LP2PB: Translating Answer Set Programs into Pseudo-Boolean Theories

Wolf De Wulf Bart Bogaerts
Vrije Universiteit Brussel
Brussels, Belgium
firstname.lastname@vub.be

Answer set programming (ASP) is a well-established knowledge representation formalism. Most ASP solvers are based on (extensions of) technology from Boolean satisfiability solving. While these solvers have shown to be very successful in many practical applications, their strength is limited by their underlying proof system, resolution. In this paper, we present a new tool LP2PB that translates ASP programs into pseudo-Boolean theories, for which solvers based on the (stronger) cutting plane proof system exist. We evaluate our tool, and the potential of cutting-plane–based solving for ASP on traditional ASP benchmarks as well as benchmarks from pseudo-Boolean solving. Our results are mixed: overall, traditional ASP solvers still outperform our translational approach, but several benchmark families are identified where the balance shifts the other way, thereby suggesting that further investigation into a stronger proof system for ASP is valuable.

1 Introduction

Answer set programming (ASP) is a well-established knowledge representation formalism that grew from the observation that stable models [33] of a logic program can be used to encode search problems [59, 62, 49]. ASP is rapidly gaining adoption, with applications in domains such as decision support for the Space Shuttle [63], product configuration [75], phylogenetic inference [45, 11], knowledge management [37], e-Tourism [65], biology [32], robotics [5], and machine learning [41, 12].

The success of ASP can, to a large extend, be explained by two factors. The first factor is a rich, first-order language, ASP-Core2 [13], to express knowledge in, with an easy-to-understand modeling methodology known as generate-define-and-test. The second factor is the availability of a large number of reliable tools — grounders [31, 46] and solvers [28, 3, 16] — that allow to efficiently compute stable models of a given logic program.

Throughout its history, ASP has always benefited from progress in other domains of combinatorial search. For instance, the addition of conflict-driven clause learning (CDCL) [60] to Boolean satisfiability (SAT) solvers is often recognized as one of the most important leaps forward in SAT solving; this technique was very quickly adopted in ASP. In fact, the relation goes two ways, CLASP, a native ASP solver has long been one of the best performing SAT solvers. A recent example of such positive reinforcement between domains is found in recent constraint ASP systems [6], which use techniques from SAT modulo theories [7] and from constraint programming [66] – in particular, lazy clause generation [73].

Next to native solvers, also various ASP tools are available based on *translations* to other formalisms: to SAT [43], to difference logic [44], to mixed integer programming [53], and to SAT modulo acyclicity [27]. Our current work fits in this line, by translating answer set programs into (linear) pseudo-Boolean (PB) constraints [67].

Most modern ASP solvers are built on conflict-driven clause learning and thus on the resolution proof system. This also holds for ASP solvers with native support for aggregates, which typically employ

Submitted to:
ICLP 2020

lazy clause generation techniques, essentially compiling their theory lazily into clauses and henceforth relying on the underlying CDCL solver. The advantage of building on CDCL technology is that this has been researched intensely, and the simplicity of using only clauses allows for highly optimized implementations, resulting in efficient, well-engineered solvers. The disadvantage is that the resolution proof system is known to be weak; for several very simple problems, resolution proofs are exponentially large; the most notorious such problem is the pigeon hole problem. In practice this means that modern CDCL solvers can, for instance, not solve the problem "do 15 pigeons fit in 14 holes?" One way to avoid this limitation, is using symmetry exploitation methods, which are well-researched in SAT [1, 19, 18, 61], and have also been ported to ASP [21, 17], but the detection of symmetries is often very brittle, *e.g.* , adding redundant constraints often removes symmetries. Another option is using a stronger proof system, such as the *cutting planes* proof system [15]. This proof system works on linear constraints over the integers, or, when restricted to $0-1$ variables, on so-called pseudo-Boolean constraints. Recent research in the field of pseudo-Boolean solving has resulted in a new and efficient solver, ROUNDINGSAT [24], that builds on previous work to integrate conflict-driven search with the cutting plane proof system [20, 14, 68, 57, 9]. This recent improvement in psuedo-Boolean solving triggers the question whether answer set programming could also benefit from these techniques.

The main contribution of this paper is the introduction and experimental validation of a new tool LP2PB that translates ground logic programs into pseudo-Boolean theories. This tool is valuable both for the ASP community and for the PB community. For ASP, it enables the use of an extra class of solvers. Furthermore, since we translate into the well-accepted OPB standard format for pseudo-Boolean problems[1], compatibility of future pseudo-Boolean solvers is obtained for free, allowing us to quickly test the potential of novel PB solving techniques for logic programming. Additionally, the OPB format is supported by important industrial tools such as GUROBI [38]. For the PB community, this tool provides access to a new set of applications and benchmarks. Additionally, it establishes answer set programming as *a modelling language for PB solvers*, thereby bypassing the need to write by hand a program that generates benchmarks for every new class of benchmarks considered.

We experimentally validate LP2PB on two classes of benchmarks. On novel ASP models of four benchmark families where the difference between cutting planes and resolution is known to be essential, our approach, unsurprisingly, outperforms traditional ASP solvers. On benchmarks from the latest ASP competition, we find that overall, traditional ASP solvers are still more efficient, but several benchmark families (mainly optimization problems) are found where the cutting plane proof system pays off.

The rest of this paper is structured as follows. In Section 2, we introduce preliminaries on ASP and pseudo-Boolean constraints. In Sections 3 and 4, we discuss our translation and its implementation respectively. Section 5 contains our experiments. In Section 6, we discuss some closely related work and we conclude in Section 7.

2 Preliminaries

A *vocabulary* is a set of symbols, also called *atoms*; vocabularies are denoted by σ, τ. A *literal* is an atom p or its negation $\bar{p}$. An interpretation I of a vocabulary σ is a subset of σ. We use the truth values true (**t**) and false (**f**) and will identify **t** with 1 and **f** with 0, as is common in pseudo-Boolean theories. The truth value of an atom $p \in \sigma$ in an interpretation I (denoted p^I) is 1 if $p \in I$ and 0 otherwise. The truth value of literals, conjunctions of literals, and clauses (disjunctions of literal) are defined as usual.

[1] See `http://www.cril.univ-artois.fr/PB10/format.pdf`.

Pseudo-Boolean Constraints A (linear) pseudo-Boolean constraint over σ is a linear constraint with variables from σ, i.e., an expression of the form

$$\sum_i w_i x_i \sim b \tag{1}$$

with $w_i, b \in \mathbb{Z}$, $x_i \in \sigma$, and $\sim$ one of $<$, $>$, $\leq$, $\geq$, and $=$. The value of a $\sum_{i=1}^n w_i x_i$ in I is, as usual, defined as $\sum_{i=1}^n w_i x_i^I$. A pseudo-Boolean constraint of the form (1) is satisfied in I if $\sum_{i=1}^n w_i x_i^I \sim b$. A pseudo-Boolean theory is a set of pseudo-Boolean constraints. A *model* of a pseudo-Boolean theory $\mathscr{T}$ is an interpretation I such that all constraints are satisfied in I.

Logic Programming A *normal logic program* $\mathscr{P}$ over vocabulary σ is a set of *rules* r of form

$$h \leftarrow a_1 \wedge \cdots \wedge a_n \wedge \overline{b_1} \wedge \cdots \wedge \overline{b_m}. \tag{2}$$

where h, the a_i's, and b_i's are atoms in σ. We call h the *head* of r, denoted head(r), and $a_1 \wedge \cdots \wedge a_n \wedge \overline{b_1} \wedge \cdots \wedge \overline{b_m}$ the *body* of r, denoted body(r). If $n = m = 0$, we simply write h.

Remark 1 *We use the notation $\overline{p}$ for the negation of p. In the context of logic programming, the type of negation used here is often referred to as "negation as failure" or "default negation", referring to the fact that in "good" models of logic programs (called stable models below), an atom p is false by default: it is false unless there is a rule that can derive it. In this work, there is no need to distinguish between different types of negation (indeed, all definitions such as when an interpretation satisfies a literal remain valid) and for uniformity and brevity we thus use the notation $\overline{p}$ which is standard in pseudo-Boolean solving throughout the paper.*

An interpretation I is a *model* of a logic program $\mathscr{P}$ if, for all rules r in $\mathscr{P}$, whenever body(r) is satisfied by I, so is head(r). The *reduct* of $\mathscr{P}$ with respect to I, denoted $\mathscr{P}^I$, is the program that consists of rules $h \leftarrow a_1 \wedge \cdots \wedge a_n$ for all rules of the form (2) in $\mathscr{P}$ such that $b_i \notin I$ for all i. An interpretation I is a *stable model* of $\mathscr{P}$ if it is the $\subseteq$-minimal model of $\mathscr{P}^I$ [33].

In practice, often not just rules of the form (2), but also aggregates are used. In non-ground programs (i.e., programs with first-order variables), they take various forms, but at the propositional level, it is well-known [71, 58] that in order to capture the standard aggregates [13], it suffices to consider only *weight constraint rules*: rules of the form

$$h \leftarrow W \tag{3}$$

where $h \in \sigma$ and W is a pseudo-Boolean constraint $l \leq \sum_i v_i a_i + \sum_i w_i \overline{b_i}$ with h, the a_i's, and the b_i's in σ and with $l, v_i, w_i \in \mathbb{Z}$. Various semantics have been proposed for programs with weight constraint rules; for completeness we here include one. The *FLP-reduct* of a program $\mathscr{P}$ (with weight constraints) with respect to $\mathscr{I}$ is the set of rules of $\mathscr{P}$ whose body is satisified in $\mathscr{I}$. A interpretation $\mathscr{I}$ is an FLP-stable model of $\mathscr{P}$ if it is a minimal model of the FLP-reduct of $\mathscr{P}$ with respect to $\mathscr{I}$. We do stress that the particular choice of semantics for these programs with weight constraints is not relevant in the current work, since we focus on the class of programs on which all proposals coincide. This is discussed in detail in the next section, and we come back to this issue in our discussion on future work in Section 7.

3 Translating Logic Programs into Pseudo-Boolean Theories

Scope and Limitations As can be seen in our definition of rules, we do not consider so-called *disjunctive* logic programs in this paper: the head of a rule is a single atom. For programs where disjunction "behaves nicely", for instance for the classes of head-cycle free [8] and head-elementary-set-free

[29] programs, disjunction in the head can be eliminated by means of an operation called *shifting* [34]. Through the use of LPSHIFT [40], which implements shifting, our tool also works for such programs.

For weight constraint programs, or more generally, programs with different notions of aggregates, many different semantics have been proposed [26, 72, 64, 25, 35, 4]. Following the ASP-Core-2 standard [13], we restrict our attention to programs *without recursion over aggregates* since for such programs all of the aforementioned semantics coincide. Formally, we say that in the context of a logic program $\mathscr{P}$, an atom h *depends directly on* atom h' if there is a rule r (of the form (2) or (3)) in $\mathscr{P}$ where h' occurs in body(r). We say that h *depends* on atom h' if it depends directly on h' or if there exists some h'' such that h depends on h'' and h'' on h'. In this paper, following the ASP-Core-2 standard, we only consider programs such that for each rule of the form (3), no atom h' that occurs in W depends on h.

Translation In order to translate logic programs into pseudo-Boolean theories, we make use of existing frameworks and tools as much as possible, to avoid reinventing the wheel. First of all, we can assume that for each rule of the form (3) in the program, h is uniquely defined by that rule. We can always obtain this situation by introducing a new atom h' and replacing the rule by two rules $h' \leftarrow W$ and $h \leftarrow h'$. This operation is sound for most semantics of logic programs with weight constraint rules, and it is always sound if there is no recursion over aggregates. It now becomes apparent that the aggregates can be "isolated".

Proposition 2 *Let $\mathscr{P}$ be a logic program without recursion over aggregates such that for each weight constraint rule $r \in \mathscr{P}$,* head(r) *has no other defining rules. Let $\mathscr{P}'$ be the logic program obtained from $\mathscr{P}$ by replacing all constraint rules $h \leftarrow W$ by two rules $h \leftarrow \overline{h'}$ and $h' \leftarrow \overline{h}$, where h' is a new atom not occurring in $\mathscr{P}$. Then there is a one-to-one correspondence between the answer sets of $\mathscr{P}$ and the answer sets of $\mathscr{P}'$ in which $h \Leftrightarrow W$ is satisfied for each rule of the form* (3) *in $\mathscr{P}$.*

This (unsurprising) proposition follows directly from well-known splitting results; for instance the seminal work of Lifschitz and Turner [50] or the results of Vennekens et al. [76] for an algebraic variant that is applicable to the semantic characterization of Pelov et al. [64] of logic programs with aggregates.

Proposition 2 shows that we can split the task of translating $\mathscr{P}$ into a pseudo-Boolean theory in two parts: first, we use any off-the-shelve method to translate $\mathscr{P}'$ into a propositional theory, and next, we add an encoding of the constraints of the form $h \Leftrightarrow W$, for instance using constraints of the form

$$h \Leftrightarrow b \leq \sum_{i=1}^{n} w_i l_i$$

is equivalent to the set of pseudo-Boolean constraints

$$b \leq \sum_{i=1}^{n} w_i l_i + M_1 \overline{h}, \qquad b > \sum_{i=1}^{n} w_i l_i - M_2 h$$

when M_1 and M_2 are sufficiently large. The equivalence holds as soon as $M_1 \geq b - \sum_{i=1}^{n} \min(0, w_i)$ and $M_2 > b + \sum_{i=1}^{n} \max(w_i, 0)$. For instance, for such large M_i, in case h is false, the first constraint is trivially satisfied; in case h is true, it reduces to $b \leq \sum_{i=1}^{n} w_i l_i$.

4 Implementation

Our tool accepts input in the LPARSE–SMODELS intermediate format [74]. In case disjunction is present in the head of a rule, it is first eliminated using LPSHIFT [40]. Of course, this is not correct in general, but

only for the classes of programs considered here, where disjunction is head-cycle free (or, more general, head-elementary set free). The input is split into two parts: one part contains all rules containing aggregates (in the LPARSE–SMODELS format these are the constraint rules, weight rules, and minimize rules) while the other part contains all other rules, as well as the other information present in the LPARSE–SMODELS intermediate format (the symbol table, and compute statements). For constraint and weight rules (the former are a special case of the latter) in the first part, the transformation from Proposition 2 is used to split them into a choice rule (the combination of the rules $h \leftarrow \bar{h}', h' \leftarrow \bar{h}$) which is added to the second part and two pseudo-Boolean constraints as described below Proposition 2 to be included in the output. A minimize statement directly corresponds to a linear term, and is hence translated directly into a corresponding minimisation statement in the OPB format. As mentioned in Section 3, we make use of existing tools as much as possible. Therefore, the second part (with the additional rules) is then given to the pipeline LP2NORMAL | LP2LP2 | LP2SAT; the combination of these three tools translates a non-disjunctive logic program into an equivalent propositional theory in CNF [10, 40]. Our tool subsequently transforms each clause produced by LP2SAT into a simple linear constraint and combines this with the linear constraints obtained from the first part to produce a complete pseudo-Boolean theory that characterizes exactly the stable models of the original logic program. We do not describe the complete implementation, but instead discuss a couple of peculiar points.

Auxiliary Variables Since the translation introduces auxiliary variables, the translation happens only after parsing the entire input; at that point the highest used variable number is known; auxiliary variables will be numbered with subsequent numbers.

Multilevel optimization While the ASP-Core-2 standard supports multilevel optimization (expressed in the LPARSE–SMODELS intermediate format by multiple minimise rules), the OPB format has no such construct. We use a well-known technique to reduce multilevel optimization to single-level optimization, namely summing up the different optimization terms but thereby multiplying the optimization terms at higher levels with a coefficient that is large enough to dominate over the terms at the lower levels. An effect of this is that — without postprocessing of the results produced by the pseudo-Boolean solver — the actual values of the optimization function cannot be read out directly from the output.

The closed world assumption Answer set programming uses a form of the closed world assumption: all variables that are not mentioned in a program are false. In propositional logic on the other hand, unmentioned variables can take an arbitrary value. When naively applying the transformation from Proposition 2, this can cause problems. For instance, if in the original program a certain variable only occurs in the body of a weight constraint rule (or in the optimization statement), then after applying the translation that variable no longer occurs in the program to be translated into SAT. Since in the original program, it is implicit that that variable must be false (due to the lack of any rules that derive it), it should still be false after translating. However, our pipeline used to translate to SAT does not enforce this constraint unless is it aware of the existence of that variable. There are two possible ways to fix this: either by including such variables in the symbol table or by manually adding a constraint that makes them false. We implemented the first option.

Unused variables A last point of attention is that LP2SAT, when translating a logic program into CNF makes some simplifications. In particular, in case a variable does not occur in the body of any rule, and that atom is not included in the symbol table (meaning that the user does not care about the value of that

atom), LP2SAT adds a constraint that falsifies this atom. However, since we only give LP2SAT a part of the program, this optimization is no longer correct. This behaviour is again avoided by adding all atoms that occur in rules not given to the LP2SAT pipeline in the symbol table.

5 Experiments

The experiments and set-ups were chosen to shine light on following research questions:

1. How well do modern Pseudo-Boolean solvers perform on ASP models of problems where cutting planes is known to be stronger than resolution?
2. To which extent is the cutting plane proof system promising for traditional ASP?

The benchmarks were ran on the VUB Hydra cluster. Each solver call was assigned a single core on a 10-core INTEL E5-2680v2 (IvyBridge) processor, a timelimit of 20 minutes and a memory-limit of 12GB, thereby matching the limits of the latest ASP competition [30]. The following benchmarks were used:

1. Four benchmark families inspired by the work of Elffers et al. [23], using problems described there, as well as the same types of instances as in that paper (e.g., the shapes of the graphs considered). These four families are known to be (with the right encoding) easy for the cutting-plane proof system in the sense that polynomial cutting plane proofs exist, but are hard for CDCL solvers. All our ASP encodings are straightforward and use aggregates. The four families are:

   Pigeon Hole The problem here is to fit n pigeons in m holes with at most one pigeon residing in each hole. All our instances are unsatisfiable with $n = m+1$.

   Even Colouring This problem takes as input a connected graph in which each vertex has an even degree. The problem is to determine if a black-white colouring of the edges exists such that each nodes has the same number of incident black and white edges. The problem is satisfiable if and only if the number of edges is even. Our instances are long toroidal grids in which one auxiliary vertex is inserted to break a single edge in two. All these instances are thus unsatisfiable.

   Vertex Cover The input to this problem is a connected graph and a number S. The problem is to decide if a size S vertex cover exists, i.e., a subset of the nodes of size S such that each edge is incident to some vertex in the set. We again use long toroidal grids, here with an even number of rows; in that case an instance is satisfiable if and only if $S \geq m \cdot \lceil n/2 \rceil$ where m is the number of rows and n the number of columns. All our instances are unsatisfiable and have $S = m \cdot \lceil n/2 \rceil - 1$.

   Dominating Set This problem again takes a graph and a number S as input. The problem is to decide if the input graph has a size-S dominating set, i.e., a set of vertices such that each vertex is either in the set or adjacent to a vertex in teh set. Our instances are long hexagonal grid. All our instances are unsatisfiable and have $S = \lfloor v/4 \rfloor$ where v is the number of vertices in the graph.

   The instances selected in these four benchmark families all scale linearly, that is, after starting from a small instance, we increase the size of the instance by a fixed step size.

2. All decision and optimization problems from the 2017 ASP competition [30], which includes many benchmarks from earlier competitions, with the exception of:

- Problems including non head-cycle-free disjunction, since those were problems beyond the first level of the polynomial hierarchy that can hence not be translated compactly into pseudo-Boolean theories.
- The video streaming benchmark family, since it contains very high coefficients (higher than what RoundingSAT supports).

For each benchmark family, the 20 instances selected for the competition were used.
All benchmarks and instances are available at `https://github.com/wulfdewolf/lp2pb_benchmarks`.

We compared four solver configurations:

- $\mathbf{C_C}$: gringo | clasp
- $\mathbf{C_{PB}}$: gringo | lp2pb | RoundingSAT
- $\mathbf{C_{N\text{-}PB}}$: gringo | lp2normal | lp2lp2 | lp2sat | RoundingSAT
- $\mathbf{C_{N\text{-}C}}$: gringo | lp2normal | lp2lp2 | clasp

Of each of the used tools, the latest available version was used, i.e. gringo 5.4.0, clasp 3.3.5, lp2normal 2.27, lp2lp2 1.23, lp2sat 1.24, lp2pb 1.0[2], and RoundingSAT at commit fd464d43a[3].

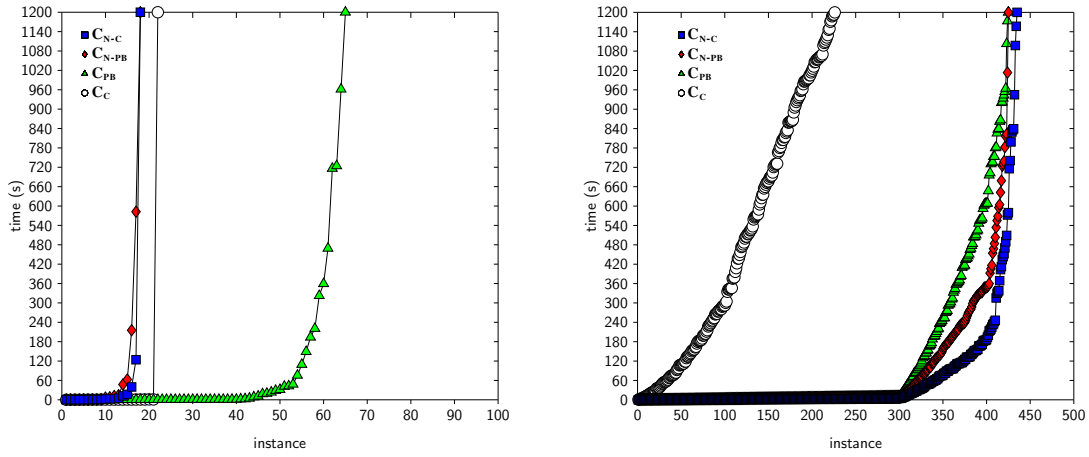
A comparison between $\mathbf{C_C}$ and $\mathbf{C_{PB}}$ should give insights into how a state-of-the-art ASP solvers compares to a state-of-the-art pseudo-Boolean solver after our translation. Interpreting the results of $\mathbf{C_{N\text{-}PB}}$ requires some care. For decision problems, in $\mathbf{C_{N\text{-}PB}}$, the input given to RoundingSAT is a CNF. It is well-known that despite the fact that cutting planes can be exponentially more powerful than resolution, this power is not used by *conflict-driven* pseudo-Boolean solvers on CNF input, where they essentially produce resolution proofs (see e.g., [77]). For *decision problems*, a comparison between $\mathbf{C_{N\text{-}PB}}$ and $\mathbf{C_{N\text{-}C}}$ should thus give an idea of the difference in engineering and optimizations between RoundingSAT and clasp. For *optimization problems*, this comparison does not hold since bounds on the objective function that are added during branch-and-bound search are typically non-clausal. Finally, a comparison between $\mathbf{C_{PB}}$ and $\mathbf{C_{N\text{-}PB}}$ should give an indication of how valuable the pseudo-Boolean constraints (coming from aggregates) are for RoundingSAT, i.e., how much is gained by using our translation compared to a plain CNF translation.

Analysis Cactus plots of the runtimes of the first benchmark set are presented in Figure 1. Overall, these results are consistent with our expectations. The combination of lp2pb and RoundingSAT outperforms resolution-based solvers by far. This is most prominently visible in the Pigeon Hole problem, where no resolution-based configuration solves the problem with 16 pigeons, while RoundingSAT solves all problems up 916 pigeons. The odd one out of the four families is the Even Colouring family, where the normalization-based configurations slightly outperform $\mathbf{C_{PB}}$. Our assumption is that the auxiliary variables introduced by lp2normal change the language of learning and in this way enable short resolution proofs. A similar effect, but less prominent, is seen in the Vertex Cover family, where normalization-based approaches also outperform $\mathbf{C_C}$, but do not reach the performance of $\mathbf{C_{PB}}$.

When examining the results on decision problems, summarized in Table 1, we notice that $\mathbf{C_C}$, i.e. gringo | clasp, outperforms all other configurations on most benchmark families. For problems without aggregates, this is in line of the expectations. But for problems with aggregates our expectation was to see a positive effect of the cutting planes proof system. Also the difference between $\mathbf{C_{PB}}$ and $\mathbf{C_{N\text{-}PB}}$ is very small, suggesting that little to no benefit of the cutting plane proof system is obtained on those benchmarks.

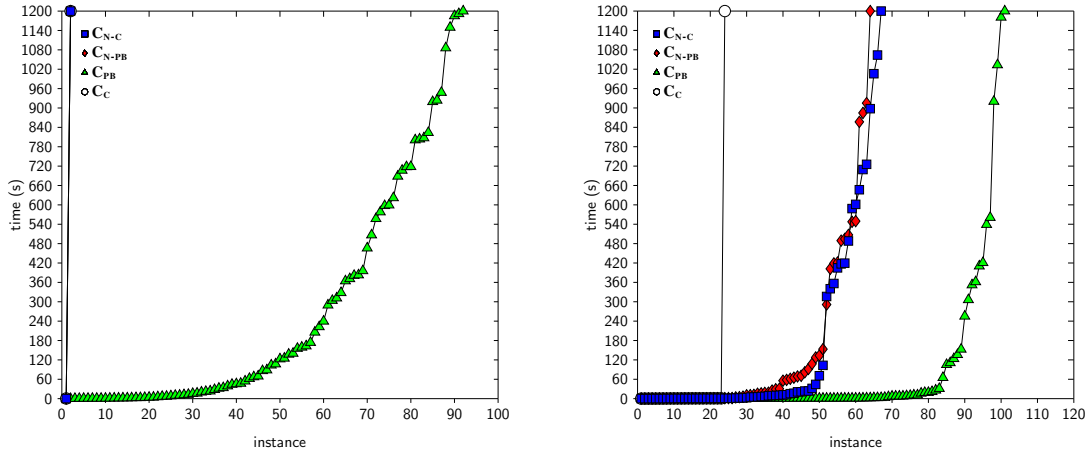
[2] `https://github.com/wulfdewolf/lp2pb`

[3] At the time of the writing, this commit has not been released yet. For reproducability, the binary can be found on our experiment github repository.



(a) Dominating Set

(b) Even Colouring



(c) Pigeon Hole

(d) Vertex Cover

Figure 1: Cactus plots for the first set of benchmark families.

The optimization problems paint a different picture. Next to the number of instances completely solved for each family, Table 2 also shows the number of instances for which a given configuration found the best solution among the four configurations. For System Synthesis, those last values are not included as this is a multilevel optimization problem and the values given by the different solvers are incomparable. In three out of ten families a clear improvement of cutting plane over state-of-the-art ASP solving is visible. Overall, $\mathbf{C_C}$ is still the best solving configuration, but $\mathbf{C_{PB}}$ comes second, performing slightly better than the approaches in which aggregates are normalized. These results suggest that LP2PB, constitutes a valuable extra tool in the ASP toolkit.

6 Related Work

There is a rich history of research on **using SAT solvers to search for computing stable models** of logic programs. One approach works by introducing loop formulas on-the-fly [52, 47, 36] and in fact lies at the basis of most modern native ASP solvers [28, 3]. Another approach is studying **translations of ASP into**

Table 1: *Decision problems of the ASP competition. For each family and set-up pair, 20 instances were ran; this table contains the number of instances for which (un)satisfiability was proven.*

			(UN)SAT Proven			
Family	**#sum?**	**#count?**	$\mathbf{C_C}$	$\mathbf{C_{PB}}$	$\mathbf{C_{N\text{-}PB}}$	$\mathbf{C_{N\text{-}C}}$
Crew Allocation	No	Yes	**18**	17	15	16
Graph Colouring	No	No	**16**	8	8	**16**
Knight Tour With Holes	No	No	**13**	2	2	3
Labyrinth	No	No	**13**	1	3	12
Stable Marriage	No	No	**8**	0	0	3
Visit-all	No	Yes	**18**	13	13	17
Combined Configuration	Yes	Yes	**14**	2	2	1
Graceful Graphs	No	Yes	**13**	8	10	**13**
Incremental Scheduling	Yes	Yes	**14**	13	1	1
Nomistery	No	No	7	**9**	7	8
Partner Units	No	Yes	**11**	10	10	10
Permutation Patternmatching	No	No	**13**	9	8	8
Qualitative Spatial Reasoning	No	No	11	11	**13**	12
Ricochet Robots	No	Yes	10	7	**11**	10
Sokoban	No	Yes	**11**	9	8	10
Total			**190**	119	111	140

SAT that are more compact than the (worst-case) exponential blow-up loop formulas induce [8, 51, 39, 43]. These translations introduce auxiliary variables; some of them induce a one-to-one correpondence between the stable models of the original program and the models of the obtained propositional theory, while others duplicate some models. What all these methods have in common is that aggregates are encoded into clause, either lazily (as done in native solvers, often following the lazy clause generation paradigm [73]), or eagerly, for instance by applying normalization tools [10] that eliminate the aggregates before the actual SAT-translation is called. Our work closely relates to the translation based approach. In fact, internally our tool makes use of the tools of Janhunen and Niemelä [43] to translate the part of the program without aggregates into SAT; the actual translation used can easily be changed in LP2PB. The main difference with the standard translational approaches is that aggregates are not normalized but preserved.

Also DLV2 [2] has an option (`--pre=wbo`) to translate ASP programs into PB theories; however, this translation is limited to tight programs and it cannot handle multilevel optimisation problems.

Another related tool is MINGO [53], which integrates answer set programming and *mixed integer programming* [70], thus allowing more types of constraints (using non-Boolean variables) than PB theories. These non-Boolean variables are used, among others, to encode the level mapping characterization [54] that ensures stability of the obtained models. Unlike our translation, MINGO does not guarantee a one-to-one correspondence between the models of the obtained theory and the stable models of the original program, making it unsuitable for model counting. Our approach does guarantee this, mainly by building on the guarantees from the used translation of aggregate-free logic programs to SAT [43]. Another difference is that part of the focus of MINGO is on developing an extension of the ASP language in which mixed-integer constraints can be written directly in the program. Nowadays, this approach is common in various Constraint-ASP formalisms [22, 48, 6, 42, 69].

Table 2: *Optimization problems. For each family and configuration, the number of instances (out of 20) for which optimality was proven, as well as the number of instances for which a configuration found the best optimization value, among the four configurations. For the System Synthesis family this column cannot be calculated as the problems in this family are multilevel optimisation problems.*

			Optimality Proven				**Best Value Found**			
Family	**#sum?**	**#count?**	$\mathbf{C_C}$	$\mathbf{C_{PB}}$	$\mathbf{C_{N\text{-}PB}}$	$\mathbf{C_{N\text{-}C}}$	$\mathbf{C_C}$	$\mathbf{C_{PB}}$	$\mathbf{C_{N\text{-}PB}}$	$\mathbf{C_{N\text{-}C}}$
Bayesian NL	No	Yes	**15**	5	4	14	**18**	10	9	**18**
Markov NL	No	Yes	**11**	0	0	9	**18**	5	4	16
Supertree	No	Yes	7	5	5	**8**	13	5	5	**17**
Connected Maximum-density Still Life	No	Yes	7	**8**	**8**	2	**19**	8	9	7
Crossing Minimization	No	Yes	13	**19**	**19**	13	15	**20**	19	17
Maximal Clique Problem	No	No	0	0	0	0	0	**16**	13	0
Max SAT	No	Yes	10	**18**	**18**	10	11	**19**	**19**	11
Steiner Tree	No	Yes	**3**	1	1	2	**20**	2	2	3
System Synthesis	Yes	Yes	0	0	0	0	-	-	-	-
Valves Location problem	Yes	Yes	**15**	13	3	6	**20**	13	3	6
Total			**81**	69	58	64	**170**	120	108	110

A final related tool is PBMODELS [55], which also uses **pseudo-Boolean solvers** to find stable models. The main difference is that PBMODELS is designed as a wrapper around a PB solver that iteratively calls the solver for supported models, next checks for stability and if the result is not stable, adds loop formulas, while LP2PB outputs a translation that can be fed to a PB solver to be solved in a single solver call, which benefits the solver's internal constraint learning mechanism.

7 Conclusion and Future Work

One direction for future work is investigating an extension of our translation to support recursive aggregates. The semantics of recursive aggregates constitute an intense topic of debate, as can be witnessed by the number of papers written about them [26, 72, 64, 25, 35, 4]. However, for monotonic (and in fact, *convex* aggregates [56]), most of them agree — the notable exception being [35]. Hence, an extension of our tool that works for recursive aggregates under the condition that they be convex, would be valuable. The most lightweight way to achieve this would be to start from the translation of [53], which builds on the level mapping of Liu and You [54], and modify it to use Boolean variables. An unresolved challenge in that case is how a one-to-one correspondence between the stable models of the program and the models of the resulting theory can be achieved.

Another interesting, but perhaps more ambitious direction for future work is to develop a new native ASP solver that uses the cutting plane proof system under the hood, for instance by developing an extension of ROUNDINGSAT with support for recursive rules.

To conclude, we presented a novel tool, called LP2PB, to translate logic programs into pseudo-Boolean formulas and experimentally validated its performance on a large set of benchmarks. The results are mixed. On the one hand, overall traditional ASP solvers seem to outperform pseudo-Boolean solvers on the benchmark traditionally tackled with ASP. But on the other hand, a couple of benchmark families was identified on which pseudo-Boolean reasoning can provide a real advantage, thus warranting further research into using the cutting plane proof system in ASP solving.

Acknowledgments

The resources and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by the Research Foundation - Flanders (FWO) and the Flemish Government. We are very grateful to Jakob Nordström and Jo Devriendt for interesting discussions on this topic and for providing us with the latest version of RoundingSAT.

References

[1] Fadi A. Aloul, Karim A. Sakallah & Igor L. Markov (2006): *Efficient symmetry breaking for Boolean satisfiability*. *IEEE Transactions on Computers* 55(5), pp. 549–558, doi:10.1109/TC.2006.75.

[2] Mario Alviano, Giovanni Amendola, Carmine Dodaro, Nicola Leone, Marco Maratea & Francesco Ricca (2019): *Evaluation of Disjunctive Programs in WASP*. In: *Proceedings of LPNMR*, *LNCS* 11481, pp. 241–255, doi:10.1007/978-3-030-20528-7_18.

[3] Mario Alviano, Carmine Dodaro, Wolfgang Faber, Nicola Leone & Francesco Ricca (2013): *WASP: A Native ASP Solver Based on Constraint Learning*. In: *Proceedings of LPNMR*, pp. 54–66, doi:10.1007/978-3-642-40564-8_6.

[4] Mario Alviano & Wolfgang Faber (2018): *Aggregates in Answer Set Programming*. *KI* 32(2-3), pp. 119–124, doi:10.1007/s13218-018-0545-9.

[5] Benjamin Andres, David Rajaratnam, Orkunt Sabuncu & Torsten Schaub (2015): *Integrating ASP into ROS for Reasoning in Robots*. In: *Proceedings of LPNMR*, pp. 69–82, doi:10.1007/978-3-319-23264-5_7.

[6] Mutsunori Banbara, Benjamin Kaufmann, Max Ostrowski & Torsten Schaub (2017): *Clingcon: The next generation*. *TPLP* 17(4), pp. 408–461, doi:10.1017/S1471068417000138.

[7] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia & Cesare Tinelli (2009): *Satisfiability Modulo Theories*. In: *Handbook of Satisfiability*, pp. 825–885, doi:10.3233/978-1-58603-929-5-825.

[8] Rachel Ben-Eliyahu & Rina Dechter (1994): *Propositional Semantics for Disjunctive Logic Programs*. *Ann. Math. Artif. Intell.* 12(1-2), pp. 53–87, doi:10.1007/BF01530761.

[9] Daniel Le Berre & Anne Parrain (2010): *The Sat4j library, release 2.2*. *JSAT* 7(2-3), pp. 59–6.

[10] Jori Bomanson (2017): *lp2normal - A Normalization Tool for Extended Logic Programs*. In: *Proceedings of LPNMR*, pp. 222–228, doi:10.1007/978-3-319-61660-5_20.

[11] Daniel R. Brooks, Esra Erdem, Selim T. Erdogan, James W. Minett & Donald Ringe (2007): *Inferring Phylogenetic Trees Using Answer Set Programming*. *J. Autom. Reasoning* 39(4), pp. 471–511, doi:10.1007/s10817-007-9082-1.

[12] Maurice Bruynooghe, Hendrik Blockeel, Bart Bogaerts, Broes De Cat, Stef De Pooter, Joachim Jansen, Anthony Labarre, Jan Ramon, Marc Denecker & Sicco Verwer (2015): *Predicate logic as a modeling language: modeling and solving some machine learning and data mining problems with IDP3*. *TPLP* 15(6), pp. 783–817, doi:10.1017/S147106841400009X.

[13] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Marco Maratea, Francesco Ricca & Torsten Schaub (2020): *ASP-Core-2 Input Language Format*. *TPLP* 20(2), pp. 294–309, doi:10.1017/S1471068419000450.

[14] Donald Chai & Andreas Kuehlmann (2005): *A fast pseudo-Boolean constraint solver*. *IEEE Trans. on CAD of Integrated Circuits and Systems* 24(3), pp. 305–317, doi:10.1109/TCAD.2004.842808.

[15] William J. Cook, Collette R. Coullard & György Turán (1987): *On the complexity of cutting-plane proofs*. *Discrete Applied Mathematics* 18(1), pp. 25–38, doi:10.1016/0166-218X(87)90039-4.

[16] Broes De Cat, Bart Bogaerts, Jo Devriendt & Marc Denecker (2013): *Model Expansion in the Presence of Function Symbols Using Constraint Programming*. In: *Proceedings of ICTAI*, pp. 1068–1075, doi:10.1109/ICTAI.2013.159.

[17] Jo Devriendt & Bart Bogaerts (2016): *BreakID: Static Symmetry Breaking for ASP (System Description)*. In Bart Bogaerts & Amelia Harrison, editors: *Proceedings of ASPOCP*, pp. 25–39.

[18] Jo Devriendt, Bart Bogaerts & Maurice Bruynooghe (2017): *Symmetric Explanation Learning: Effective Dynamic Symmetry Handling for SAT*. In: *Proceedings of SAT*, pp. 83–100, doi:10.1007/978-3-319-66263-3_6.

[19] Jo Devriendt, Bart Bogaerts, Maurice Bruynooghe & Marc Denecker (2016): *Improved Static Symmetry Breaking for SAT*. In: *Proceedings of SAT*, pp. 104–122, doi:10.1007/978-3-319-40970-2_8.

[20] Heidi E. Dixon & Matthew L. Ginsberg (2002): *Inference Methods for a Pseudo-Boolean Satisfiability Solver*. In: *Proceedings of AAAI*, pp. 635–640.

[21] C. Drescher, O. Tifrea & T. Walsh (2011): *Symmetry-breaking Answer Set Solving*. *AI Communications* 24(2), pp. 177–194, doi:10.3233/AIC-2011-0495.

[22] Christian Drescher & Toby Walsh (2011): *Conflict-Driven Constraint Answer Set Solving with Lazy Nogood Generation*. In: *AAAI*, pp. 1772–1773.

[23] Jan Elffers, Jesús Giráldez-Cru, Jakob Nordström & Marc Vinyals (2018): *Using Combinatorial Benchmarks to Probe the Reasoning Power of Pseudo-Boolean Solvers*. In: *Proceedings of SAT*, pp. 75–93, doi:10.1007/978-3-319-94144-8_5.

[24] Jan Elffers & Jakob Nordström (2018): *Divide and Conquer: Towards Faster Pseudo-Boolean Solving*. In: *Proceedings of IJCAI*, pp. 1291–1299, doi:10.24963/ijcai.2018/180.

[25] Wolfgang Faber, Gerald Pfeifer & Nicola Leone (2011): *Semantics and complexity of recursive aggregates in answer set programming*. *AIJ* 175(1), pp. 278–298, doi:10.1016/j.artint.2010.04.002.

[26] Paolo Ferraris (2005): *Answer Sets for Propositional Theories*. In: *Proceedings of LPNMR*, pp. 119–131, doi:10.1007/11546207_10.

[27] Martin Gebser, Tomi Janhunen & Jussi Rintanen (2014): *Answer Set Programming as SAT modulo Acyclicity*. In: *Proceedings of ECAI*, pp. 351–356, doi:10.3233/978-1-61499-419-0-351.

[28] Martin Gebser, Benjamin Kaufmann & Torsten Schaub (2012): *Conflict-driven answer set solving: From theory to practice*. *AIJ* 187, pp. 52–89, doi:10.1016/j.artint.2012.04.001.

[29] Martin Gebser, Joohyung Lee & Yuliya Lierler (2007): *Head-Elementary-Set-Free Logic Programs*. In: *Proceedings of LPNMR*, pp. 149–161, doi:10.1007/978-3-540-72200-7_14.

[30] Martin Gebser, Marco Maratea & Francesco Ricca (2020): *The Seventh Answer Set Programming Competition: Design and Results*. *TPLP* 20(2), pp. 176–204, doi:10.1017/S1471068419000061.

[31] Martin Gebser, Torsten Schaub & Sven Thiele (2007): *GrinGo: A New Grounder for Answer Set Programming*. In: *Proceedings of LPNMR*, pp. 266–271, doi:10.1007/978-3-540-72200-7_24.

[32] Martin Gebser, Torsten Schaub, Sven Thiele & Philippe Veber (2011): *Detecting inconsistencies in large biological networks with answer set programming*. *TPLP* 11(2-3), pp. 323–360, doi:10.1017/S1471068410000554.

[33] Michael Gelfond & Vladimir Lifschitz (1988): *The Stable Model Semantics for Logic Programming*. In: *Proceedings of ICLP/SLP*, pp. 1070–1080.

[34] Michael Gelfond, Halina Przymusinska, Vladimir Lifschitz & Miroslaw Truszczynski (1991): *Disjunctive Defaults*. In: *Proceedings of KR*, pp. 230–237.

[35] Michael Gelfond & Yuanlin Zhang (2014): *Vicious Circle Principle and Logic Programs with Aggregates*. *TPLP* 14(4–5), pp. 587–601, doi:10.1017/S1471068414000222.

[36] Enrico Giunchiglia, Yuliya Lierler & Marco Maratea (2006): *Answer Set Programming Based on Propositional Satisfiability*. *J. Autom. Reasoning* 36(4), pp. 345–377, doi:10.1007/s10817-006-9033-2.

[37] Giovanni Grasso, Salvatore Iiritano, Nicola Leone & Francesco Ricca (2009): *Some DLV Applications for Knowledge Management*. In: *Proceedings of LPNMR*, pp. 591–597, doi:10.1007/978-3-642-04238-6_63.

[38] LLC Gurobi Optimization (2020): *Gurobi Optimizer Reference Manual*.

[39] Tomi Janhunen (2004): *Representing Normal Programs with Clauses*. In: *Proceedings of ECAI*, pp. 358–362.

[40] Tomi Janhunen (2018): *Cross-Translating Answer Set Programs Using the ASPTOOLS Collection*. *KI* 32(2-3), pp. 183–184, doi:10.1007/s13218-018-0529-9.

[41] Tomi Janhunen, Martin Gebser, Jussi Rintanen, Henrik J. Nyman, Johan Pensar & Jukka Corander (2017): *Learning discrete decomposable graphical models via constraint optimization*. *Statistics and Computing* 27(1), pp. 115–130, doi:10.1007/s11222-015-9611-4.

[42] Tomi Janhunen, Roland Kaminski, Max Ostrowski, Sebastian Schellhorn, Philipp Wanko & Torsten Schaub (2017): *Clingo goes linear constraints over reals and integers*. *TPLP* 17(5-6), pp. 872–888, doi:10.1017/S1471068417000242.

[43] Tomi Janhunen & Ilkka Niemelä (2011): *Compact Translations of Non-disjunctive Answer Set Programs to Propositional Clauses*. In: *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning - Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*, pp. 111–130, doi:10.1007/978-3-642-20832-4_8.

[44] Tomi Janhunen, Ilkka Niemelä & Mark Sevalnev (2009): *Computing Stable Models via Reductions to Difference Logic*. In: *LPNMR*, pp. 142–154, doi:10.1007/978-3-642-04238-6_14.

[45] Laura Koponen, Emilia Oikarinen, Tomi Janhunen & Laura Säilä (2015): *Optimizing phylogenetic supertrees using answer set programming*. *TPLP* 15(4-5), pp. 604–619, doi:10.1017/S1471068415000265.

[46] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri & Francesco Scarcello (2006): *The DLV system for knowledge representation and reasoning*. *ACM Trans. Comput. Log.* 7(3), pp. 499–562, doi:10.1145/1149114.1149117.

[47] Yuliya Lierler (2005): *cmodels - SAT-Based Disjunctive Answer Set Solver*. pp. 447–451, doi:10.1007/11546207_44.

[48] Yuliya Lierler (2012): *On the Relation of Constraint Answer Set Programming Languages and Algorithms*. In: *Proceedings of AAAI*.

[49] Vladimir Lifschitz (1999): *Answer Set Planning*. In: *Proceedings of ICLP*, pp. 23–37.

[50] Vladimir Lifschitz & Hudson Turner (1994): *Splitting a Logic Program*. In: *Proceedings of ICLP*, pp. 23–37.

[51] Fangzhen Lin & Jicheng Zhao (2003): *On Tight Logic Programs and Yet Another Translation from Normal Logic Programs to Propositional Logic*. In: *Proceedings of IJCAI*, pp. 853–858.

[52] Fangzhen Lin & Yuting Zhao (2004): *ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers*. *AIJ* 157(1-2), pp. 115–137, doi:10.1016/j.artint.2004.04.004.

[53] Guohua Liu, Tomi Janhunen & Ilkka Niemelä (2012): *Answer Set Programming via Mixed Integer Programming*. In: *Proceedings of KR*.

[54] Guohua Liu & Jia-Huai You (2010): *Level Mapping Induced Loop Formulas for Weight Constraint and Aggregate Logic Programs*. *Fundam. Inform.* 101(3), pp. 237–255, doi:10.3233/FI-2010-286.

[55] Lengning Liu & Miroslaw Truszczynski (2005): *Pbmodels - Software to Compute Stable Models by Pseudo-boolean Solvers*. In: *Proceedings of LPNMR*, pp. 410–415, doi:10.1007/11546207_37.

[56] Lengning Liu & Mirosław Truszczyński (2006): *Properties and Applications of Programs with Monotone and Convex Constraints*. *J. AI Res. (JAIR)* 27, pp. 299–334, doi:10.1613/jair.2009.

[57] Vasco M. Manquinho & João P. Marques Silva (2006): *On Using Cutting Planes in Pseudo-Boolean Optimization*. *JSAT* 2(1-4), pp. 209–219.

[58] Victor Marek, Ilkka Niemelä & Mirosław Truszczyński (2008): *Logic programs with monotone abstract constraint atoms*. *TPLP* 8(2), pp. 167–199, doi:10.1017/S147106840700302X.

[59] Victor Marek & Mirosław Truszczyński (1999): *Stable Models and an Alternative Logic Programming Paradigm*. In: *The Logic Programming Paradigm: A 25-Year Perspective*, Springer-Verlag, pp. 375–398, doi:10.1007/978-3-642-60085-2_17.

[60] João P. Marques-Silva & Karem A. Sakallah (1999): *GRASP: A Search Algorithm for Propositional Satisfiability*. *IEEE Transactions on Computers* 48(5), pp. 506–521, doi:10.1109/12.769433.

[61] Hakan Metin, Souheib Baarir & Fabrice Kordon (2019): *Composing Symmetry Propagation and Effective Symmetry Breaking for SAT Solving*. In: *Proceedings of NFM*, *LNCS* 11460, pp. 316–332, doi:10.1007/978-3-030-20652-9_21.

[62] Ilkka Niemelä (1999): *Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm*. *Ann. Math. Artif. Intell.* 25(3-4), pp. 241–273, doi:10.1023/A:1018930122475.

[63] Monica Nogueira, Marcello Balduccini, Michael Gelfond, Richard Watson & Matthew Barry (2001): *An A-Prolog Decision Support System for the Space Shuttle*. In: *PADL*, pp. 169–183.

[64] Nikolay Pelov, Marc Denecker & Maurice Bruynooghe (2007): *Well-founded and Stable Semantics of Logic Programs with Aggregates*. *TPLP* 7(3), pp. 301–353, doi:10.1017/S1471068406002973.

[65] Francesco Ricca, Antonella Dimasi, Giovanni Grasso, Salvatore Maria Ielpa, Salvatore Iiritano, Marco Manna & Nicola Leone (2010): *A Logic-Based System for e-Tourism*. *Fundam. Inform.* 105(1-2), pp. 35–55, doi:10.3233/FI-2010-357.

[66] Francesca Rossi, Peter van Beek & Toby Walsh, editors (2006): *Handbook of Constraint Programming*. *Foundations of Artificial Intelligence* 2, Elsevier.

[67] Olivier Roussel & Vasco M. Manquinho (2009): *Pseudo-Boolean and Cardinality Constraints*. In Armin Biere, Marijn Heule, Hans van Maaren & Toby Walsh, editors: *Handbook of Satisfiability*, *Frontiers in Artificial Intelligence and Applications* 185, IOS Press, pp. 695–733, doi:10.3233/978-1-58603-929-5-695.

[68] Hossein M. Sheini & Karem A. Sakallah (2006): *Pueblo: A Hybrid Pseudo-Boolean SAT Solver*. *JSAT* 2(1-4), pp. 165–189.

[69] Da Shen & Yuliya Lierler (2018): *SMT-Based Constraint Answer Set Solver EZSMT+ for Non-Tight Programs*. In: *Proceedings of KR*, pp. 67–71.

[70] G. Sierksma, P. van Dam & G.A. Tijssen (1996): *Linear and integer programming: theory and practice*. Monographs and textbooks in pure and applied mathematics, Dekker.

[71] Patrik Simons, Ilkka Niemelä & Timo Soininen (2002): *Extending and implementing the stable model semantics*. *AIJ* 138(1-2), pp. 181–234, doi:10.1016/S0004-3702(02)00187-X.

[72] Tran Cao Son, Enrico Pontelli & Islam Elkabani (2006): *An Unfolding-Based Semantics for Logic Programming with Aggregates*. *CoRR* abs/cs/0605038.

[73] Peter J. Stuckey (2010): *Lazy Clause Generation: Combining the Power of SAT and CP (and MIP?) Solving*. In: *Proceedings of CPAIOR*, pp. 5–9, doi:10.1007/978-3-642-13520-0_3.

[74] Tommi Syrjänen (2000): *Lparse 1.0 User's Manual*. `http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz`.

[75] Juha Tiihonen, Timo Soininen, Ilkka Niemelä & Reijo Sulonen (2003): *A practical tool for mass-customising configurable products*. In: *Proceedings ICED*, pp. 1290–1299.

[76] Joost Vennekens, David Gilis & Marc Denecker (2006): *Splitting an operator: Algebraic modularity results for logics with fixpoint semantics*. *ACM Trans. Comput. Log.* 7(4), pp. 765–797, doi:10.1145/1182613.1189735.

[77] Marc Vinyals, Jan Elffers, Jesús Giráldez-Cru, Stephan Gocht & Jakob Nordström (2018): *In Between Resolution and Cutting Planes: A Study of Proof Systems for Pseudo-Boolean SAT Solving*. In: *Proceedings of SAT*, pp. 292–310, doi:10.1007/978-3-319-94144-8_18.