A Hybrid Approach to Optimization in Answer Set Programming

Paul Saikko[1], **Carmine Dodaro**[2], **Mario Alviano**[3], and **Matti Järvisalo**[1]
[1]HIIT, Department of Computer Science, University of Helsinki, Finland
[2]DIBRIS, University of Genova, Italy
[3]DeMaCS, University of Calabria, Italy

{paul.saikko, matti.jarvisalo}@helsinki.fi, dodaro@dibris.unige.it, alviano@mat.unical.it

Abstract

Answer set programming (ASP) is today a successful approach to knowledge representation and reasoning in various real-world problem domains. ASP offers an expressive rule-based constraint modelling language, supporting concise declarative modelling of both decision and optimization problems within the first or the second level of the polynomial hierarchy. In this paper, we propose a new approach to solving optimization problems via ASP, i.e., to the problem of finding optimal solutions (in terms of optimal answer sets or stable models) under a given weight function over soft atoms (weak constraints). Our approach constitutes the first adaptation of the so-called implicit hitting set approach in the context of ASP. In particular, in contrast to the earlier proposed family of core-guided algorithms for optimization in answer set programming, we present a hybrid approach which makes use of interactions between an ASP decision solver (as an unsatisfiable core extractor) and an integer programming solver (as a minimum-cost hitting set algorithm). We explain how various concepts and features specific to ASP and IP can be harnessed within the approach, including several ways for obtaining better upper and lower bounds during search, with the aim of speeding up the computation of an optimal answer set. By a careful integration of the interactions between state-of-the-art ASP and IP solvers, we show that already our first implementation provides a complementary approach when empirically compared to the currently available solvers supporting optimization in answer set programming.

1 Introduction

Answer set programming (ASP) (Gelfond and Lifschitz 1991; Niemelä 1999; Marek and Truszczyński 1999; Lifschitz 2002; Gelfond and Kahl 2014) is a well-known declarative problem solving paradigm, and has established itself as a successful approach to knowledge representation and reasoning in various real-world problem domains (Balduccini et al. 2001; Koponen et al. 2015; Gebser, Ryabokon, and Schenner 2015; Baral and Uyan 2001; Marileo and Bertossi 2010). Implementations of ASP reasoning systems offer non-monotonic reasoning under the answer set (or stable model) semantics over an expressive rule-based constraint modelling language, supporting concise declarative modelling of both decision and optimization problems within the first or the second level of the polynomial hierarchy.

In particular, computational problems in the real world often give rise to an objective of finding *optimal* solutions in terms of, for example, minimizing use of resources, or maximizing profit or well-being. Early ASP systems have addressed optimization problems in terms of finding optimal stable models under weak constraints, allowing for expressing various types of objective functions uniformly. This fact has been central for developing successful ASP-based approaches to various important real-world problems (Alviano, Dodaro, and Maratea 2017; Abseher et al. 2016; Ricca et al. 2012). The development of effective algorithmic techniques to finding optimal stable models for answer set programs is therefore important for extending the reach of ASP as an already established problem solving paradigm.

Current algorithms are mainly branch and bound, implemented for example in Smodels (Simons, Niemelä, and Soininen 2002), DLV (Leone et al. 2006), and Clasp (Gebser, Kaufmann, and Schaub 2012), and the so-called *core-guided* approaches (Andres et al. 2012; Alviano et al. 2015; Alviano, Dodaro, and Ricca 2015), originally introduced in the field of maximum satisfiability (Fu and Malik 2006; Marques-Silva and Manquinho 2008; Marques-Silva and Planes 2008; Manquinho, Silva, and Planes 2009; Ansótegui, Bonet, and Levy 2009). Our focus here is mainly on the latter type of algorithms.

Currently implemented core-guided ASP optimization algorithms are based on a two-step loop. The first step applies decision procedures developed for ASP for extracting unsatisfiable cores, that is, sets of weak constraints (or soft atoms) that cannot be jointly satisfied by any stable model of the input program. The second step compiles the knowledge expressed by the unsatisfiable core in ASP rules, namely, that at least one of the weak constraints in the core has to be violated under any (optimal) stable model. The different core-guided algorithms available today, as implemented in Clasp and Wasp (Andres et al. 2012; Alviano and Dodaro 2016), differ mainly in terms of the details of the compilation step, and are often more efficient than branch and bound for unweighted optimization problems. On the other hand, if weak constraints in the unsatisfiable core have heterogeneous weights, then the compilation step has to introduce new copies of the weak constraints in order to encode a preference for higher weights, and such copies may deteriorate the performance of the solver.

Beyond branch-and-bound and core-guided approaches, there have been attempts to use integer programming (IP) solvers for finding optimal stable models (Liu, Janhunen, and Niemelä 2012). Specifically, the task of finding an optimal stable model can be monolithically encoded into the task of finding an optimal solution to an IP: linear inequalities are used to declaratively represent the stable models of the ASP program in input, while weighted weak constraints are compiled into the objective function of the IP (under minimization or maximization).

In this paper, we propose a new approach to solving optimization problems in ASP. Specifically, we present the first adaptation of the *implicit hitting set* (IHS) (Karp 2010; Moreno-Centeno and Karp 2013; Saikko, Wallner, and Järvisalo 2016) approach in the context of ASP, motivated by the success of IHS algorithms in MaxSAT solving (Davies and Bacchus 2011; Davies 2013; Saikko 2015; Saikko, Berg, and Järvisalo 2016). While IHS could be considered a core-guided approach, its second step is not a compilation step, but instead consists of computing a minimum-cost hitting set of the extracted unsatisfiable cores; the hit weak constraints are temporarily not enforced in the subsequent extraction step, which either terminates with a new unsatisfiable core, or with an optimal stable model.

In contrast to the previously proposed family of core-guided algorithms for optimization in ASP, IHS is a hybrid approach that makes use of interactions between an ASP decision solver and an IP solver. In a nutshell, the ASP solver is used as an unsatisfiable core extractor, and the IP solver computes minimum-cost hitting sets of the extracted unsatisfiable cores. Rather than relying only on one type of declarative solver (either an ASP decision procedure or an IP solver), the approach aims at taking advantage of the "best of both worlds": an ASP solver is used as an efficient proof search engine for the task of core extraction, while the task of handling potentially very heterogeneous weights is passed on to an IP solver. A further main advantage of the proposed approach is that the ASP program in input is not modified at all for the whole duration of search for an optimal stable model. This is in contrast to the core-guided algorithms currently available in ASP systems, whose compilation step brings in extra rules to the input program; such extra rules make the subsequent core extraction steps harder, and potentially bloat up the program in case of heterogeneous weights.

In the following, we explain how various concepts and features specific to ASP and IP can be harnessed within the approach, including several ways for obtaining better upper and lower bounds during the search, with the aim of speeding up the computation of an optimal stable model. By a careful integration of the interactions between the state-of-the-art ASP solver Wasp and IP solver CPLEX, we show that a resulting implementation of the approach provides a competitive and complementary approach when empirically compared to the currently available solvers supporting optimization in ASP.

This paper is organized as follows. We start with preliminaries on ASP, and briefly recall a previous core-guided algorithm (Section 2). We then give an overview of a basic IHS approach to optimization in ASP, and describe in detail various search techniques and optimizations to the basic algorithm which are central for making the approach competitive in practice (Section 3). We provide details of our first implementation of the proposed approach, and overview the results of an empirical evaluation of the approach from various perspectives (Section 4). Finally, before conclusions, we review related work (Section 5).

2 ASP and Optimization

In this section we recall the syntax and semantics of answer set programs to the extent necessary and as is convenient for the rest of the paper. We opt for soft atoms instead of weak constraints as weak constraints are normalized in terms of soft atoms by modern grounders. The semantics of a logic program with soft atoms is given by the so-called *optimal stable models*. Finally, we recall the notion of unsatisfiable cores, and give an example of a core-guided algorithm as currently available in modern ASP solvers.

Answer Set Programs

Let $\mathcal{A}$ be a fixed, countable set of (propositional) *atoms* including $\bot$. A *literal* ℓ is either an atom p, or its negation $\sim p$, where $\sim$ denotes *default negation*. Let $\overline{\ell}$ denote the complement of ℓ, i.e., $\overline{p} := \sim p$, and $\overline{\sim p} := p$, for all $p \in \mathcal{A}$. For a set L of literals, $\overline{L} := \{\overline{\ell} \mid \ell \in L\}$, $L^+ := L \cap \mathcal{A}$, and $L^- := \overline{L} \cap \mathcal{A}$. A *program* Π consists of a set of *disjunctive* and *choice rules*, which have respectively the forms

$$p_1 \vee \cdots \vee p_m \leftarrow \ell_1, \ldots, \ell_n \quad (1)$$

$$\{\ell_1, \ldots, \ell_n\} \leq k \quad (2)$$

where $p_1, \ldots, p_m$ are distinct atoms, $\ell_1, \ldots, \ell_n$ are distinct literals, and $k, m, n \geq 0$. For a rule r of the form (1), the disjunction $p_1 \vee \cdots \vee p_m$ is called the *head* of r, denoted $H(r)$; the conjunction $\ell_1, \ldots, \ell_n$ is the *body* of r, denoted $B(r)$. With a slight abuse of notation, $H(r)$ and $B(r)$ also denote the sets of their elements. A *constraint* is a rule of the form (1) such that $H(r) \subseteq \{\bot\}$. For a rule r of the form (2), let $\mathit{lits}(r)$ be $\{\ell_1, \ldots, \ell_n\}$, and $\mathit{bound}(r)$ be k. Let $\mathit{heads}(\Pi, p)$ be the set of rules $r \in \Pi$ of the form (1) such that $p \in H(r)$.

An *interpretation* I is a subset of $\mathcal{A} \setminus \{\bot\}$; atoms in I are assigned true, and those in $\mathcal{A} \setminus I$ are assigned false. I is a model of a rule r of the form (1), denoted $I \models r$, if $H(r) \cap I \neq \emptyset$ whenever $B(r)^+ \subseteq I$ and $B(r)^- \cap I = \emptyset$. I is a model of a rule r of the form (2), denoted $I \models r$, if $|\mathit{lits}(r)^+ \cap I| + |\mathit{lits}(r)^- \setminus I| \leq \mathit{bound}(r)$. I is a model of a program Π, denoted $I \models \Pi$, if it is a model of all rules in Π. The definition of stable models is based on the following notion of reduct. Let Π be a program, and I be an interpretation. The *reduct* of Π with respect to I, denoted Π^I, is obtained from Π by deleting each rule r of the form (1) such that $B(r)^- \cap I \neq \emptyset$, by replacing each rule r of the form (2) with rules of the form $p \leftarrow$ for all $p \in \mathit{lits}(r) \cap I$, and removing negated atoms in the remaining rules. An interpretation I is a stable model of Π if $I \models \Pi$ and there is no $J \subset I$ such that $J \models \Pi^I$. We denote by $SM(\Pi)$ the set of stable models of Π. If $SM(\Pi) \neq \emptyset$ then Π is *coherent*, otherwise it is *incoherent*.

Example 2.1. Let Π_1 be the program

$$\begin{array}{lll} a \vee c \leftarrow {\sim}b, {\sim}d & \bot \leftarrow s_a, {\sim}a & \{s_a, s_b, s_c, s_d\} \leq 4 \\ a \leftarrow c, {\sim}d & \bot \leftarrow s_b, {\sim}b & \\ c \leftarrow a, {\sim}b & \bot \leftarrow s_c, {\sim}c & \\ b \vee d \leftarrow {\sim}c & \bot \leftarrow s_d, {\sim}d & \end{array}$$

We have that $I_1 = \{b, s_b\}$ and $I_2 = \{d, s_d\}$ are among the stable models of Π_1. The program reduct $\Pi_1^{I_1}$ is $\{a \leftarrow c$, $c \leftarrow a$, $\bot \leftarrow s_a, {\sim}a$, $\bot \leftarrow s_c, {\sim}c$, $\bot \leftarrow s_d, {\sim}d$, $s_b \leftarrow\}$, and it can be checked that no strict subset of I_1 is a model of $\Pi_1^{I_1}$. A similar observation holds for $\Pi_1^{I_2}$. ■

Let $\mathcal{W}$ be a finite set of (soft) atoms, such that each atom $p \in \mathcal{W}$ is associated with a positive integer denoted $weight(p)$. The *cost* of an interpretation I for $\mathcal{W}$ is $\mathcal{W}(I) := \sum_{p \in \mathcal{W} \setminus I} weight(p)$. For any pair I, J of interpretations, $J <_{\mathcal{W}} I$ if $\mathcal{W}(J) < \mathcal{W}(I)$. I is an *optimal stable model* of a program Π with respect to a set $\mathcal{W}$ of soft atoms if $I \in SM(\Pi)$ and there is no $J \in SM(\Pi)$ such that $J <_{\mathcal{W}} I$. We denote by $OSM(\Pi, \mathcal{W})$ the set of optimal stable models of Π with respect to $\mathcal{W}$.

Example 2.2 (Continuing Example 2.1)**.** Let $\mathcal{W}_1$ be $\{s_a, s_b, s_c, s_d\}$, and $weight$ be $\{s_a \mapsto 1,\ s_b \mapsto 2,\ s_c \mapsto 4,\ s_d \mapsto 8\}$. It can be checked that $OSM(\Pi_1, \mathcal{W}_1)$ is $\{I_2\}$, as for example $\mathcal{W}_1(I_2) = 7 < 13 = \mathcal{W}_1(I_1)$. ■

Unsatisfiable Cores and Core-Guided Algorithms

Intuitively, an *unsatisfiable core* (or simply core) witnesses the fact that a set of (soft) atoms cannot be jointly satisfied by any stable model of the program in input. Formally, for a program Π and a set S of atoms, a set $C \subseteq S$ is an unsatisfiable core of Π with respect to S if $\Pi \cup \{\bot \leftarrow {\sim}p \mid p \in C\}$ is incoherent, that is, there is no stable model of Π containing all atoms in C. We denote by $cores(\Pi, S)$ the set of cores of Π with respect to S.

Example 2.3 (Continuing Example 2.2)**.** The unsatisfiable cores of Π_1 and $\mathcal{W}_1$ are $\{s_a, s_b\}$, $\{s_a, s_d\}$, $\{s_b, s_c\}$, $\{s_b, s_d\}$, $\{s_c, s_d\}$, and their supersets. ■

Modern ASP solvers implement several algorithms for computing optimal stable models via unsatisfiable core analysis. All of them are based on the following underlying idea: a set of soft atoms that cannot be jointly satisfied, i.e., an unsatisfiable core, is identified. The reason of unsatisfiability represented by the core is removed by modifying the program and the soft atoms. This process is repeated until a stable model is found, which is also guaranteed to be optimal. The definition and implementation of such algorithms take advantage of the fact that modern ASP solvers accept as input a set S of atoms, called *assumptions*, in addition to the usual logic program Π. They return a stable model I of Π such that $S \subseteq I$ if it exists, or otherwise an unsatisfiable core of Π with respect to S if such an I does not exist.

An example of such an algorithm is ONE, presented as Algorithm 1, whose main steps are described next. Every atom p is associated with a weight $w(p)$, initially set to zero if $p \notin \mathcal{W}$, and to $weight(p)$ otherwise. The ASP solver is then called to search for a stable model containing all soft atoms. If an unsatisfiable core $\{p_0, \ldots, p_n\}$ is found, at least one of $p_0, \ldots, p_n$ must be false in any optimal stable model, and hence the lower bound is increased by the smallest weight among $w(p_0), \ldots, w(p_n)$. Such a quantity is removed from $p_0, \ldots, p_n$ and assigned to n new soft atoms $s_1, \ldots, s_n$ (note that there are $n + 1$ atoms in the core, while only n new soft atoms are introduced). The new soft literals and the choice rule $\{{\sim}p_0, \ldots, {\sim}p_n, s_1, \ldots, s_n\} \leq n + 1$ enforce the next call to function *solve* to search for a stable model satisfying at least n literals among $p_0, \ldots, p_n$. Moreover, symmetry breakers of the form $\bot \leftarrow s_i, {\sim}s_{i+1}$ are also added to Π, so that s_i is true if and only if at least $n - i + 1$ literals among $p_0, \ldots, p_n$ are true. The program is processed again until a stable model is found, at which point the algorithm terminates. Note that the program Π is assumed to be coherent to simplify the presentation.

Algorithm 1: Unsatisfiable core analysis with ONE

```
Input : A coherent program $\Pi$, and a nonempty set of soft atoms $\mathcal{W}$.
Output: An optimal stable model $I \in OSM(\Pi, \mathcal{W})$.
1  for $p \in \mathcal{A}$ do $w(p) := 0$;
2  for $p \in \mathcal{W}$ do $w(p) := weight(p)$;
3  loop
4      $(I, C) := solve(\Pi, \{p \in \mathcal{A} \mid w(p) \neq 0\})$;
5      if $I \neq \bot$ then return $I$;
6      Let $C$ be $\{p_0, \ldots, p_n\}$ (for some $n \geq 0$), and $s_1, \ldots, s_n$ be $|C| - 1$ fresh atoms;
7      $mw := \min\{w(p) \mid p \in C\}$;
8      for $i \in [0..n]$ do $w(p_i) := w(p_i) - mw$;
9      for $i \in [1..n]$ do $w(s_i) := mw$;
10     $\Pi := \Pi \cup \{\{{\sim}p_0, \ldots, {\sim}p_n, s_1, \ldots, s_n\} \leq n + 1\} \cup \{\bot \leftarrow s_i, {\sim}s_{i-1} \mid i \in [2..n]\}$;
```

Example 2.4 (Continuing Example 2.3)**.** At the beginning, $w(a) = w(b) = w(c) = w(d) = 0$ and $w(s_a) = 1$, $w(s_b) = 2$, $w(s_c) = 4$, $w(s_d) = 8$. Assume that the first call to $solve(\Pi_1, \{s_a, s_b, s_c, s_d\})$ returns $(\bot, \{s_c, s_d\})$. Then a fresh atom s_1 is created and $w(s_c) = 0$, $w(s_d) = 4$, $w(s_1) = 4$. Moreover, $\{{\sim}s_c, {\sim}s_d, s_1\} \leq 2$ is added to Π_1. Assume that the subsequent call to $solve(\Pi_1, \{s_a, s_b, s_d, s_1\})$ returns $(\bot, \{s_a, s_b\})$. Then a fresh atom s_2 is created and $w(s_a) = 0$, $w(s_b) = 1$, $w(s_2) = 1$. Moreover, $\{{\sim}s_a, {\sim}s_b, s_2\} \leq 2$ is added to Π_1. Assume that the subsequent call to $solve(\Pi, \{s_b, s_d, s_1, s_2\})$ returns $(\bot, \{s_b, s_2\})$. Then a fresh atom s_3 is created and $w(s_b) = 0$, $w(s_2) = 0$, $w(s_3) = 1$. Moreover, $\{{\sim}s_b, {\sim}s_2, s_3\} \leq 2$ is added to Π_1. The subsequent call to $solve(\Pi_1, \{s_d, s_1, s_3\})$ returns $(I_2, \emptyset)$ and the algorithm terminates returning I_2. ■

3 Implicit Hitting Set Approach for ASP

We now detail an implicit hitting set approach to optimization in ASP which is based on iteratively alternating between calls to an unsatisfiable core extractor (implemented via an ASP solver) and a minimum-cost hitting set algorithm (implemented via IP). We will first give a basic overview of

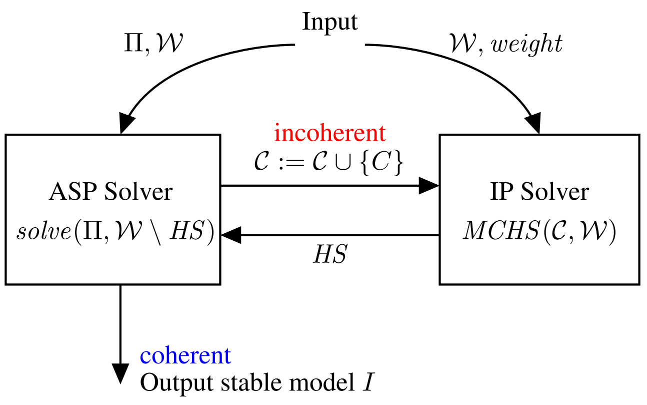

Figure 1: Overview of the implicit hitting set approach to ASP optimization

the approach, and will later on describe several search techniques that are important in practice for making an efficient solver with this approach.

The Basic Approach

As a starting point, the approach is based on the following observations. Any stable model I of a program Π omits at least one atom of every core C of Π with respect to any set $\mathcal{W}$ of atoms (this is a direct consequence of the definition of unsatisfiable core given in the previous section). Stated differently, if $I \in SM(\Pi)$, then $\mathcal{W} \setminus I$ forms a *hitting set* of the set $cores(\Pi, \mathcal{W})$ of all cores of Π with respect to $\mathcal{W}$. Recall that a hitting set of a set S of sets intersects (or hits) each set of S. Hence, if $I \in SM(\Pi)$, then $(\mathcal{W} \setminus I) \cap C \neq \emptyset$ for all $C \in cores(\Pi, \mathcal{W})$. We define the cost of such a hitting set, denoted $cost(\mathcal{W} \setminus I)$, as the sum of the weights of its atoms, and therefore we have that $cost(\mathcal{W} \setminus I) = \mathcal{W}(I)$. Then we can establish the following duality between optimal stable models of Π with respect to $\mathcal{W}$, and *minimum-cost hitting sets* (MCHS) of $cores(\Pi, \mathcal{W})$.

Theorem 3.1. *Let Π be a program, and $\mathcal{W}$ be a set of soft atoms. Then $I \in OSM(\Pi)$ if and only if $\mathcal{W} \setminus I$ is an MCHS of $cores(\Pi, \mathcal{W})$.*

Let $\mathcal{C}$ be a subset of $cores(\Pi, \mathcal{W})$. The computation of an MCHS of $\mathcal{C}$ can be passed on to an IP solver by means of the integer linear program

$$\begin{aligned} \text{minimize} & \sum_{p \in \mathcal{W}} weight(p) \cdot b_p & \\ \text{subject to} & \sum_{p \in C} b_p \geq 1 & \forall C \in \mathcal{C}, \\ & b_p \in \{0,1\} & \forall p \in \mathcal{W}, \end{aligned}$$

where each soft atom p is associated to a new binary variable b_p. Note that the weights of the soft atoms are used to define the objective function, while each core in $\mathcal{C}$ is associated with an inequality. We denote by $MCHS(\mathcal{C}, \mathcal{W})$ a call to the IP solver for the above program.

The implicit hitting set algorithm for ASP, outlined in Figure 1, uses of sets $\mathcal{C}$ and HS to store respectively the extracted cores and one of its hitting sets, and performs the following steps.

1. $\mathcal{C} := \emptyset$, $HS := \emptyset$;
2. $(I, C) := solve(\Pi, \mathcal{W} \setminus HS)$;
3. if $I = \bot$, then $\mathcal{C} := \mathcal{C} \cup \{C\}$, $HS := \{p \mid b_p = 1$ in the output of $MCHS(\mathcal{C}, \mathcal{W})\}$, and go to step 2;
4. I is given in output, and denoted by $IHS(\Pi, \mathcal{W})$.

Initially both $\mathcal{C}$ and HS are empty. The ASP solver is initialized with the program Π and the assumptions are set to $\mathcal{W} \setminus HS$, and the IP solver is initialized with the objective function associated with $\mathcal{W}$ (and their weights). The algorithm then alternates invoking the two solvers: the ASP solver searches for a core C of Π with respect to $\mathcal{W} \setminus HS$; if found, C is added to $\mathcal{C}$. The IP solver is asked to compute a new minimum-cost hitting set HS of $\mathcal{C}$. The algorithm terminates once the ASP solver does not find a core of Π with respect to $\mathcal{W} \setminus HS$, that is, a stable model I (containing the atoms in $\mathcal{W} \setminus HS$) is found.

Proposition 3.1. *The implicit hitting set algorithm for ASP halts.*

Proof. As in the previous section, we assume without loss of generality that Π is coherent in order to simplify the presentation. Hence, there exists some $I \in SM(\Pi)$. The hitting set HS hits each core in $\mathcal{C}$, so each call to the ASP solver (step 2) results in either a new core $C \in cores(\Pi, \mathcal{W}) \setminus \mathcal{C}$ or a stable model I. The algorithm must eventually find a stable model and consequently halt because $cores(\Pi, \mathcal{W})$ is finite ($cores(\Pi, \mathcal{W}) \subseteq 2^{\mathcal{W}}$). □

Theorem 3.2. *Let Π be a coherent program, and $\mathcal{W}$ be a set of soft literals. Then $IHS(\Pi, \mathcal{W})$ is in $OSM(\Pi, \mathcal{W})$.*

Proof. For the sake of a contradiction, if $IHS(\Pi, \mathcal{W}) \in SM(\Pi) \setminus OSM(\Pi, \mathcal{W})$, then there would be $I \in SM(\Pi)$ with $\mathcal{W}(I) < \mathcal{W}(IHS(\Pi, \mathcal{W}))$. Let $HS = \mathcal{W} \setminus I, HS' = \mathcal{W} \setminus IHS(\Pi, \mathcal{W})$ be the hitting sets of $cores(\Pi, \mathcal{W})$ associated with I and $IHS(\Pi, \mathcal{W})$ respectively, and let $\mathcal{C}$ be the cores extracted by the algorithm. Since $\mathcal{C} \subseteq cores(\Pi, \mathcal{W})$ is an invariant of the algorithm, we have that HS and HS' are hitting sets of $\mathcal{C}$ as well. However, $\mathcal{W}(I) < \mathcal{W}(IHS(\Pi, \mathcal{W}))$ implies that $cost(HS) < cost(HS')$, which contradicts the fact that HS' is a MCHS of $\mathcal{C}$. □

Example 3.1 (Continuing Example 2.3)**.** At the beginning, both $\mathcal{C}$ and HS are empty. Assume that the first call to $solve(\Pi_1, \mathcal{W}_1)$ returns $(\bot, \{s_c, s_d\})$. Now, $\mathcal{C} = \{\{s_c, s_d\}\}$, and hence $HS = \{s_c\}$. Say that the subsequent call to $solve(\Pi_1, \{s_a, s_b, s_d\})$ returns $(\bot, \{s_a, s_b\})$. Then, $\mathcal{C} = \{\{s_c, s_d\}, \{s_a, s_b\}\}$, and $HS = \{s_a, s_c\}$. Assume that the subsequent call to $solve(\Pi_1, \{s_b, s_d\})$ returns $(\bot, \{s_b, s_d\})$. Thus $\mathcal{C} = \{\{s_c, s_d\}, \{s_a, s_b\}, \{s_b, s_d\}\}$, and $HS = \{s_b, s_c\}$. Say that $solve(\Pi_1, \{s_a, s_d\})$ returns $(\bot, \{s_a, s_d\})$. We have $\mathcal{C} = \{\{s_c, s_d\}, \{s_a, s_b\}, \{s_b, s_d\}, \{s_a, s_d\}\}$, and $HS = \{s_a, s_b, s_c\}$. Finally, $solve(\Pi_1, \{s_d\})$ returns $(I_2, \emptyset)$, and the algorithm terminates returning I_2. ■

Algorithm 2: Bounds–based implicit hitting set algorithm for ASP optimization

```
Input : A coherent program $\Pi$, and a nonempty set of soft atoms $\mathcal{W}$.
Output: An optimal stable model $I \in OSM(\Pi, \mathcal{W})$.
1  $\mathcal{C} := \emptyset$;   $HS := \emptyset$;   $LB := 0$;   $UB := \infty$;
2  while $LB < UB$ do
3      loop
4          $(I, C) := solve(\Pi, \mathcal{W} \setminus HS)$;
5          if $I \neq \bot$ then
6              if $\mathcal{W}(I) < UB$ then
7                  $UB := \mathcal{W}(I)$;
8                  $I_{best} := I$;
9              break;
10         $HS := HS \cup C$;
11         $\mathcal{C} := \mathcal{C} \cup \{C\}$;
12     $HS := MCHS(\mathcal{C}, \mathcal{W})$;
13     $LB := \mathcal{W}(\mathcal{W} \setminus HS)$;
14 return $I_{best}$;
```

Search Techniques

The basic approach can be refined through a *bounds-based* view. In fact, costs of optimal hitting sets found by the IP solver give increasing lower bounds on the cost of optimal stable models. On the other hand, upper bounds on the cost of optimal stable models are naturally given by any computed stable model. Hence termination of the algorithm can be stated in terms of these bounds: termination happens once the upper and lower bounds match. For the implicit hitting set algorithm, this happens when Π is coherent with assumptions $\mathcal{W} \setminus HS$, and a stable model of cost $cost(HS)$ is found.

The bounds-based view allows for the integration of several search techniques for obtaining better bounds during the computation and speeding up the convergence to an optimal stable model. We describe such techniques in this section. The first two techniques also have the added benefit of yielding intermediate solutions during search.

Core Minimization. The computation of minimum-cost hitting sets takes advantage of a simple destructive algorithm to minimize cores (Bakker et al. 1993): for each atom p in an extracted core C, we check if $C \setminus \{p\}$ is also a core; if this is the case, we remove p from C. After this process, C is minimal, that is, no subset of C is a core. Moreover, if an atom p cannot be removed from C, then Π is coherent with assumptions $C \setminus \{p\}$, and we find an answer set and a corresponding upper bound (Alviano and Dodaro 2016). However, in some cases the check with assumptions $C \setminus \{p\}$ is computationally hard. In practice, we mitigate this potential problem by imposing a resource limit on the minimization process and leave the core only partially minimized if the limit is exceeded.

Disjoint Cores. Calls to the IP solver are expensive, and therefore we do not call the IP solver after each extracted core, but instead we first extract a disjoint set of cores. Such a strategy is shown in Algorithm 2, and it is essentially the "disjoint" strategy of (Saikko 2015) applied to ASP; it can be also seen as an extension of the preliminary disjoint cores analysis step of Wasp (Alviano and Dodaro 2016). The algorithm maintains upper and lower bounds, initially set to ∞ and 0, respectively (line 1). The main loop of the algorithm (lines 2–12) is repeated until the bounds match, and computes non-overlapping cores of Π with respect to $\mathcal{W} \setminus HS$ (the inner loop in lines 3–10). Specifically, soft atoms of extracted cores are added to HS (line 10), which guarantees that those atoms will not occur in subsequent cores extracted in the same iteration of the main loop. By searching for disjoint cores, we may find a stable model which possibly improves the upper bound (lines 7–8) and which—regardless of whether the upper bound is improved—terminates the inner loop (line 5). After that, an optimal hitting set is computed (line 11), the lower bound is possibly improved (line 12), and the main loop is repeated. We note that stable models found by Algorithm 2 give upper bounds that allow for *reduced cost fixing* (as explained next), or even let the algorithm halt earlier (when the bounds match there is no need to extract more cores). Moreover, disjoint sets of cores may even lead to better lower bounds because no hitting set can hit multiple cores in a disjoint set with a single atom.

Example 3.2 (Continuing Example 2.3)**.** At the beginning $LB = 0$ and $UB = \infty$. Let $solve(\Pi_1, \mathcal{W}_1)$ return $(\bot, \{s_c, s_d\})$. Then $HS = \{s_c, s_d\}$ and $\mathcal{C} = \{\{s_c, s_d\}\}$. Now let $solve(\Pi_1, \{s_a, s_b\})$ return $(\bot, \{s_a, s_b\})$. Therefore $HS = \mathcal{W}_1$ and $\mathcal{C} = \{\{s_c, s_d\}, \{s_a, s_b\}\}$. Then let $solve(\Pi_1, \emptyset)$ return $(I_1, \emptyset)$, and therefore UB is set to 13 and I_{best} is updated. The call to $MCHS(\{\{s_c, s_d\}, \{s_a, s_b\}\}, \mathcal{W}_1)$ returns $\{s_a, s_c\}$; therefore $HS = \{s_a, s_c\}$ and LB is updated to 5. The algorithm then starts a second iteration of the main loop. ■

Reduced Cost Fixing is an established technique in IP solving (Danzig, Fulkerson, and Johnson 1954; Crowder, Johnson, and Padberg 1983; Nemhauser and Wolsey 1999), and was recently applied in the context of MaxSAT (Bacchus et al. 2017). Here we use reduced cost fixing to simplify the input ASP program during the execution of the implicit hitting set algorithm. In a nutshell, the technique works on the linear programming (LP) relaxation of the hitting set IP (that is, the IP without integrality constraints), and aims at fixing the value of some variables by combining an optimal solution of the LP and the current upper bound. The reduced cost $rc(b_p)$ of a variable b_p in an optimal LP solution quantifies the minimum amount by which a unit increase of the value of b_p would increase the cost of the LP. Since an optimal solution of the LP provides a lower bound lb for the IP, and also a lower bound $lb + rc(b_p)$ for all variables b_p assigned 0, we check whether $lb + rc(b_p) > UB$, where UB is the cost of the best stable model computed so far. If this is the case, we fix $b_p = 0$ in the IP, and also reduce the search space of the ASP solver by adding the integrity constraint $\bot \leftarrow p$.

Clark's Completion based Bounds. The reduced cost fixing procedure depends on good upper and lower bounds to be effective. The lower bound given by a minimum-cost hitting set is usually very low in the first steps of computation due to the small number of extracted cores. With the goal of enabling earlier application of reduced cost fixing, we derive an initial lower bound from the Clark's completion of Π, a notion capturing the supported models of Π as an overapproximation of the stable models of Π (Clark 1977). For our purposes, we target Clark's completion towards linear programs in order to use an LP solver to obtained lower bounds from the completion. The Clark's completion of a program Π is denoted $Comp(\Pi)$, and comprises the following constraints: for each atom $p \in \mathcal{A}$, $p \in \{0,1\}$ (each atom is either true or false); $\bot = 0$ ($\bot$ is false); for each disjunctive rule $r \in \Pi$,

$$\sum_{p \in H(r) \cup B(r)^-} p + \sum_{p \in B(r)^+} (1-p) \geq 1,$$

and for each choice rule $r \in \Pi$,

$$\sum_{p \in lits(r)^+} p + \sum_{p \in lits(r)^-} (1-p) \leq bound(r)$$

(each rule is satisfied); for each atom $p \in \mathcal{A} \setminus \{\bot\}$ not occurring positively in any choice rule of Π,

$$(1-p) + \sum_{r \in heads(\Pi,p)} r_p \geq 1,$$

and for $r \in heads(\Pi, p)$, $r_p \in \{0,1\}$,

$$r_p + \sum_{p' \in (H(r)\setminus\{p\}) \cup B(r)^-} p' + \sum_{p' \in B(r)^+} (1-p') \geq 1$$

$$(1-r_p) + (1-p') \geq 1 \quad \forall p' \in (H(r) \setminus \{p\}) \cup B(r)^-$$
$$(1-r_p) + p' \geq 1 \quad \forall p' \in B(r)^+$$

(true atoms are supported; $r_p = 1$ if and only if rule r supports p). In particular, this is a relaxation of Π in that its solution gives a lower bound for the cost of optimal stable models of Π with respect to $\mathcal{W}$. Solving the Clark's completion using an LP solver is efficient, and gives lower bounds which can improve subsequent search by e.g. enabling more reduced cost fixing.

Non-core Constraints. The Clark's completion may also provide additional information for the IP solver, as constraints involving only soft atoms can be added to the IP. For example, if Π contains a rule of the form $W \leftarrow$ for some $\emptyset \subset W \subseteq \mathcal{W}$, we can instruct the IP solver that no hitting set contains every $p \in W$, i.e., $\sum_{p \in W} b_p < |W|$. Such constraints possibly allow the IP solver to find hitting sets hitting several cores of Π even without explicitly extracting the cores. We note that these constraints are not necessarily cores of Π with respect to $\mathcal{W}$, and therefore the IP being solved is not a pure hitting set problem, though its solutions are still hitting sets of $cores(\Pi, \mathcal{W})$.

4 Experiments

We overview results of an empirical evaluation of the implicit hitting set approach to optimization in ASP, considering the effectiveness of the approach from multiple viewpoints: the impact of the different search techniques included in our implementation; comparison with the current state of the art solvers for optimization in ASP; and the robustness of the approach in terms of the impact of high-resolution weights. The experiments were run on a cluster of machines with Intel Xeon E5-2680 processors using CentOS Linux 7.4. All solvers are subject to per-instance time and memory limits respectively set to 30 minutes and 50 GB.

We implemented the implicit hitting set approach to optimization in ASP, referred to in the following as ASP-HS, using Wasp 2.0 as the core extractor and CPLEX 12.7 (IBM 2017) as the underlying IP and LP solver for computing hitting sets and for obtaining bounds via LP relaxations. All of the search techniques described in Section 3 are integrated in ASP-HS.

We used the following benchmarks from the 2015 and 2017 ASP Competitions (Gebser, Maratea, and Ricca 2017; 2015): Bayes, Markov, SuperTree, and TravelingSalesman (2017 ASP Competition); SteinerTree, ValvesLocation, and VideoStreaming (2015 ASP Competition); SystemSynthesis, StillLife, CrossingMinimization, MaximalClique, and ADF-WF (2015 ASP Competition). Moreover, we used instances of causal discovery (Hyttinen, Eberhardt, and Järvisalo 2014) as a further problem domain referred to as CausalGraph, as well as instances of Fastfood and OpenDoors from the Asparagus repository (`https://asparagus.cs.uni-potsdam.de/`). All instances but those of ADF-WF, MaximalClique, CrossingMinimization, StillLife and SystemSynthesis are weighted.

Impact of Search Techniques. We first investigate the marginal impact of the individual search techniques described in Section 3 on the efficiency of ASP-HS. For this, we ran the default ASP-HS with all of the techniques enabled, and versions with exactly one of the following techniques disabled: disjoint cores (*No disjoint*); bounding techniques including reduced cost fixing (*No reduced cost*) and Clark's completion based bounds (*No Clark LB*); core minimization (*No minimize*); and non-core constraints (*No non-core*). The results are shown in Table 1 and Figure 2. We observe that core minimization and disjoint core computations make the greatest marginal contributions in terms of the number of solved instances. In fact, both of these techniques seem crucial for the CausalGraph problem domain. Furthermore, core minimization alone is important for SuperTree, while disjoint cores alone is crucial for TravelingSalesman. We also observed that the resource limits set on minimization had a notable impact only on the TravelingSalesman instances. The positive effects of lower bounding techniques and integration of non-core constraints is less pronounced, but still both allow for solving more instances without introducing significant overhead in any benchmark.

Comparison with State of the Art. We compare with Wasp 2.0, using the core-guided algorithm ONE, and Clasp 3.3.2, using both the core-guided algorithm OLL (*usc*) and its default branch-and-bound algorithm (*bb*). The solvers were run using their default settings. The results are shown in Table 2 and Figure 3. Overall, ASP-HS is very competitive when compared to other core-guided approaches. In fact, ASP-HS complements all of the other solvers, and

Table 1: Impact of individual search techniques on the performance of ASP-HS. For each variant, “#s” gives the number of instances solved within the per-instance time limit, and “time” the cumulative running times on all instances, incl. timeouts.

	ASP-HS		No non-core		No Clark LB		No reduced cost		No disjoint		No minimize	
Family	#s	time	#s	time	#s	time	#s	time	#s	time	#s	time
Weighted												
Bayes1000 (20)	11	17793.4	9	21403.9	11	17658.6	11	17591.5	**14**	16573.4	11	19074.5
CausalGraph (20)	**18**	8891.7	17	8521.5	17	8748.1	14	12197.1	8	23941.8	10	20308.7
Fastfood (20)	**9**	20490.9	**9**	20498.1	**9**	20486.6	**9**	20880.1	8	22473.5	8	22977.6
Markov (20)	1	35607.2	0	36000.0	1	35739.1	1	35880.0	**2**	34508.5	0	36000.0
MaxSAT (20)	**12**	17690.5	10	21667.5	**12**	17624.9	**12**	17642.5	10	20485.2	10	19950.3
OpenDoors (20)	**20**	297.5	**20**	297.3	**20**	283.4	**20**	318.3	**20**	408.8	**20**	316.6
SteinerTree (20)	0	36000.0	0	36000.0	0	36000.0	0	36000.0	0	36000.0	0	36000.0
SuperTree (20)	**5**	27358.7	**5**	27364.6	**5**	27556.6	**5**	27338.1	**5**	27735.5	0	36000.0
TravelingSalesman (20)	10	19510.3	**11**	17890.8	10	19208.0	10	19104.6	5	27455.5	9	20445.0
ValvesLocation (20)	**15**	10470.8	**15**	10497.8	**15**	9942.4	**15**	10480.0	14	11768.2	**15**	11315.8
VideoStreaming (20)	**11**	16201.9	**11**	16202.7	**11**	16201.3	**11**	16202.2	**11**	16202.0	**11**	16201.7
All Weighted	**112**	210313	107	216344	111	209450	108	213634	97	237552	94	238590
Unweighted												
ADF-WF (20)	**20**	2701.8	19	2984.7	**20**	1753.1	**20**	3113.1	19	3129.5	19	4089.2
CrossingMinimization (20)	**19**	2788.5	**19**	2781.6	**19**	3032.1	**19**	2766.6	18	7404.0	18	3686.7
MaximalClique (20)	0	36000.0	0	36000.0	0	36000.0	0	36000.0	0	36000.0	0	36000.0
StillLife (20)	**2**	35531.5	1	35756.1	1	35380.3	0	36000.0	0	36000.0	0	36000.0
SystemSynthesis (20)	0	36000.0	0	36000.0	0	36000.0	0	36000.0	0	36000.0	0	36000.0
All Unweighted	**41**	113022	39	113523	40	112166	39	113880	37	118534	37	115776

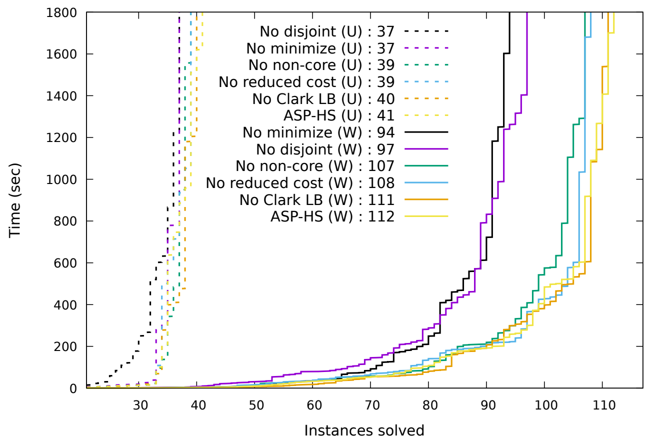

Figure 2: Impact of individual search techniques in ASP-HS on weighted (W) and unweighted (U) instances.

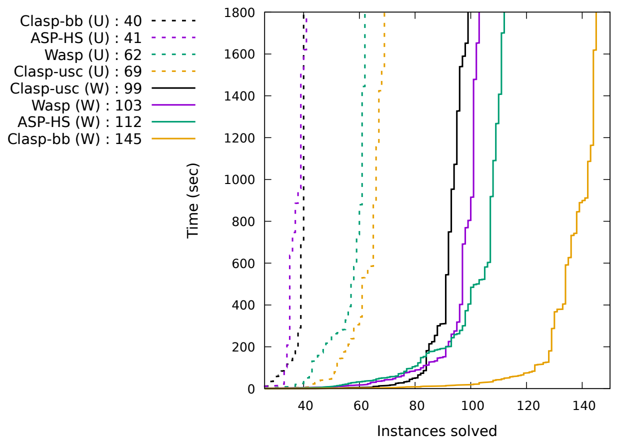

Figure 3: Comparison of the performance of ASP-HS with the core-guided approaches in Wasp and Clasp on weighted (W) and unweighted (U) instances.

performs best on TravelingSalesman. On the other hand, the branch-and-bound approach of Clasp is overall effective on weighted instances, and outperforms all core-guided approaches on Markov, FastFood, and SuperTree. Concerning unweighted instances, the performance of ASP-HS is on par with Clasp-bb, but does not reach the performance of the other core-guided approaches. This is somehow expected because for unweighted instances the number of iterations of ONE and OLL is bound by the number of soft atoms. On the contrary, we expected more positive results from weighted instances because ONE and OLL introduce several copies of the initial soft atoms, while in ASP-HS weights are handled by CPLEX. A more in-depth analysis of Bayes will help to understand the obtained results.

Impact of Weight Precision. For evaluating the impact of weight precision on solver performance, we focus on the benchmark Bayes, whose instances encode instances of Bayesian network structure learning, a central combinatorial optimization problem from the field of machine learning. The task is to find an optimal acyclic graph induced by choosing the set of parent nodes, each of different value, for each node in the graph. The weights of these instances arise from statistical tests, which naturally results in high-precision weights. The accuracy of the computed network structures depends on the precision of weights, but the ASP encoding for this problem divides all weights by $X = 1000$. From the perspective of the application domain, this is an

Table 2: Comparison of the performance of ASP-HS with Wasp and Clasp.

	ASP-HS		Wasp		Clasp-usc		Clasp-bb	
Family	#s	time	#s	time	#s	time	#s	time
Weighted								
Bayes1000 (20)	11	17793.4	15	11362.2	**16**	8943.7	15	9659.9
CausalGraph (20)	**18**	8891.7	13	14487.4	4	28800.7	**18**	5899.2
Fastfood (20)	9	20490.9	11	16554.4	12	17125.2	**20**	96.3
Markov (20)	1	35607.2	0	36000.0	0	36000.0	**15**	12289.0
MaxSAT (20)	12	17690.5	**19**	4174.7	15	11095.5	10	18258.3
OpenDoors (20)	**20**	297.5	**20**	214.85	**20**	137.9	**20**	158.1
SteinerTree (20)	0	36000.0	1	34242.1	1	34201.1	**3**	31035.8
SuperTree (20)	5	27358.7	5	27494.7	7	24536.8	**11**	21604.7
TravelingSalesman (20)	**10**	19510.3	3	31145.8	4	29505.1	4	29557.6
ValvesLocation (20)	15	10470.8	**16**	9431.4	9	23189.9	**16**	7866.9
VideoStreaming (20)	11	16201.9	0	36000.0	11	16200.3	**13**	12794.6
All Weighted	112	210313	103	221107	99	229736	**145**	149220
Unweighted								
ADF-WF (20)	**20**	2701.8	15	9847.4	**20**	182.1	**20**	73.2
CrossingMinimization (20)	**19**	2788.5	**19**	1933.2	**19**	1802.9	13	13686.2
MaximalClique (20)	0	36000.0	8	25002.1	**12**	18699.0	0	36000.0
StillLife (20)	2	35531.5	**19**	4439.2	12	15515.8	7	24313.5
SystemSynthesis (20)	0	36000.0	1	34936.9	**6**	29358.8	0	36000.0
All Unweighted	41	113022	62	76159	**69**	65559	40	110073

Table 3: Impact of weight precision on the performance of ASP-HS and Wasp on the Bayes benchmark family.

	ASP-HS		Wasp	
Family	solved	time	solved	time
Bayes1 (60)	**23**	73881.3	5	99335.6
Bayes10 (60)	**23**	73273.9	6	97480.6
Bayes100 (60)	**23**	72843.5	12	89454.1
Bayes1000 (60)	27	64406.4	**35**	51484.3
Total	**96**	284405	58	337755

unnatural choice, but gives us a natural way to evaluate how the precision of weights influence the performance of ASP solvers. Specifically, we varied $X \in \{1, 10, 100, 1000\}$, and refer to the corresponding benchmark by Bayes$X$. The smaller the X, the more precisely (better) the solutions to the ASP instances reflect the optimal solutions to the original problem instances. The comparison between ASP-HS and Wasp is shown in Table 3 and Figure 4. While the running time performance of Wasp noticeably deteriorates as weight precision is increased, with only 5 out of a total of 60 instances solved for $X = 1$; we observed similar behavior for Clasp-usc as well. In contrast, the performance of ASP-HS is mostly maintained when increasing precision.

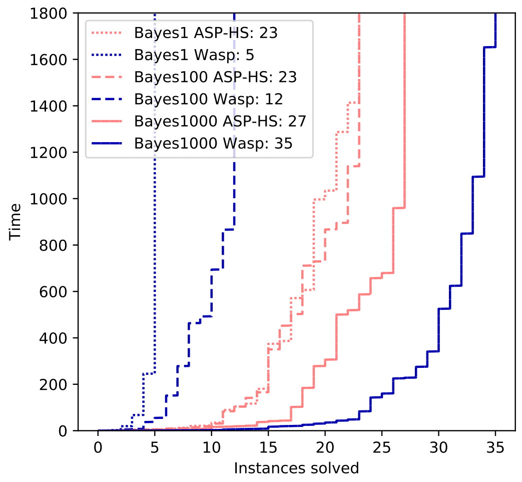

Figure 4: Comparison of ASP-HS with Wasp on the Bayes benchmark family.

5 Related Work

The first ASP solver applying unsatisfiable cores for optimization was Unclasp (Andres et al. 2012), based on Clasp and implementing the algorithm OLL. A main drawback of OLL is that it may add new constraints if soft literals introduced by unsatisfiable core analysis belong to subsequently detected unsatisfiable cores, and these new constraints only slightly differ from earlier extracted cores. This drawback was later circumvented in the MaxSAT solver MSCG (Morgado, Dodaro, and Marques-Silva 2014) using an encoding based on sorting networks. Subsequently, the ASP solver Wasp (Alviano et al. 2015) also implements OLL. The algorithms OLL and PMRES are also the origin of two other algorithms for MaxSAT, ONE and K (Alviano, Dodaro, and Ricca 2015), today implemented in Wasp and Clasp. Both of these solvers apply a technique for shrinking unsatisfiable cores (Alviano and Dodaro 2016; 2017).

The idea of implicit hitting set algorithms traces back to Reiter (1987) who developed a domain-specific approach to diagnosis using conflict sets (cores) and hitting sets. Later, implicit hitting set algorithms were proposed by Moreno-Centeno and Karp (2013) using notions of *separation oracles* (polynomial-time algorithms, in contrast to using

NP-decision procedures for core extraction, thus covering less general problems) and *circuits* (essentially unsatisfiable cores); specifically, they implemented the idea in the context of an NP-complete multigenome alignment problem.

More recently, IHS solvers have been proposed for MaxSAT (Davies and Bacchus 2011; 2013b; 2013a; Davies 2013; Saikko 2015; Saikko, Berg, and Järvisalo 2016), and their success is the main motivation of the present work. The approach has also been lifted to weighted CSPs (Delisle and Bacchus 2013) and most recently to satisfiability modulo theories (Fazekas, Bacchus, and Biere 2018), as well as to finding smallest-cardinality unsatisfiable subformulas (Ignatiev et al. 2015). More generally, the hitting set duality principle underlying core-guided algorithms has been successfully applied to various other NP-hard problems, such as enumerating of subset-minimal cores (Liffiton et al. 2016; Bailey and Stuckey 2005; Arif, Mencía, and Marques-Silva 2015), computing implicants and implicates of propositional formulas (Rymon 1994; Previti et al. 2015), quantified Boolean formula satisfiability (Janota and Marques-Silva 2015), model-based diagnosis (Stern et al. 2012), and computing Horn least upper bounds (Mencía, Previti, and Marques-Silva 2015). Moreover, Saikko, Wallner, and Järvisalo (2016) presented an extendable and generic framework for implicit hitting set algorithms, which is focused on applying one or several SAT solvers in addition to an IP solver for instantiating problems also in higher levels of the polynomial hierarchy, providing an implementation instantiating the framework for the second-level complete problem of propositional abduction. Following this trend, our work brings the IHS approach to answer set optimization. To the best of our knowledge, practical instantiations of the general IHS approach to optimization in ASP have not been developed earlier.

6 Conclusions

The ability to solve optimization problems, i.e., to find optimal solutions, considerably extends the range of applications of answer set programming as a declarative problem solving paradigm. We described a first approach of an answer set optimizer based on the implicit hitting set paradigm. This constitutes a hybrid approach, making use of both an ASP decision solver (as an unsatisfiable core extractor, covering both normal and disjunctive programs) and an IP solver (as a hitting set algorithm). The approach can benefit from advances in both ASP and IP solver technology. To this end, we described new combinations of interacting search techniques that are not directly available to ASP optimizers purely based on ASP decision solvers. We showed empirically that our first implementation of the approach is already promising as a complementary approach to optimization in ASP. Interesting directions for further work include a closer investigation into the interactions between different search techniques within the approach and adopting further IP techniques for further speeding up the approach in practice.

Acknowledgements

Work supported by Academy of Finland (grants 276412, 312662), Doctoral Programme in Computer Science of the University of Helsinki, POR CALABRIA FESR 2014-2020 project "DLV Large Scale" (CUP J28C17000220006), EU H2020 PON I&C 2014-2020 project "S2BDW" (CUP B28I17000250008), and GNCS-INdAM.

References

Abseher, M.; Gebser, M.; Musliu, N.; Schaub, T.; and Woltran, S. 2016. Shift design with answer set programming. *Fund. Inform.* 147(1):1–25.

Alviano, M., and Dodaro, C. 2016. Anytime answer set optimization via unsatisfiable core shrinking. *Theor. Pract. Log. Prog.* 16(5-6):533–551.

Alviano, M., and Dodaro, C. 2017. Unsatisfiable core shrinking for anytime answer set optimization. In *Proc. IJCAI*, 4781–4785. ijcai.org.

Alviano, M.; Dodaro, C.; Marques-Silva, J.; and Ricca, F. 2015. Optimum stable model search: algorithms and implementation. *J. Log. Comput.* in press.

Alviano, M.; Dodaro, C.; and Maratea, M. 2017. An advanced answer set programming encoding for nurse scheduling. In *Proc. AI*IA*, volume 10640 of *LNCS*, 468–482. Springer.

Alviano, M.; Dodaro, C.; and Ricca, F. 2015. A MaxSAT algorithm using cardinality constraints of bounded size. In *Proc. IJCAI*, 2677–2683. AAAI Press.

Andres, B.; Kaufmann, B.; Matheis, O.; and Schaub, T. 2012. Unsatisfiability-based optimization in clasp. In *ICLP Tech. Commun.*, volume 17 of *LIPIcs*, 211–221. Leibniz-Zentrum für Informatik.

Ansótegui, C.; Bonet, M. L.; and Levy, J. 2009. Solving (weighted) partial MaxSAT through satisfiability testing. In *Proc. SAT*, volume 5584 of *LNCS*, 427–440. Springer.

Arif, M. F.; Mencía, C.; and Marques-Silva, J. 2015. Efficient MUS enumeration of Horn formulae with applications to axiom pinpointing. In *Proc. SAT*, volume 9340 of *LNCS*, 324–342. Springer.

Bacchus, F.; Hyttinen, A.; Järvisalo, M.; and Saikko, P. 2017. Reduced cost fixing in MaxSAT. In *Proc. CP*, volume 10416 of *LNCS*, 641–651. Springer.

Bailey, J., and Stuckey, P. J. 2005. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In *Proc. PADL*, volume 3350 of *LNCS*, 174–186. Springer.

Bakker, R. R.; Dikker, F.; Tempelman, F.; and Wognum, P. M. 1993. Diagnosing and solving over-determined constraint satisfaction problems. In *Proc. IJCAI*, 276–281. Morgan Kaufmann.

Balduccini, M.; Gelfond, M.; Watson, R.; and Nogueira, M. 2001. The usa-advisor: A case study in answer set planning. In *Proc. LPNMR*, volume 2173 of *LNCS*, 439–442. Springer.

Baral, C., and Uyan, C. 2001. Declarative specification and solution of combinatorial auctions using logic programming. In *Proc. LPNMR*, volume 2173 of *LNCS*, 186–199. Springer.

Clark, K. L. 1977. Negation as failure. In *Symposium on Logic and Data Bases*, 293–322. Plemum Press.

Crowder, H.; Johnson, E. L.; and Padberg, M. 1983. Solving large-scale zero-one linear programming problems. *Oper. Res.* 31(5):803–834.

Danzig, G. B.; Fulkerson, D. R.; and Johnson, S. M. 1954. Solution of a large-scale traveling-salesman problem. *Oper. Res.* 2:393–410.

Davies, J., and Bacchus, F. 2011. Solving MAXSAT by solving a sequence of simpler SAT instances. In *Proc. CP*, volume 6876 of *LNCS*, 225–239. Springer.

Davies, J., and Bacchus, F. 2013a. Exploiting the power of MIP solvers in MaxSAT. In *Proc. SAT*, volume 7962 of *LNCS*, 166–181. Springer.

Davies, J., and Bacchus, F. 2013b. Postponing optimization to speed up MAXSAT solving. In *Proc. CP*, volume 8124 of *LNCS*, 247–262. Springer.

Davies, J. 2013. *Solving MAXSAT by Decoupling Optimization and Satisfaction*. Ph.D. Dissertation, University of Toronto. `http://www.cs.toronto.edu/~jdavies/Davies_Jessica_E_201311_PhD_thesis.pdf`.

Delisle, E., and Bacchus, F. 2013. Solving weighted CSPs by successive relaxations. In *Proc. CP*, volume 8124 of *LNCS*, 273–281. Springer.

Fazekas, K.; Bacchus, F.; and Biere, A. 2018. Implicit hitting set algorithms for maximum satisfiability modulo theories. In *Proc. IJCAR*, volume 10900 of *LNCS*, 134–151. Springer.

Fu, Z., and Malik, S. 2006. On Solving the Partial MAX-SAT Problem. In *Proc. SAT*, volume 4121 of *LNCS*, 252–265. Springer.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2012. Conflict-driven answer set solving: From theory to practice. *Artif. Intell.* 187:52–89.

Gebser, M.; Maratea, M.; and Ricca, F. 2015. The design of the sixth answer set programming competition. In *Proc. LPNMR*, volume 9345 of *LNCS*, 531–544. Springer.

Gebser, M.; Maratea, M.; and Ricca, F. 2017. The design of the seventh answer set programming competition. In *Proc. LPNMR*, volume 10377 of *LNCS*, 3–9. Springer.

Gebser, M.; Ryabokon, A.; and Schenner, G. 2015. Combining heuristics for configuration problems using answer set programming. In *Proc. LPNMR*, volume 9345 of *LNCS*, 384–397. Springer.

Gelfond, M., and Kahl, Y. 2014. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. Cambridge University Press.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Gen. Comput.* 9(3/4):365–386.

Hyttinen, A.; Eberhardt, F.; and Järvisalo, M. 2014. Constraint-based causal discovery: Conflict resolution with answer set programming. In *Proc. UAI*, 340–349. AUAI Press.

IBM. 2017. CPLEX Optimizer. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

Ignatiev, A.; Previti, A.; Liffiton, M. H.; and Marques-Silva, J. 2015. Smallest MUS extraction with minimal hitting set dualization. In *Proc. CP*, volume 9255 of *LNCS*, 173–182. Springer.

Janota, M., and Marques-Silva, J. 2015. Solving QBF by clause selection. In *Proc. IJCAI*, 325–331. AAAI Press.

Karp, R. M. 2010. Implicit hitting set problems and multi-genome alignment. In *Proc. CPM*, volume 6129 of *LNCS*, 151. Springer.

Koponen, L.; Oikarinen, E.; Janhunen, T.; and Säilä, L. 2015. Optimizing phylogenetic supertrees using answer set programming. *Theor. Pract. Log. Prog.* 15(4-5):604–619.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM T. Comput. Log.* 7(3):499–562.

Liffiton, M. H.; Previti, A.; Malik, A.; and Marques-Silva, J. 2016. Fast, flexible MUS enumeration. *Constraints* 21(2):223–250.

Lifschitz, V. 2002. Answer set programming and plan generation. *Artif. Intell.* 138:39–54.

Liu, G.; Janhunen, T.; and Niemelä, I. 2012. Answer set programming via mixed integer programming. In *Proc. KR*. AAAI Press.

Manquinho, V. M.; Silva, J. P. M.; and Planes, J. 2009. Algorithms for weighted Boolean optimization. In *Proc. SAT*, volume 5584 of *LNCS*, 495–508. Springer.

Marek, V. W., and Truszczyński, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm – A 25-Year Perspective*. Springer. 375–398.

Marileo, M. C., and Bertossi, L. E. 2010. The consistency extractor system: Answer set programs for consistent query answering in databases. *Data & Knowl. Eng.* 69(6):545–572.

Marques-Silva, J., and Manquinho, V. M. 2008. Towards more effective unsatisfiability-based maximum satisfiability algorithms. In *Proc. SAT*, volume 4996 of *LNCS*, 225–230. Springer.

Marques-Silva, J., and Planes, J. 2008. Algorithms for maximum satisfiability using unsatisfiable cores. In *Proc. DATE*, 408–413. IEEE.

Mencía, C.; Previti, A.; and Marques-Silva, J. 2015. SAT-based Horn least upper bounds. In *Proc. SAT*, volume 9340 of *LNCS*, 423–433. Springer.

Moreno-Centeno, E., and Karp, R. M. 2013. The implicit hitting set approach to solve combinatorial optimization problems with an application to multigenome alignment. *Oper. Res.* 61(2):453–468.

Morgado, A.; Dodaro, C.; and Marques-Silva, J. 2014. Core-guided MaxSAT with soft cardinality constraints. In *Proc. CP*, volume 8656 of *LNCS*, 564–573. Springer.

Nemhauser, G. L., and Wolsey, L. A. 1999. *Integer and Combinatorial Optimization*. Wiley-Interscience.

Niemelä, I. 1999. Logic programming with stable model semantics as constraint programming paradigm. *Ann. Math. Artif. Intell.* 25(3–4):241–273.

Previti, A.; Ignatiev, A.; Morgado, A.; and Marques-Silva, J. 2015. Prime compilation of non-clausal formulae. In *Proc. IJCAI*, 1980–1988. AAAI Press.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artif. Intell.* 32(1):57–95.

Ricca, F.; Grasso, G.; Alviano, M.; Manna, M.; Lio, V.; Iiritano, S.; and Leone, N. 2012. Team-building with answer set programming in the Gioia-Tauro seaport. *Theor. Pract. Log. Prog.* 12(3):361–381.

Rymon, R. 1994. An SE-tree-based prime implicant generation algorithm. *Ann. Math. Artif. Intell.* 11(1-4):351–366.

Saikko, P.; Berg, J.; and Järvisalo, M. 2016. LMHS: A SAT-IP hybrid MaxSAT solver. In *Proc. SAT*, volume 9710 of *LNCS*, 539–546. Springer.

Saikko, P.; Wallner, J. P.; and Järvisalo, M. 2016. Implicit hitting set algorithms for reasoning beyond NP. In *Proc. KR*, 104–113. AAAI Press.

Saikko, P. 2015. Re-implementing and extending a hybrid SAT-IP approach to maximum satisfiability. Master's thesis, University of Helsinki. `http://hdl.handle.net/10138/159186`.

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artif. Intell.* 138(1-2):181–234.

Stern, R. T.; Kalech, M.; Feldman, A.; and Provan, G. M. 2012. Exploring the duality in conflict-directed model-based diagnosis. In *Proc. AAAI*. AAAI Press.