iterating a greedy clique finding algorithm. To construct one clique, the algorithm takes the first vertex from L then adds as many as possible other vertices in the order of L, breaking ties (where the degree is equal) by choosing the vertex whose coefficient is most common within the clique (as a heuristic to reduce the number of outgoing edges of the corresponding nodes in the MDD). Whenever a vertex v is added to a clique, both v and $\neg v$ are removed from L. The end result is a clique cover containing one literal of each Boolean variable in the constraint.

For each clique in the cover, a new AMO or EO group is built as follows. If the negations of literals in the clique correspond with negations in the PB constraint (or the clique has one literal) then we do (1), otherwise (2).

1. The AMO group is constructed directly from the clique. If all literals in the group form an EO corresponding to an integer variable (i.e. literals correspond to $(x = a)$ or $\neg(x \neq a)$ for all values a of some integer variable x), then we can exploit the EO relation to reduce the size of the group. We delete the term(s) with the smallest coefficient c, and subtract c from K and from the other coefficients within the AMO group.
2. If the negation of the term $q_i e_i$ does not match the literal in the clique, the term is rewritten as $q_i(1 - \neg e_i)$ (and rearranged as above), creating a term with a negative coefficient. Once all terms of the group have the appropriate sign, an EO is created by making a new Boolean variable b (constrained to be true iff all expressions e_i in the group are false) and adding a term $0b$ to the group. All coefficients within the group and K are adjusted by subtracting the smallest coefficient. Terms with coefficient zero are removed to create an AMO group.

The result in all cases is an AMO group whose size is at most the size of the clique. In case (1), if an EO is detected then at least one term can be removed relative to the clique. In case (2), if multiple terms have the smallest coefficient

then the AMO group is smaller than the clique. Each AMO group detected in this way will be added to the model as an AMO constraint.

We find EO groups by a syntactic check in case (1) above. EO groups can also be detected semantically using propagation (Section 3), and the semantic approach may find more EO groups. In our case this would involve calling Minion a second time, with more overhead than the syntactic check.

4.4 Reformulation Example

In this section we give an example of the normalisation and reformulation process that illustrates the described steps and cases. Suppose we have a CSP instance $\mathcal{C}$ with the following variables:

- x which is an integer variable with domain $\{1, 2, 3\}$;
- y which is an integer variable with domain $\{-2, -1, 0, 1\}$; and
- z and t that are Boolean variables.

Suppose $\mathcal{C}$ has the following two constraints to be translated to SAT:

$$C_1 : 2(x = 1) + 4(x = 2) + 3(x = 3) - 3y + 4z + 5t \leq 13$$
$$C_2 : \neg z \vee \neg t$$

Before performing the mutex inference, we replace each of the expressions of the form $(x \diamond k)$ with a Boolean auxiliary variable b, and add the constraint $b \leftrightarrow (x \diamond k)$. C_1 is replaced with the following four constraints:

$$b_1 \leftrightarrow (x = 1)$$
$$b_2 \leftrightarrow (x = 2)$$
$$b_3 \leftrightarrow (x = 3)$$
$$C_3 : 2b_1 + 4b_2 + 3b_3 - 3y + 4z + 5t \leq 13$$

The inference mechanism described in Section 4.1 detects the following mutexes, where the first three come from the decomposition of integer variable x, and the last one is due to constraint C_2:

$$\neg b_1 \vee \neg b_2$$
$$\neg b_1 \vee \neg b_3$$
$$\neg b_2 \vee \neg b_3$$
$$\neg z \vee \neg t$$

The following two AMO relations are inferred from the above mutexes:

$$b_1 + b_2 + b_3 \leq 1$$
$$z + t \leq 1$$

These two AMO relations are added to the model as AMO constraints.

An EO relation is detected among b_1, b_2, and b_3, as described in Section 4.3. The EO relation is converted into an AMO by removing the term with the

smallest coefficient in C_3 (b_1 in this case), and adjusting the coefficients of the other terms (as described in Section 4.3). The two Boolean variables z and t form an AMO group. Finally, the integer variable y with four values will form an AMO group of three terms, as described in Section 4.2.

C_3 is reformulated into C_4 as follows:

$$C_4 : 2b_2 + 1b_3 + 9[y = -2] + 6[y = -1] + 3[y = 0] + 4z + 5t \leq 14$$

Note that the right hand side constant has been adjusted to 14, and the coefficients of the terms corresponding to x and y have been adjusted as well. The variables of C_4 are partitioned into the following three AMO groups:

$$\{b_2, b_3\}$$
$$\{[y = -2], [y = -1], [y = 0]\}$$
$$\{z, t\}$$

If the AMO and EO detection process is enabled, the SAT encoding has 18 variables and 33 clauses. Without the detection, it has 33 variables and 53 clauses. The SAT encoding of the MDD derived from C_4 has only 7 clauses, whereas the MDD derived from the constraint without AMO and EO detection (which has only one non-singleton AMO group derived from y) is encoded with 37 clauses.

5 Experimental Evaluation

In this section we evaluate our approach on four diverse case studies: Combinatorial Auctions (CA), the Multi-Mode Resource-Constrained Project Scheduling Problem (MRCPSP), the Nurse Scheduling Problem (NSP), and the Multiple-Choice Multidimensional Knapsack Problem (MMKP). Each of these problem classes have AMO and EO relations that could be identified by expert modellers, and we show that our system is able to identify them without any human effort. The effects on the size and solving time of the resulting SAT formula are dramatic.

All problems except MRCPSP use a PB objective function. To abstract solving performance from any particular optimisation process of the PB objective function, we converted CA, NSP and MMKP problem classes into decision problems. Specifically, we bound the objective function with the best known value of the objective function, so we are searching for a solution that is as good as the best known solution.

For the decision problems CA, NSP, and MMKP, we use the Glucose 4.1 SAT solver [4]. For MRCPSP, where we minimise an integer variable, we use the MaxSAT solver Open-WBO version 2.0 [23], which uses Glucose 4.1 as its core SAT solver. All the experiments were run on an 8GB Intel® Xeon® E3-1220v2 machine at 3.10 GHz. In a preliminary experiment we ran the SAT solver Lingeling (version bcj) [6] on the CA problem and obtained similar results to those reported below with Glucose.

In our experiments we use three configurations. The first (*PB*) has no AMO or EO detection, however normalisation is always applied when encoding a constraint via an MDD (Section 4.2). The second configuration (*PB(AMO)*) performs AMO detection but not EO detection (i.e. the EO check in step (1) of Section 4.3 is switched off). The third configuration (*PB(EO)*) has both AMO and EO detection.

Reported solving times include both reformulation preprocessing and time spent by the SAT solver.

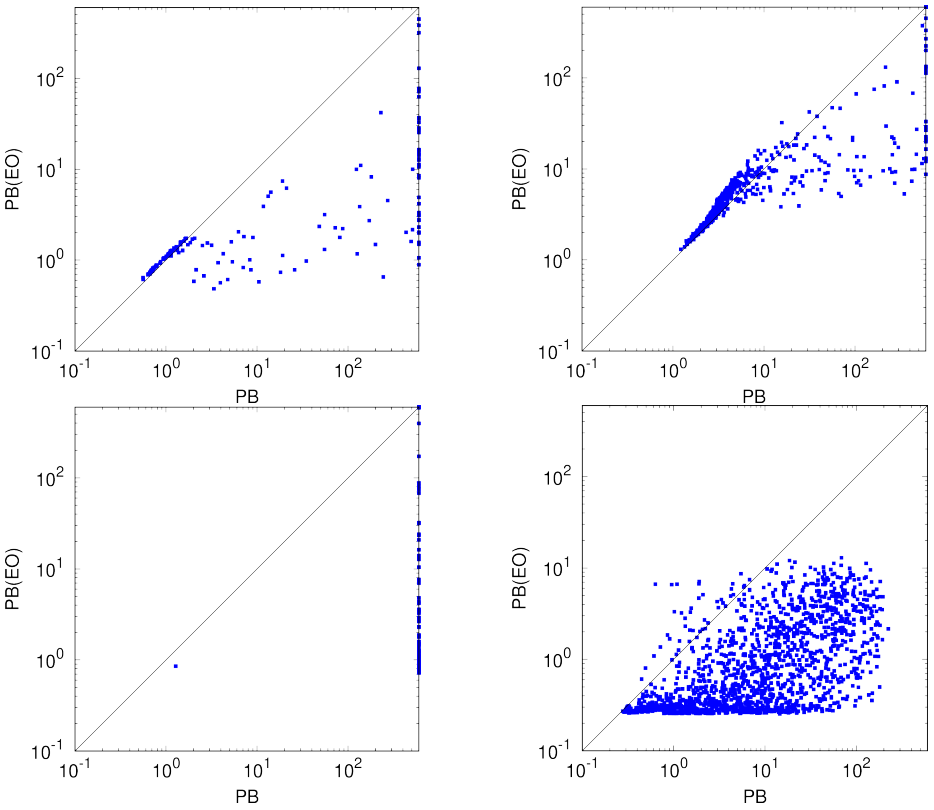

Fig. 2. Scatter plots comparing the median of the solving time among all 10 executions for each instance in the dataset. From left to right and top to bottom: CA, MRCPSP, NSP, MMKP.

5.1 Combinatorial Auctions

The Combinatorial Auctions (CA) problem can be stated as the problem of assigning items to bidders in such a way that the maximum profit is obtained [22].

Table 1. Summary statistics of configurations PB, PB(AMO) and PB(EO) for the four case studies. — indicates time out.

problem	setting	Q1	med	Q3	t.o.	vars	clauses
CA	PB	1.11	3.74	—	42	506	1006
	PB(AMO)	0.98	1.40	3.33	0	47	236
	PB(EO)	0.98	1.40	3.33	0	47	236
MRCPSP	PB	2.55	3.68	9.33	29	54	112
	PB(AMO)	2.89	4.70	8.57	8	12	59
	PB(EO)	2.89	4.68	8.41	8	12	57
NSP	PB	—	—	—	199	116	231
	PB(AMO)	1.30	1.85	6.81	4	26	120
	PB(EO)	0.76	0.86	1.26	3	8	22
MMKP	PB	0.82	4.98	21.09	0	31	62
	PB(AMO)	0.33	0.47	1.53	0	3	17
	PB(EO)	0.28	0.39	1.43	0	2	10

Every bidder makes an offer for a set of items (a package), and it has to be decided whether to sell the whole package to the bidder. It is not allowed to sell only a proper subset of the demanded items. A natural viewpoint to model the problem is to introduce a Boolean variable `sold[b]` for each package `b`, that states whether it is sold or not. Then, the decision version of the problem can be stated as:

```
forAll b1: int(1..nBids-1) .
   forAll b2: int(b1+1..nBids) .
      incompBids[b1,b2] ->
      (!sold[b1] \/ !sold[b2]),

(sum b : int(1..nBids) .
     sold[b] * profit[b] ) >= lb
```

where `nBids` is the number of bids, `profit[b]` is the bid value for package `b`, `incompBids[b1,b2]` is true when two bids have a non-empty intersection, and `lb` is the minimum total profit that is required.

The first constraint ensures that no item is sold in two different packages, or equivalently that every item is sold in at most one package. This will allow Savile Row to detect mutexes between variables `sold[b]` where packages share some item. Typically the sets of packages that contain each particular item will not be disjoint, so the clique cover finding algorithm plays an especially important role when reformulating this problem.

In this work we consider the dataset reported in [9] which was generated using the Combinatorial Auctions Test Suite [22], and have an appropriate complexity to illustrate the effects of our techniques. It consists of 170 instances with the number of bids between 70 and 200. For this problem the syntactic check does not identify any EO relation, so PB(AMO) and PB(EO) are identical.

5.2 MRCPSP

The Multi-mode Resource-Constrained Project Scheduling Problem (MRCPSP) is an iconic problem in the scheduling field [10]. The problem requires deciding a start time (schedule) and an execution mode (schedule of modes) for each job of a project. The jobs are non-preemptive, i.e. they cannot be paused once they have started. Also, the jobs have demands over a set of resources, that can be either renewable, i.e. the amount of resource assigned to a job is recovered once the job finishes, or non-renewable, i.e. availability is not restored when jobs finish. For each job, its duration and its demands depend on the chosen execution mode. The schedule must ensure that a given set of precedence relations between jobs are all satisfied, that the given availability of renewable resources is never surpassed during the execution of the project, and that the given availability of non-renewable resources is enough to supply the demands. Moreover, the project completion time (makespan) must be minimised.

We model the resource constraints as follows. We introduce an auxiliary integer variable `mode[j]` for each job `j`, which represents the selected execution mode for job `j`. To deal with renewable resources constraints we also introduce a Boolean variable `jobActive[j,m,t]` for each job `j`, execution mode `m` and time instant `t` within a scheduling horizon, which is constrained to be true iff job `i` is running in mode `m` at time `t`. The renewable resource constraints are:

```
forAll t: int(0..horizon) .
forAll res: int(1..resRenew) . (
  sum j: int(1..jobs) .
    sum m: int(1..nModes[j]) .
      jobActive[j,m,t]*resUsage[j,m,res]
    ) <= resLimits[res]
```

We model non-renewable resource constraints as:

```
forAll res : int(resRenew+1..nRes) . (
   sum j: int(1..jobs) .
    sum m: int(1..nModes[j]) .
       (mode[j]=m) * resUsage[j,m,res]
   ) <= resLimits[res]
```

where `horizon` is a scheduling horizon which accepts a valid schedule (if the instance is satisfiable), `1..resRenew` and `resRenew+1..nRes` are the sets of renewable and non-renewable resources respectively, `1..jobs` is the set of all jobs, `1..nModes[j]` is the set of available execution modes for job `j`, `resUsage[j,m,res]` is the consumption of job `j` on resource `res` when it runs in mode `m`, and `resLimits[res]` is the availability of resource `res`.

MRCPSP contains many notions of activity and mode incompatibilities, which allow the reformulation process to find AMO constraints on the variables of resource PB constraints. For instance, every activity must run in exactly one

execution mode, and if an activity precedes another they will never run in parallel. Further, two modes of a pair of activities are incompatible if the combined demands for the two modes surpass the availability of some resource.

For this problem we have used the 552 satisfiable instances of the j30 dataset, which is the hardest from PSPLib [21]. These instances contain projects of 30 activities, 3 possible execution modes for each activity, 2 renewable resources and 2 non-renewable resources.

5.3 NSP

The Nurse Scheduling Problem (NSP) is the problem of finding an optimal assignment of nurses to shifts per day considering some coverage and shift preference constraints. There are plenty of variants of this problem depending on the constraints considered [12, 27]. In this work we consider the basic version of the problem where solutions must satisfy all shift coverage constraints, i.e. each shift and day must have a certain number of nurses assigned, and must satisfy the constraint that each nurse only works a certain number of days per week, and must minimise the total penalisation according to the preferences of the nurses.

PB constraints appear in the Essence Prime model when bounding the total amount of penalisation allowed. We use integer variable `nS[n,d]` to state the shift assignment of each nurse `n` and day `d`, and the penalisation constraint is as follows:

```
(sum n: int(1..nNurses) .
  sum d: int(1..nDays) .
   sum st: int(1..nShiftTypes) .
    (nS[n,d]=s) * p[n,d,st] ) <= ub
```

where `nNurses` is the number of nurses, `nDays` the number of days, `nShiftTypes` the number of shift types and `p[n,d,st]` is the penalty of assigning shift `st` to nurse `n` on day `d`. Finally, since we are computing the decision version of NSP, `ub` is the maximum cost allowed. Notice that EO relations occur among the penalties for each nurse and day, since `nS` ranges over integer values from 1 to `nShiftTypes`.

In this work we consider a set of instances from NSPLib, a repository of thousands of NSP instances grouped into classes by several complexity indicators. Details can be found in [27]. We focus on a sample of 200 instances taken uniformly and independently at random from the N25 Set: 25 nurses, 7 days and 4 shift types (including the free shift). Each instance has a minimum number of nurses required per shift and day, and includes the nurses preferences to work on each shift and day (a penalty is between 1 and 4, where 1 is the rank of the most preferred shift).

5.4 MMKP

The Multiple-choice Multidimensional Knapsack Problem (MMKP) is a maximisation problem. Given a set of classes of items and a knapsack with several

capacity-bounded dimensions, it is required to pack exactly one item of each class without surpassing the knapsack capacities. Each item of each class has a given profit, and a weight in each dimension. It is also required to maximise the profit of the chosen items [20]. The decision version of the problem requires that the profit is greater than or equal to a lower bound `lb`.

The PB constraints appear in our Essence Prime model when bounding capacities and profit. We use integer variables `item[c]` to state which item of class `c` has been chosen. The constraints are as follows:

```
forAll d: int(1..nDimensions) . (
 sum c: int(1..nClasses) .
  sum i: int(1..classSize) .
   (item[c]=i) * weight[c,i,d]
 ) <= cap[d],

(sum c: int(1..nClasses) .
  sum i:  int(1..classSize) .
   (item[c]=i) * profit[c,i] ) >= lb
```

where `nDimension` is the number of dimensions, `nClasses` is the number of classes, `classSize` is the number of items in each class (n.b. in this dataset all classes have the same number of items), `weight[c,i,d]` is the weight of item `i` of class `c` for dimension `d`, `cap[d]` is the capacity of dimension `d`, `profit[c,i]` is the profit of item `i` of class `c` and `lb` is the minimum profit to be achieved.

Notice that EO relations occur because `item` ranges over integer values from 1 to `classSize`.

For conducting the experimental evaluation we have chosen the 1983 satisfiable instances from the 2000 instances of dataset (10-5-5-G-R-W) from [16], that contain 10 classes of 5 items each, and the knapsack has 5 dimensions. This dataset turns out to be reasonably hard in comparison to others from the same work that appear to be easy for SAT solvers.

5.5 Experimental Results

Our results in Table 1 show a very significant reduction in the sizes of the SAT formulas for all four studied problems, both in the number of variables and number of clauses, thanks to the AMO and EO detection and reformulation process. The greatest reduction with approach PB(AMO) occurs in CA, where the number of variables is divided by 10 and the number of clauses by 4. In all four problem classes, the reduction in size directly translates to improved solving time. The most extreme case is NSP, in which only one instance is solved within the given timeout if AMO detection is not used, whereas almost all instances are solved with PB(AMO). Only 4 instances reach the time limit with PB(AMO). PB(EO) gives a further size reduction on all problems except CA, and it has a particular impact on NSP, where the additional size reduction reduces the number of clauses by ten times overall.

Figure 2 compares total time (including reformulation and solving) of PB and PB(EO) for every instance of each problem class. The solving time improvements are remarkable for all four problem classes. There are improvements between one and two orders of magnitude in many cases between PB and PB(EO), although there is a small overhead on some of the easiest instances.

6 Conclusion and Future Work

We have presented a fully automatic approach to find and exploit at-most-one (AMO) and exactly-one (EO) relations in SAT encodings of PB constraints. The approach is integrated into Savile Row, a constraint modelling tool that can automatically produce a SAT encoding of any constraint model written in the language Essence Prime. Until now, AMO and EO relations have been exploited for this purpose only in problem-specific encodings constructed by experts. Results show dramatic improvements in SAT formula size and solving time on four problem classes.

In future work we will explore stronger inference mechanisms for the detection of mutexes, which could lead to larger and more effective AMO relations. We also plan to study whether we can reformulate PB constraints more efficiently through detection of cardinality constraints with $k \geq 2$ applying the approach in [7].

References

1. Abío, I., Mayer-Eichberger, V., Stuckey, P.J.: Encoding linear constraints with implication chains to CNF. In: CP: Principles and Practice of Constraint Programming. pp. 3–11. LNCS 9255, Springer (2015). https://doi.org/10.1007/978-3-319-23219-5_1
2. Abío, I., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., Mayer-Eichberger, V.: A new look at BDDs for pseudo-Boolean constraints. Journal of Artificial Intelligence Research pp. 443–480 (2012). https://doi.org/10.1613/jair.3653
3. Ansótegui, C.: Complete SAT solvers for Many-Valued CNF Formulas. Ph.D. thesis, University of Lleida (2004)
4. Audemard, G., Simon, L.: On the Glucose SAT solver. International Journal on Artificial Intelligence Tools **27**(1), 1–25 (2018). https://doi.org/10.1142/S0218213018400018
5. Bailleux, O., Boufkhad, Y., Roussel, O.: New encodings of pseudo-Boolean constraints into CNF. In: SAT: Theory and Applications of Satisfiability Testing. pp. 181–194. LNCS 5584 (2009). https://doi.org/10.1007/978-3-642-02777-2_19
6. Biere, A.: Lingeling. SAT Race (2010)
7. Biere, A., Le Berre, D., Lonca, E., Manthey, N.: Detecting cardinality constraints in CNF. In: SAT: Theory and Applications of Satisfiability Testing. pp. 285–301. LNCS 8561, Springer (2014). https://doi.org/10.1007/978-3-319-09284-3_22
8. Bofill, M., Coll, J., Suy, J., Villaret, M.: Compact MDDs for Pseudo-Boolean Constraints with At-Most-One Relations in Resource-Constrained Scheduling Problems. In: IJCAI. pp. 555–562 (2017). https://doi.org/10.24963/ijcai.2017/78

9. Bofill, M., Palahí, M., Suy, J., Villaret, M.: Solving Intensional Weighted CSPs by Incremental Optimization with BDDs. In: CP: Principles and Practice of Constraint Programming. pp. 207–223. LNCS 8656, Springer (2014). https://doi.org/10.1007/978-3-319-10428-7_17
10. Brucker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E.: Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research **112**(1), 3–41 (1999). https://doi.org/10.1016/S0377-2217(98)00204-5
11. Darras, S., Dequen, G., Devendeville, L., Mazure, B., Ostrowski, R., Saïs, L.: Using Boolean constraint propagation for sub-clauses deduction. In: CP: Principles and Practice of Constraint Programming. pp. 757–761. LNCS 3709, Springer (2005). https://doi.org/10.1007/11564751_59
12. De Causmaecker, P., Vanden Berghe, G.: A categorisation of nurse rostering problems. Journal of Scheduling **14**(1), 3–16 (2011). https://doi.org/10.1007/s10951-010-0211-z
13. Eén, N., Sorensson, N.: Translating pseudo-boolean constraints into SAT. Journal on Satisfiability, Boolean Modeling and Computation **2**, 1–26 (2006), http://satassociation.org/jsat/index.php/jsat/article/view/18
14. Fourdrinoy, O., Grégoire, É., Mazure, B., Saïs, L.: Eliminating redundant clauses in sat instances. In: CPAIOR: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. pp. 71–83. LNCS 4510, Springer (2007). https://doi.org/10.1007/978-3-540-72397-4_6
15. Gent, I.P., Jefferson, C., Miguel, I.: Minion: A fast scalable constraint solver. In: ECAI: European Conference on Artificial Intelligence. Frontiers in Artificial Intelligence and Applications, vol. 141, pp. 98–102. IOS Press (2006), http://www.booksonline.iospress.nl/Content/View.aspx?piid=1654
16. Han, B., Leblet, J., Simon, G.: Hard multidimensional multiple choice knapsack problems, an empirical study. Computers & Operations Research **37**(1), 172–181 (2010). https://doi.org/10.1016/j.cor.2009.04.006
17. Hölldobler, S., Manthey, N., Steinke, P.: A compact encoding of pseudo-Boolean constraints into SAT. In: KI 2012: 35th Annual German Conference on Artificial Intelligence. pp. 107–118. LNCS 7526, Springer (2012). https://doi.org/10.1007/978-3-642-33347-7_10
18. Huang, J.: Universal Booleanization of constraint models. In: CP: Principles and Practice of Constraint Programming. pp. 144–158. LNCS 5202, Springer (2008). https://doi.org/10.1007/978-3-540-85958-1_10
19. Joshi, S., Martins, R., Manquinho, V.: Generalized totalizer encoding for pseudo-Boolean constraints. In: CP: Principles and Practice of Constraint Programming. pp. 200–209. LNCS 9255, Springer (2015). https://doi.org/10.1007/978-3-319-23219-5_15
20. Kellerer, H., Pferschy, U., Pisinger, D.: Multidimensional knapsack problems. In: Knapsack Problems, pp. 235–283. Springer (2004). https://doi.org/10.1007/978-3-540-24777-7_9
21. Kolisch, R., Sprecher, A.: PSPLIB - A Project Scheduling Problem Library. European Journal of Operational Research **96**(1), 205–216 (1997). https://doi.org/10.1016/S0377-2217(96)00170-1
22. Leyton-Brown, K., Shoham, Y.: A test suite for combinatorial auctions. In: Combinatorial auctions, chap. 18, pp. 451–478. The MIT Press (2006)
23. Martins, R., Manquinho, V., Lynce, I.: Open-WBO: A modular MaxSAT solver. In: SAT: Theory and Applications of Satisfiability Testing. pp. 438–445. LNCS 8561, Springer (2014). https://doi.org/10.1007/978-3-319-09284-3_33

24. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard CP modelling language. In: CP: Principles and Practice of Constraint Programming. pp. 529–543. LNCS 4741, Springer (2007). https://doi.org/10.1007/978-3-540-74970-7_38
25. Nightingale, P., Akgün, Ö., Gent, I.P., Jefferson, C., Miguel, I., Spracklen, P.: Automatically improving constraint models in Savile Row. Artificial Intelligence **251**, 35–61 (2017). https://doi.org/10.1016/j.artint.2017.07.001
26. Nightingale, P., Rendl, A.: Essence' description. arXiv:1601.02865 (2016), https://arxiv.org/abs/1601.02865
27. Vanhoucke, M., Maenhout, B.: NSPLib: a nurse scheduling problem library: a tool to evaluate (meta-)heuristic procedures. In: Brailsford, S., Harper, P. (eds.) Operational research for health policy: making better decisions. pp. 151–165. Peter Lang (2007)
28. Zhou, N.F., Kjellerstrand, H.: The picat-sat compiler. In: PADL: International Symposium on Practical Aspects of Declarative Languages. pp. 48–62. LNCS 9585, Springer (2016). https://doi.org/10.1007/978-3-319-28228-2_4