Optimality-preserving Logic-Based Benders Decomposition of Answer Set Programs

Carmine Dodaro[1] , **Antonio Ielo**[2] , **Marco Maratea**[1] , **Cinzia Marte**[1] , **Alice Tarzariol**[2]
[1]University of Calabria, Italy
[2]University of Klagenfurt, Austria
carmine.dodaro@unical.it, antonio.ielo@unical.it, marco.maratea@unical.it, cinzia.marte@unical.it, alice.tarzariol@aau.at

Abstract

Bender Decomposition is a well-known solving technique in Operation Research that decomposes a problem into a master and a subproblem, which interact via "cuts". This technique has been extended to Logic-Based Bender Decomposition (LBBD), solving problems specified by logic-based languages and enabling a wider applicability. However, while Bender Decomposition guarantees optimality, LBBD does not: this property is problem-specific and depends on the defined decomposition and cuts.

In this paper, we present a theoretical analysis of the conditions under which LBBD, including cuts, can preserve optimality in the context of Answer Set Programming (ASP), a prominent logic-based language in the field of Artificial Intelligence. We also introduce a general-purpose algorithm that employs both minimal unsatisfiable subsets and minimal correction subsets to define cuts in a fully automated, problem-independent way. The algorithm preserves the optimality guarantees of Bender Decomposition. An empirical evaluation on real-world scheduling instances shows that our approach can find significantly more solutions, and more optimal ones, compared to a standard direct ASP encoding, while also consistently reducing the execution time.

1 Introduction

Bender Decomposition [Geoffrion, 1972; Hooker and Ottosson, 2003; Rahmaniani *et al.*, 2017] is a well-known solving technique in Operation Research that works by decomposing a problem at hand. Logic-Based Benders Decomposition (LBBD) [Hooker and Ottosson, 2003] is an optimisation technique that extends the classical Bender Decomposition approach, offering greater flexibility and applicability to a wider range of complex problems. The two approaches work by decomposing a given problem into a master and a subproblem. The former typically focuses on high-level decisions, whereas the latter deals with more detailed and often logically complex constraints. They interact via *cuts*, i.e., constraints based on an analysis of subproblem solutions (or lack of) that remove part of the search space of the master. Classical Bender Decomposition and LBBD share many similarities but also have important differences [Hooker, 2024]: the most significant is that the Bender Decomposition approach guarantees optimality, while LBBD does not, unless certain conditions are met. The hurdles with applying LBBD are identifying a proper decomposition into master and subproblem, and/or the fact that the generated cuts do not preserve the solution of the original problem. Regarding cuts, in the literature [Hooker, 2024] several types are listed, defined according to their scope and the characteristics of the master and subproblem, where a problem-specific analysis is needed in order to identify the cuts. In the literature, there is no uniform way to define cuts that preserve optimality or feasibility *in a problem-independent way*, as well as handling multiple subproblems. Among the various logic-based programming paradigms, we focus on Answer Set Programming (ASP) [Brewka *et al.*, 2011], which is a declarative programming paradigm based on non-monotonic reasoning and the stable model semantics [Gelfond and Lifschitz, 1991]. Thanks to its characteristics and efficient solvers, ASP has been successfully adopted in numerous domains [Erdem *et al.*, 2016; Falkner *et al.*, 2018]. Recently, promising studies employing LBBD for ASP solving have emerged (see e.g., [Cappanera *et al.*, 2023; Dodaro *et al.*, 2026]), where the authors identified cuts originating from problem-specific analysis. In this paper, we provide the precise conditions to split an ASP program modelling a target problem, while preserving its solutions according to the LBBD approach. We identify cuts relying on the ASP notions of minimal unsatisfiable subsets (MUSes) and correction sets (MCSes) [Alviano *et al.*, 2023], originating from the infeasibility of subproblems, that preserve satisfiability/optimality of the solutions.

Further, we also introduce and formally prove the correctness of a general-purpose algorithm employing both MUSes and MCSes to define cuts in a fully automated, problem-independent way, while guaranteeing problem solutions' optimality. We also formalise the conditions to apply cuts in our approach and algorithm, extending them to efficiently handle more subproblems. Lastly, we implemented our algorithm and evaluated it on healthcare scheduling instances where an LBBD approach had previously been applied with ASP [Dodaro *et al.*, 2026; Caruso *et al.*, 2025]. We compared it to a

direct encoding and a domain-specific, expert-crafted decomposition. Results show that our method outperforms the direct encoding in standard metrics, i.e., the number of (optimal) solutions found. Despite the domain expert-tuning in [Dodaro *et al.*, 2026], our problem-independent approach finds more optimal solutions since it is correct by design, and remains competitive in terms of the number of solutions found.

2 Preliminaries

This section begins with the notions of Answer Set Programming, including its syntax, semantics, and the necessary concepts for the results developed in the paper. Then, it provides a brief overview of Logic-Based Benders Decomposition.

2.1 Answer Set Programming (ASP)

Syntax and Semantics. Let $\mathcal{A}$ be a set of atoms. An *atomic formula* is either an atom or the connective $\bot$. A *literal* is an atomic formula possibly preceded by the default negation symbol $\sim$. Let $\top$ be a compact representation of $\sim \bot$. A *rule* r has the following form:

$$r:\ p \leftarrow q_1, \ldots, q_k, \sim q_{k+1}, \ldots, \sim q_n \tag{1}$$

where $n \geq 0$, and $p, q_1, \ldots, q_n$ are atomic formulas. Let $head(r)$ denote the set $\{p\} \cap \mathcal{A}$, and let respectively $body^+(r)$ and $body^-(r)$ denote the set $\{q_1, \ldots, q_k\}$ and $\{q_{k+1}, \ldots, q_n\}$ of positive and negative body literals. We say that r is a *fact* if $body^+(r) = body^-(r) = \emptyset$, while it is a *constraint* if $head(r) = \emptyset$. For a rule r of the form (1), we define the set of atoms appearing in it as $atoms(r) = head(r) \cup body^+(r) \cup body^-(r)$. An ASP *program* Π is a finite set of rules of the form (1); accordingly, $atoms(\Pi) = \{atoms(r) \mid r \in \Pi\}$.

The semantics of Π is defined in terms of its equivalent *propositional instantiation* $G(\Pi)$. To compute it, we replace each rule $r \in \Pi$ with all its possible instantiations obtained by substituting the variables in r with constants occurring in Π. An *interpretation* $\mathcal{I}$ is a set of (*true*) propositional atoms occurring in $G(\Pi)$ that does not contain $\bot$. We say that $\mathcal{I}$ *satisfies* a normal rule $r \in G(\Pi)$, if $body^+(r) \subseteq \mathcal{I}$ and $body^-(r) \cap \mathcal{I} = \emptyset$ imply $head(r) \subseteq \mathcal{I}$. If $\mathcal{I}$ satisfies all rules $r \in G(\Pi)$, it is a *model* of Π and it is *stable* if it is a subset-minimal model of the *reduct*, which is defined as $\{head(r) \leftarrow body^+(r) \mid r \in \Pi, body^-(r) \cap \mathcal{I} = \emptyset\}$. The set of all stable models of Π, also called *answer sets*, is denoted by $Ans(\Pi)$. We say that a program Π is *incoherent* if $Ans(\Pi) = \emptyset$, else Π is *coherent*.

We can extend an ASP program with weak constraints, defining an *optimisation program*. A *weak constraint* has the form: $\leftsquigarrow q_1, \ldots, q_k, \sim q_{k+1}, \ldots, \sim q_n\ [w@p, t]$, where q_i, for $1 \leq i \leq n$, are atoms, w and p are integers representing the *weight* and *priority level* of the weak constraint, and t is a tuple of terms. Weak constraints in Π do not change $Ans(\Pi)$, but induce a preference among them. If $\{q_1, \ldots, q_m\} \subseteq \mathcal{I}$ and $\{q_{m+1}, \ldots, q_n\} \cap \mathcal{I} = \emptyset$ for a (ground) weak constraint in $G(\Pi)$, the vector t of terms is associated with the weight w at priority level p. Summing the weights w of term tuples t at each priority level p yields a vector of integers, where an *optimal* answer set incurs the smallest sum of weights at the highest priority level, then at the second highest priority level in case of a tie, and so on. We refer to *cost* of an answer set as the vector of integers obtained from the weak constraints and use $\prec$ to define the order among them. Refer to [Calimeri *et al.*, 2020] for more details on ASP syntax and semantics.

As syntactic sugar, we can extend an ASP program Π with *free choice rules*, denoted as $\{p\} \leftarrow \top$ for an atom p, which can be seen as the rules $p \leftarrow\!\sim np$ and $np \leftarrow\!\sim p$ with np being a *fresh* atom not appearing in Π. For each atom $a \in atoms(\Pi)$, we denote by $choice(a)$ a free choice rule for a, by $fix(a)$ a constraint ρ with $body^+(\rho) = \emptyset$ and $body^-(\rho) = a$, and $fact(a)$ a rule with a in the head and empty body. With a slight abuse of notation, we extend $choice$ to a set of atoms A as follows: $choice(A) = \{choice(a) : a \in A\}$. Similarly, we extend also the functions fix and $fact$.

Example 1 (Running example)**.** *Let Π be the ASP program:*

$r_1 : \{a\}.$ $\quad r_5 : x \leftarrow\!\sim y, a.$ $\quad r_8 : \bot \leftarrow x, z.$
$r_2 : \{b\}.$ $\quad r_6 : y \leftarrow\!\sim x, a.$ $\quad r_9 : \bot \leftarrow y, z.$
$r_3 : \leftsquigarrow a\ [-2@1]$ $\quad r_7 : z \leftarrow b.$ $\quad r_{10} : \bot \leftarrow\!\sim a, \sim b.$
$r_4 : \leftsquigarrow b\ [-1@1]$

$Ans(\Pi) = \big\{\{a, x\}, \{a, y\}, \{b, z\}\big\}$, among which the optimal answer sets are $\{a, x\}$ and $\{a, y\}$.

Minimal Unsatisfiable and Correction Subset. Here we report concepts presented in [Alviano *et al.*, 2023]. We define $enforce(\Pi, O, U) = \Pi \cup choice(O) \cup fix(U)$.

Definition 1 (Minimal Unsatisfiable Subset - MUS)**.** *Given a program Π and a set of objective atoms $O \subseteq atoms(\Pi)$ such that $head(\Pi) \cap O = \emptyset$, we say that a set $U \subseteq O$ is an unsatisfiable subset for Π wrt O if $enforce(\Pi, O, U)$ is incoherent. Moreover, U is minimal (MUS) if none of its proper subsets is an unsatisfiable subset for Π wrt O.*

Definition 2 (Minimal Correction Subset - MCS)**.** *Given a program Π and a set of objective atoms $O \subseteq atoms(\Pi)$ such that $head(\Pi) \cap O = \emptyset$, we say that a set $M \subseteq O$ is a correction subset for Π wrt O if $enforce(\Pi, O, O \setminus M)$ is coherent. Moreover, M is minimal (MCS) if none of its proper subsets is a correction subset for Π wrt O.*

Example 2. *Consider the set of rules of Example 1. Let $\Pi' = \{r_5, .., r_9\}$. The only MUS for Π' wrt $\{a, b\}$ is the set $\{a, b\}$, while the MCSes are $m_1 = \{a\}$ and $m_2 = \{b\}$.*

Theorem 1. ***[Alviano* et al., *2023]*** *Let Π be a program, and $O \subseteq atoms(\Pi)$ be a set of objective atoms. For all sets U, U', if U is an unsatisfiable subset for Π wrt O and $U \subseteq U' \subseteq O$, then U' is an unsatisfiable subset for Π wrt O.*

Splitting Sets. Similarly to the previous paragraph, here we recap notions presented in [Lifschitz and Turner, 1994].

Definition 3 (Splitting Set)**.** *A set S of atoms is a splitting set of a logic program Π if for each $r \in \Pi$, $head(r) \cap S \neq \emptyset$ implies that $atoms(r) \subseteq S$.*

A splitting set S partitions Π into two sets of rules, $b_S(\Pi)$ and $t_S(\Pi)$, namely the *bottom* and *top* of Π wrt S, where $b_S(\Pi) = \{r \mid r \in \Pi, head(r) \cap S \neq \emptyset\}$ and $t_S(\Pi) = \Pi \setminus b_S(\Pi)$. We highlight that splitting sets can be extended to optimisation programs, as weak constraints do not change the ASP semantics but only define an order over answer sets.

Example 3. *Consider the set of rules of Example 1. The set* $S = \{a, b\}$ *is a splitting set for* Π*, where* $b_S(\Pi) = \{r_1, \ldots, r_4\} \cup \{r_{10}\}$ *and* $t_S(\Pi) = \{r_5, \ldots, r_9\}$.

Given a rule r and a set of atoms S, we define $r' = eval(r, S)$ such that $head(r') = head(r)$, $body^+(r') = body^+(r) \setminus S$, and $body^-(r') = body^-(r) \setminus S$. Given a program Π, a splitting set S, and a set of atoms X, we define the (partial) evaluation of $t_S(\Pi)$ according to X as $e_S(\Pi, X) = \{eval(r, S) \mid r \in t_S(\Pi), body^+(r) \cap S \subseteq X, (body^-(r) \cap S) \cap X = \emptyset\}$.

Definition 4 (Solution to Π wrt Splitting Set S)**.** *Let* S *be a splitting set for* Π*. A pair* (X, Y) *is a solution for* Π *wrt* S *if* $X \in Ans(b_S(\Pi))$*,* $Y \in Ans(e_S(\Pi, X))$*, and* $X \cup Y$ *is consistent, namely,* $(atoms(b_S(\Pi)) \setminus X) \cap Y = \emptyset$ *and* $(atoms(e_S(\Pi, X)) \setminus Y) \cap X = \emptyset$.

Theorem 2. ***[Lifschitz and Turner, 1994]*** *Let* S *be a splitting set of a program* Π*. A set* S *is an answer set of* Π *iff* $S = X \cup Y$ *for some solution* (X, Y) *of* Π *wrt* S.

2.2 Logic-Based Benders Decomposition (LBBD)

The LBBD approach consists of decomposing a problem into a master and a subproblem [Hooker, 2024]. The core idea is to iteratively optimise the overall solution by generating *cuts* derived from the subproblem solution, i.e., constraints restricting the search space of the master problem, without affecting its optimal solutions. In our work, we focus on decision subproblems, which consequently generate *feasibility cuts*, i.e., a cut derived from an infeasible subproblem that excludes one or more infeasible assignments to the master problem variables. Consider a problem P:

$$\min_{x,y,w} \{f(x) \mid C(x, y),\, C'(x, w),\, x \in \mathcal{D}_x,\, y \in \mathcal{D}_y,\, w \in \mathcal{D}_w\} \tag{2}$$

where x, y and w are variables respectively defined over the domains $\mathcal{D}_x, \mathcal{D}_y$ and $\mathcal{D}_w$. The constraint set $C(x, y)$ includes variables from y and potentially from x, while the constraints in $C'(x, w)$ involve variables from x and/or w. From the problem P, we can identify the *master problem* as the optimisation problem aiming to minimise a function $f(x)$ and containing only the constraints $C'(x, w)$ and variables x and w, while the *subproblem* considers the constraints $C(x, y)$ and defines only the variables y. Notice that the auxiliary variables w are not part of a standard Benders decomposition, but are used to characterise variables in the master problem that have no influence on the subproblem's formulation. Then, we can follow an iterative procedure that first searches for an optimal solution of the master problem:

$$\min_{x,w} \{f(x) \mid C'(x, w), x \in \mathcal{D}_x,\, w \in \mathcal{D}_w\} \tag{3}$$

and uses the optimal value found for x, namely $\bar{x}$, to define the *subproblem* $P_{\mathcal{S}}(\bar{x})$:

$$\{y \mid C(\bar{x}, y),\, y \in \mathcal{D}_y\}\,. \tag{4}$$

If $P_{\mathcal{S}}(\bar{x})$ is infeasible, a feasibility cut $F_{\bar{x}}$ is generated and included in the master problem. The LBBD approach continues until, at a generic iteration k, either the master problem obtained by extending Equation (3) with $\{F_{\bar{x}^\ell} \mid \ell \in [1..k-1]\}$ becomes inconsistent (which means that no solution for the original problem exists) or it returns the values $(\bar{w}^k, \bar{x}^k)$ and the subproblem $P_{\mathcal{S}}(\bar{x}^k)$ has a solution $\bar{y}$; in this case, the solution of the original problem P is equal to $(\bar{w}^k, \bar{x}^k, \bar{y})$.

3 LBBD for ASP

In this section, we define the LBBD approach applied to the ASP formalism. Given a problem P defined as in Equation (2), we model it with the logic program Π where $Ans(\Pi)$ corresponds to solutions of P. It is well known that a problem can be modelled with different ASP programs, as long as their answer sets coincide with the problem's solutions. From now on, we consider a program Π such that $atoms(\Pi)$ can be partitioned into three sets, representing the elements in D_w, D_x, and D_y, respectively. In the following, we formalise the concept of LBBD for ASP.

Definition 5 (LBBD Program)**.** *Given an (optimisation) program* $\Pi_{\mathcal{M}}$ *and a program* $\Pi_{\mathcal{S}}$ *such that* $\Pi_{\mathcal{M}} \cap \Pi_{\mathcal{S}} = \emptyset$*, we define an LBBD program* $(\Pi_{\mathcal{M}}, \Pi_{\mathcal{S}})$ *if for all* $r \in \Pi_{\mathcal{S}}$*,* $head(r) \cap O = \emptyset$*, where* $O = atoms(\Pi_{\mathcal{M}}) \cap atoms(\Pi_{\mathcal{S}})$.

Example 4. *Consider the Example 1. An LBBD program is* $(\Pi_{\mathcal{M}}, \Pi_{\mathcal{S}})$*, where* $\Pi_{\mathcal{M}} = \{r_1, \ldots, r_4\} \cup \{r_{10}\}$ *and* $\Pi_{\mathcal{S}} = \{r_5, \ldots, r_9\}$*, with* $atoms(\Pi_{\mathcal{M}}) \cap atoms(\Pi_{\mathcal{S}}) = \{a, b\}$.

In other words, an LBBD program identifies a partition of rules of Π into a *master program* $\Pi_{\mathcal{M}}$, reflecting the optimisation problem in Equation (3), and a *subprogram* $\Pi_{\mathcal{S}}$, reflecting the problem in Equation (4). Since the variable vector x appears both in the master and in the subproblem, we represent it with the set of atoms O and impose that $\Pi_{\mathcal{S}}$ does not define them. We can consider the variable vector w as the remaining atoms of the master program, i.e. $atoms(\Pi_{\mathcal{M}}) \setminus O$, while y is defined through $atoms(\Pi_{\mathcal{S}}) \setminus O$.

Definition 6 (LBBD Solution)**.** *Given an LBBD program* $(\Pi_{\mathcal{M}}, \Pi_{\mathcal{S}})$*, we define its solutions, denoted as* $Sol(\Pi_{\mathcal{M}}, \Pi_{\mathcal{S}})$*, as the set of pairs* (X, Y) *where* $Y \in Ans(\Pi_{\mathcal{S}} \cup fact(X))$ *and* $X \in Ans(\Pi_{\mathcal{M}})$*. The projection wrt* X *is denoted as* $Sol_{\Pi_{\mathcal{M}}}(\Pi_{\mathcal{M}}, \Pi_{\mathcal{S}})$.

To guarantee that every LBBD solution corresponds to an answer set of Π, we first prove that any LBBD program $(\Pi_{\mathcal{M}}, \Pi_{\mathcal{S}})$ can be obtained from a splitting set S, and vice versa. To be consistent with our definition, we assume that S is such that all the weak constraints in $\Pi_{\mathcal{M}} \cup \Pi_{\mathcal{S}}$ are collected in $b_S(\Pi_{\mathcal{M}} \cup \Pi_{\mathcal{S}})$.

Lemma 1. *Let* Π *be an (optimisation) program. Then,* S *is a splitting set for* Π *iff* $(\Pi_{\mathcal{M}}, \Pi_{\mathcal{S}})$ *is an LBBD program, where* $\Pi_{\mathcal{M}} = b_S(\Pi)$*,* $\Pi_{\mathcal{S}} = t_S(\Pi)$.

Proof. ($\Leftarrow$) Let $(\Pi_{\mathcal{M}}, \Pi_{\mathcal{S}})$ be an LBBD program, $\Pi = \Pi_{\mathcal{M}} \cup \Pi_{\mathcal{S}}$. Consider the set $atoms(\Pi_{\mathcal{M}})$. We observe that the only rules in Π such that $head(r) \cap atoms(\Pi_{\mathcal{M}}) \neq \emptyset$ are the ones in $\Pi_{\mathcal{M}}$, since the atoms of $\Pi_{\mathcal{M}}$ that are shared with $\Pi_{\mathcal{S}}$ can appear only in the body of the subprogram by definition of LBBD program. Hence, $atoms(\Pi_{\mathcal{M}}) = S$ is a splitting set for Π and $\Pi_{\mathcal{M}} = b_S(\Pi)$ and $\Pi_{\mathcal{S}} = t_S(\Pi)$.

($\Rightarrow$) Let S be a splitting set. Then, $b_S(\Pi)$ and $t_S(\Pi)$ are disjoint and $t_S(\Pi) = \{r \mid r \in \Pi, head(r) \cap S = \emptyset\}$. Let $O = atoms(b_S(\Pi)) \cap atoms(t_S(\Pi))$. Hence,

$O \subseteq atoms(b_S(\Pi))$, while $atoms(b_S(\Pi)) \subseteq S$ from the definition of splitting set; thus $O \subseteq S$. Therefore, for all $r \in t_S(\Pi)$ it follows that $head(r) \cap O = \emptyset$, which proves that $(b_S(\Pi), t_S(\Pi))$ is an LBBD program. □

Corollary 1. *Let S be a splitting set for the program Π. Then, $\{X \cup Y \mid Y \in Ans(e_S(\Pi, X))\} = Ans(t_S(\Pi) \cup fact(X))$, where $X \in Ans(b_S(\Pi))$.*

Proof. We observe that, by the definition of $t_S(\Pi)$, the atoms in S can only be justified in $b_S(\Pi)$. Given $X \in Ans(b_S(\Pi))$, $e_S(\Pi, X)$ contains the rules of $t_S(\Pi)$ such that the atoms in X are true and the atoms in $S \setminus X$ are false, simplifying the literals of S. Thus, $Ans(t_S(\Pi) \cup fact(X)) = \{X \cup Y \mid Y \in Ans(e_S(\Pi, X))\}$. □

Theorem 3. *Let $(\Pi_\mathcal{M}, \Pi_\mathcal{S})$ be an LBBD program. Then, $(X, Y) \in Sol(\Pi_\mathcal{M}, \Pi_\mathcal{S})$ iff $Y \in Ans(\Pi_\mathcal{M} \cup \Pi_\mathcal{S})$.*

Proof. Let $I = (\Pi_\mathcal{M}, \Pi_\mathcal{S})$ be an LBBD program and $\Pi = \Pi_\mathcal{M} \cup \Pi_\mathcal{S}$. From Lemma 1, there exists a splitting set S for Π such that $I = (b_S(\Pi), t_S(\Pi))$. By Corollary 1, we have that $Ans(t_S(\Pi) \cup fact(X))$ corresponds to $X \cup Y$ for all (X, Y) that are solutions of Π wrt S. Thus, for Theorem 2, we have that $Ans(t_S(\Pi) \cup fact(X)) = Ans(\Pi)$. Then, the thesis follows from the definition of an LBBD solution. □

Lastly, we extend the concept of an optimal solution for an LBBD program. Thanks to Theorem 3 and the assumption that the subprogram is not an optimisation one, we get that it coincides with the optimal answer sets of Π.

Definition 7 (LBBD Optimal Solution)**.** *Given an LBBD program I, a solution $(X, Y) \in Sol(I)$ is optimal if there is no $(X', Y') \in Sol(I)$ such that $c(X', Y') \prec c(X, Y)$, where c returns the cost associated to the master program solution.*

In the following, we consider only non-trivial splitting sets, i.e., inducing a bottom and top program having atoms in common. This allows us to obtain only "meaningful" LBBD instances where the variable vector x is not empty.

3.1 LBBD Cuts via MUSes and MCSes

Here, we formalise LBBD cuts within the ASP context, which can be seen as a set of constraints reducing the number of answer sets of the master program.

Definition 8 (LBBD (Safe) Cut)**.** *A cut for an LBBD program $(\Pi_\mathcal{M}, \Pi_\mathcal{S})$ is a set of constraints C, where $Ans(\Pi_\mathcal{M} \cup C) \subset Ans(\Pi_\mathcal{M})$. Moreover, C is safe if $Sol_{\Pi_\mathcal{M}}(\Pi_\mathcal{M}, \Pi_\mathcal{S}) \subseteq Ans(\Pi_\mathcal{M} \cup C)$.*

Next, we exploit MUSes and MCSes (see Definition 1 and 2) to define specific types of cuts for an LBBD program.

Definition 9 (MUS-Cut / MCS-Cut)**.** *Let $(\Pi_\mathcal{M}, \Pi_\mathcal{S})$ be an LBBD program and $O \subseteq atoms(\Pi_\mathcal{M}) \cap atoms(\Pi_\mathcal{S})$.*

- *Given an MUS U for $\Pi_\mathcal{S}$ wrt O, an MUS-Cut is a singleton $\{\rho\}$ such that ρ is an LBBD cut with $body^+(\rho) = U$ and $body^-(\rho) = \emptyset$.*
- *Given an MCS M for $\Pi_\mathcal{S}$ wrt O, an MCS-Cut is the set of LBBD cuts $C = \{\bot \leftarrow m \mid m \in M\}$.*

We now prove that unsatisfiable subsets identify LBBD safe cuts and, to this aim, we first provide the following results.

Lemma 2. *Let U be an unsatisfiable subset for Π wrt O, then $fact(U) \cup \Pi$ is incoherent.*

Proof. By contradiction, $fact(U) \cup \Pi$ is coherent. Let $A \in Ans(fact(U) \cup \Pi)$. By the ASP semantics, $fact(U)$ assigns all atoms in U to true, thus $U \subseteq A$. Consequently, A satisfies all the constraints in $fix(U)$. Hence, $A \in Ans(enforce(\Pi, O, U))$ as choice rules in $choice(O)$ are always satisfied, contradicting the assumption that U is an unsatisfiable subset, i.e., $enforce(\Pi, O, U)$ is incoherent. □

Theorem 4. *Let $(\Pi_\mathcal{M}, \Pi_\mathcal{S})$ be an LBBD program and $O \subseteq atoms(\Pi_\mathcal{M}) \cap atoms(\Pi_\mathcal{S})$. Every MUS-Cut defined from an MUS for $\Pi_\mathcal{S}$ wrt O is a safe cut.*

Proof. Let U be an MUS for $\Pi_\mathcal{S}$ wrt O. We assume by contradiction that there exists an MUS-Cut $\{\rho\}$ with $body^+(\rho) = U$ that is not safe. Then, there exists a solution $A \in Sol_{\Pi_\mathcal{M}}(\Pi_\mathcal{M}, \Pi_\mathcal{S})$ such that $A \notin Ans(\Pi_\mathcal{M} \cup \{\rho\})$. Since $A \in Ans(\Pi_\mathcal{M})$ but $A \notin Ans(\Pi_\mathcal{M} \cup \{\rho\})$, it must be that A satisfies the body of ρ, thus $U \subseteq A$. Since U is an unsatisfiable subset of $\Pi_\mathcal{S}$ wrt O, we have that $fact(U) \cup \Pi_\mathcal{S}$ is incoherent by Lemma 2. However, by the definition of LBBD solution, we have that $fact(A) \cup \Pi_\mathcal{S}$ is coherent. Since $U \subseteq A$, this contradicts the monotonicity of unsatisfiable subsets (see Theorem 1). □

Note that only LBBD safe cuts correspond to feasibility cuts. While MUS-Cuts are safe, MCS-Cuts are not, despite being more effective in the pruning of the solution space (since their body always has a single literal).

Example 5 (Running example)**.** *Consider the Examples 2 and 4, we can see that $\Pi_\mathcal{S} = \Pi'$, $\Pi_\mathcal{M} = \Pi \setminus \Pi'$, and $\{a, b\} \in Ans(\Pi_\mathcal{M})$. Then, the MCSes m_1 and m_2 produce the MCS-Cuts $C_1 = \{\bot \leftarrow a\}$ and $C_2 = \{\bot \leftarrow b\}$, respectively. Both cuts are not safe since $Sol_{\Pi_\mathcal{M}}(\Pi_\mathcal{M}, \Pi_\mathcal{S}) = \{\{a\}, \{b\}\}$, but C_1 removes $\{a\}$, while C_2 removes $\{b\}$.*

3.2 Handling Multiple Subprograms

Inspired by the methodology introduced in [Cappanera *et al.*, 2023], we generalise the concept of LBBD program to include multiple independent subprograms, and formalise the conditions under which such a family of subprograms can be considered valid within our model. Given an LBBD program $(\Pi_\mathcal{M}, \Pi_\mathcal{S})$ we can represent it as $(\Pi_\mathcal{M}, \widetilde{\Pi}_\mathcal{S})$, where $\widetilde{\Pi}_\mathcal{S}$ is the partition $\{\Pi_{\mathcal{S}_1}, \ldots, \Pi_{\mathcal{S}_n}\}$ of $\Pi_\mathcal{S}$ for $n \geq 1$, and for each $\Pi_{\mathcal{S}_i}, \Pi_{\mathcal{S}_j} \in \widetilde{\Pi}_\mathcal{S}$, with $i \neq j$, $atoms(\Pi_{\mathcal{S}_i}) \cap atoms(\Pi_{\mathcal{S}_j}) \subseteq atoms(\Pi_\mathcal{M})$. Accordingly, the LBBD solutions $Sol(\Pi_\mathcal{M}, \widetilde{\Pi}_\mathcal{S})$ is the set of pairs $(X, \widetilde{Y})$, where $X \in Ans(\Pi_\mathcal{M})$ and $\widetilde{Y} = Y_1 \cup \cdots \cup Y_n$, with $Y_i \in Ans(\Pi_{\mathcal{S}_i} \cup fact(X))$. Thanks to the subprograms partition, we have that $Sol(\Pi_\mathcal{M}, \Pi_\mathcal{S}) = Sol(\Pi_\mathcal{M}, \widetilde{\Pi}_\mathcal{S})$. Moreover, note that each $(\Pi_\mathcal{M}, \Pi_{\mathcal{S}_i})$ is also an LBBD program.

Lemma 3. *Let $(\Pi_\mathcal{M}, \widetilde{\Pi}_\mathcal{S})$ be an LBBD program, with $\widetilde{\Pi}_\mathcal{S} = \{\Pi_{\mathcal{S}_1}, \ldots, \Pi_{\mathcal{S}_n}\}$. For any $i \in \{1, \ldots, n\}$, an MUS-Cut $\{\rho\}$ for $(\Pi_\mathcal{M}, \Pi_{\mathcal{S}_i})$ is safe for $(\Pi_\mathcal{M}, \widetilde{\Pi}_\mathcal{S})$.*

Proof. By contradiction, there exists $A \in Sol_{\Pi_{\mathcal{M}}}(\Pi_{\mathcal{M}}, \widetilde{\Pi}_{\mathcal{S}})$ such that $A \notin Ans(\Pi_{\mathcal{M}} \cup \{\rho\})$. For the definition of LBBD solution with multiple subprograms, A is such that for each $i \in \{1, .., n\}$ there exists a $Y_i \in Ans(\Pi_{\mathcal{S}_i} \cup fact(A))$. This means that $(A, Y_i) \in Sol(\Pi_{\mathcal{M}}, \Pi_{\mathcal{S}_i})$. For Theorem 4, $\{\rho\}$ is safe for $(\Pi_{\mathcal{M}}, \Pi_{\mathcal{S}_i})$, thus $Sol_{\Pi_{\mathcal{M}}}(\Pi_{\mathcal{M}}, \Pi_{\mathcal{S}_i}) \subseteq Ans(\Pi_{\mathcal{M}} \cup \{\rho\})$, which contraddicts our assumption, i.e., $A \in Ans(\Pi_{\mathcal{M}} \cup \{\rho\})$. □

Algorithm 1 Main Algorithm

Input: An LBBD program $(\Pi_{\mathcal{M}}, \Pi_{\mathcal{S}})$
Output: An optimal solution $Y \in Ans(\Pi_{\mathcal{M}} \cup \Pi_{\mathcal{S}})$

```
1:  last, ub ← ⊥, +∞
2:  Safe, Unsafe ← ∅, ∅
3:  loop
4:      X, c ← SolveUB(Π_M ∪ Safe ∪ Unsafe, ub)
5:      if X = ⊥ then
6:          if Unsafe = ∅ then
7:              return last
8:          else
9:              Unsafe ← ∅
10:     else
11:         Y ← Solve(Π_S ∪ fact(X))
12:         if Y = ⊥ then
13:             U, M ← Extract(Π_S, X ∩ atoms(Π_S))
14:             Unsafe ← Unsafe ∪ cut_mcs(M)
15:             Safe ← Safe ∪ cut_mus(U)
16:         else
17:             last, ub ← Y, c
18:             Unsafe ← RemoveFirst(Unsafe)
```

4 MUS/MCS-Guided LBBD Solving

This section presents an algorithm that allows for applying both MUS- and MCS-Cuts to LBBD solving, combining the optimality-preserving nature of the former, with the "pruning power" of the latter.

Algorithm Description. Algorithm 1 takes as input an LBBD program $(\Pi_{\mathcal{M}}, \Pi_{\mathcal{S}})$ and iteratively searches for an optimal solution of $\Pi_{\mathcal{M}} \cup \Pi_{\mathcal{S}}$. It starts initialising the variables $last$ to $\bot$ and ub to $+\infty$ (line 1), which keep track of the best solution and the corresponding cost found so far, respectively. In line 2, the sets $Safe$ and $Unsafe$, used to store the MUSes and MCSes-Cuts, respectively, are initiallyised to $\emptyset$. The algorithm then calls the predefined function SolveUB that searches for an $X \in Ans(\Pi_{\mathcal{M}} \cup Safe \cup Unsafe)$ with cost c strictly smaller than the upper bound ub (line 4).[1] If such X does not exist and $Unsafe = \emptyset$, the algorithm terminates, returning the last solution found (line 7), while if $Unsafe \neq \emptyset$, it is cleared (line 9), and the algorithm resumes its search. This is done since inconsistency might be caused by one or more unsafe cuts. Otherwise, if a solution X exists, the predefined function Solve searches for a $Y \in Ans(\Pi_{\mathcal{S}} \cup fact(X))$ (line 11). If Y does not exist, an MUS and an MCS are extracted from $X \cap atoms(\Pi_{\mathcal{S}})$ with the predefined function Extract (line 13) and their cuts (see Definition 9) are added to $Unsafe$ and $Safe$ (lines 14 and 15), respectively. For a set S, $cut\_mus(S)$ (respectively, $cut\_mcs(S)$) returns the corresponding MUS-Cut (MCS-Cut). Lastly, if a valid answer set of the subprogram exists, it is stored in $last$, while the value of ub is updated to the cost c of X (line 17), and the first element in $Unsafe$ is removed (line 18). The algorithm then resumes the search, attempting to find a better solution than the one currently stored in $last$.

Note that removing at least one element from $Unsafe$ is not mandatory for the algorithm's correctness, but it avoids unnecessary computation. Indeed, if all MCS-Cuts are kept, the next invocation of the master program will remain unsatisfiable (as we force the solver to find a better answer set). While it is possible to clear the entire $Unsafe$ set, doing so would cause the master program to lose all learned information and restart the search from scratch. As a heuristic, we aim to preserve most of the previously learned cuts to guide the search, while still restoring satisfiability to the master program. In our implementation, we remove the oldest MCS-Cut (i.e., the first one added to $Unsafe$). This approach is inspired by *clause forgetting strategies* used in CDCL-based SAT (and ASP) solvers [Marques-Silva *et al.*, 2021], where older constraints are discarded to improve performance.

Theorem 5. *Algorithm 1 is correct and always terminates.*

Proof. We start by showing the correctness. By construction, the algorithm computes an LBBD solution (X, Y) if one exists. Thanks to Theorem 3, it follows that Y is also an answer set of $\Pi_{\mathcal{M}} \cup \Pi_{\mathcal{S}}$. Whenever Y is found, the algorithm stores it in $last$, and the upper bound ub is updated with its cost, corresponding to $cost(X, Y)$ (see Definition 6). The algorithm stops only if SolveUB is unable to find a solution with a cost strictly lower than ub and $Unsafe = \emptyset$. Hence, the only cuts considered are those in *Safe*, corresponding to MUS-Cuts. For Theorem 4, MUS-Cuts never remove LBBD solutions. Thus, *last* is optimal. Now, we prove that Algorithm 1 always terminates. The terminating condition relies on the fact that SolveUB always returns different solutions. Consider that SolveUB searches for an element $X \in Ans(\Pi_{\mathcal{M}} \cup Safe \cup Unsafe)$ with a cost c that is strictly lower than ub, namely, the cost of the last LBBD solution found. If such X exists, then it is employed to compute a solution Y for $\Pi_{\mathcal{S}} \cup fact(X)$. If Y exists, ub is updated with c and the algorithm proceeds by iterating on finding a (new) better solution; else, the MUS U is extracted and the corresponding cut added to *Safe*. This prevents finding the same X in the next iteration. If X does not exist, we observe that SolveUB can return $\bot$ at most two consecutive times. □

Handling Multiple Subproblems. Algorithm 1 can be naturally extended to handle an LBBD program $(\Pi_{\mathcal{M}}, \widetilde{\Pi}_{\mathcal{S}})$ with a subprogram partition $\widetilde{\Pi}_{\mathcal{S}}$ (i.e., *multiple subprograms*). The idea is to iterate on each subprogram in $\widetilde{\Pi}_{\mathcal{S}}$, performing for each of them the same checks described in lines 11–18, where the only relevant difference is that an answer set can be assigned to $last$ only if all subprograms admit at least one an-

[1] Note that SolveUB implements a simple optimality cut [Hooker, 2024].

swer set, given the model X of the master program. If one or more subprograms have no answer set, then an MUS and an MCS are extracted for each such subprogram and their relative cuts are added to the *Unsafe* and *Safe* sets, respectively, as described before.

Implementation. Algorithm 1 has been implemented as a practical tool built around core reasoning services, like functions SOLVEBOUND and SOLVE, which are supported by all major ASP solvers. The implementation uses the ASP system `clingo` [Gebser *et al.*, 2018] (version 5.6.1) as the underlying solver, with a Python script orchestrating multiple ASP solver calls, exploiting `clingo` multi-shot solving capabilities. MUS computation is achieved by shrinking unsatisfiable cores (as found by `clingo` after a solve call) by a linear elimination strategy, while MCS extraction is achieved by solving an optimisation program $\Pi_{\mathcal{S}}^{w}$, obtained by the extension of $\Pi_{\mathcal{S}}$ with a weak constraint $\leftarrow\!\!\sim p.\ [1@1, p]$, for each atom p discerning different answer sets of $\Pi_{\mathcal{M}}$, i.e., there exists $A, A' \in Ans(\Pi_{\mathcal{M}})$ where $p \in A$ and $p \notin A'$. Algorithm 1 requires many calls to an ASP oracle which may be computationally expensive. Therefore, it is important to reduce as much as possible such calls and use timeouts to avoid spending too much time on a single iteration. In practice, we start by solving $\Pi_{\mathcal{S}}^{w}$ under a time budget to search for a correction set. Empty correction sets indeed show that $\Pi_{\mathcal{S}}$ is actually coherent. If we are unable to prove optimality of $\Pi_{\mathcal{S}}^{w}$ but we find a correction set, we can obtain an unsafe cut, but no safe cut (we did not prove the optimal cost of $\Pi_{\mathcal{S}}^{w}$ is zero). When we are able to prove optimality of $\Pi_{\mathcal{S}}^{w}$, we obtain the smallest correction set, and we can easily obtain also an unsastisfiable set. We then further invest a time budget to minimise this unsatisfiable core (leading to shorter safe cuts that are typically more useful in the solving phase).

5 Experiments

This section presents our experiments. The material used to reproduce them is available as supplementary material.

To the best of our knowledge, only three existing approaches combine LBBD with ASP. The first, by Cappanera *et al.* [2023], cannot be directly compared to our approach as it focuses on handling programs with high number of subproblems, requiring the usage of parallelism, which is not currently supported by our tool. The second, by Dodaro *et al.* [2026], includes 370 real-world instances of the Nuclear Medicine Scheduling (NMS) problem. The third, by Caruso *et al.* [2025], includes 4 real-world instances of the Chemotherapy Treatment Scheduling (CTS) problem. For the last two problems, the ASP program that solves the NMS/CTS problem as a whole is referred to as DIRECT; and the corresponding LBBD-based encoding is referred to as LBBD. For LBBD approaches, the cuts used to refine the master program were created by the authors and are domain-specific. The NMS cuts are unsafe, while the CTS cuts' safety has not been formally proved. However, we assume their safety, since we verified that the obtained optima match the ones of the original formulation. In the following, Algorithm 1 is referred to as MUS+MCS. We do not consider variants of Algorithm 1 using only MCS-cuts or only MUS-cuts, as the former leads to an incomplete approach, while the latter has shown to be supbar to using both in preliminary experiments. Experiments were executed on an Intel(R) Xeon(R) CPU E7-8880 v4 @ 2.20GHz server with 500GB RAM and 88 physical cores, running up to 16 concurrent jobs. For both problems, each run was limited to a timeout of 180 CPU seconds. For the LBBD-based encodings, we use the same parameters of the original papers. In particular, for NMS we set the timeout for the master and sub programs to 5 and 3 seconds, respectively, while for CTS we set the master timeout to 180 seconds and the subprogram timeout to 15 seconds.

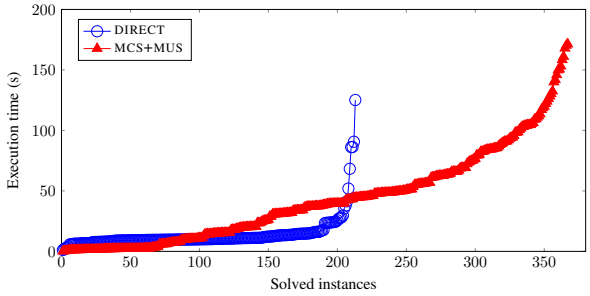

Figure 1: Number of solved instances in terms of optimal solutions (x-axis) within a time limit (y-axis).

5.1 NMS

Results for the NMS are reported in Table 1 where we show, for each approach, the number of instances for which a feasible solution was found and the number of instances solved to proven optimality. In addition, since LBBD can be regarded as an incomplete system, we also evaluate it using performance indicators inspired by the incomplete track of recent MaxSAT Evaluations (columns Score and ScoreOpt). In particular, for an instance i and a solver s, we define $\text{Score}_s(i) = \frac{\text{cost}^\star(i)+1}{\text{cost}_s(i)+1}$, where $\text{cost}^\star(i)$ denotes the best cost achieved on i among all solvers, and $\text{cost}_s(i)$ is the cost returned by s. $\text{Score}_s(i)$ is 0 if the solver s fails to produce any solution for the instance i. However, Score does not capture whether a solver is able to prove optimality. To take into consideration also this aspect, we consider a variant, denoted as ScoreOpt, which adds a bonus of 0.2 on each instance for which the solver successfully certifies optimality. For both Score and ScoreOpt, we report in the table the average value over all instances. As a first observation, MUS+MCS outperforms DIRECT in terms of optimal solutions (257 vs. 213), but it finds less solutions overall (367 vs. 370). Moreover, MUS+MCS obtains better values for both Score and ScoreOpt. The better performance is demonstrated also in terms of solving times, as shown in the cactus plots of Fig-

Approach	Solved	Optimal	Score	ScoreOpt
MUS+MCS	367	**257**	0.891	**1.030**
LBBD	**370**	-	**0.998**	0.998
DIRECT	**370**	213	0.703	0.818

Table 1: Number of (optimal) solutions found and (optimal) scores on NMS instances.

ure 1. We recall that a point (x, y) on a curve means that the approach solved optimally x instances within y seconds. It is possible to observe that DIRECT performs well on simpler instances but shows a limited scalability. In contrast, MUS+MCS solves more instances within the time limit and it scales better than DIRECT.

When compared with LBBD, we observe a different trade-off. Indeed, LBBD obtains the largest number of feasible solutions with no possibility of proving the optimality. In contrast, MUS+MCS finds slightly fewer feasible solutions (367 vs 370), but it proves that 257 of them are optimal. As for the scores the two approaches are very close, LBBD is an incomplete system and obtains a higher Score than MUS+MCS (0.998 vs 0.891), while MUS+MCS is a complete system and obtains a higher $\mathrm{ScoreOpt}$ (1.030 vs 0.998).

5.2 CTS

Concerning CTS, DIRECT does not solve any instance within the timeout. In contrast, both MUS+MCS and LBBD prove optimality on three instances, while they do not return any feasible solution on the remaining one. As for the runtime, MUS+MCS is slightly slower, with an average solving time of 118 seconds, compared to 78 seconds for DIRECT.

With respect to Score and $\mathrm{ScoreOpt}$, MUS+MCS and LBBD achieve identical average values, namely 0.75 and 0.90, respectively. These results show that MUS+MCS is competitive with the manual and domain-specific approach and consistently outperforms DIRECT. Finally, although MUS+MCS and LBBD solve the same number of instances, they succeed on different subsets as MUS+MCS solves instances 1, 3, and 4, whereas LBBD solves the first three instances.

6 Related Work

The LBBD [Hooker and Ottosson, 2003; Geoffrion, 1972; Rahmaniani *et al.*, 2017] approach has been extensively applied to combinatorial optimisation problems, particularly in cases where the problem can be naturally decomposed into multiple interconnected subproblems. In scheduling applications, the LBBD approach has been recently used in the context of ASP for solving three problems from the healthcare domain [Cappanera *et al.*, 2023; Dodaro *et al.*, 2026; Caruso *et al.*, 2025]. The decomposition and cuts are problem-specific, require manual design, integration, and deep knowledge of the target domain. Although very effective, the optimality of the solution must be checked for each problem. Conversely, our algorithm is problem-independent, relying on structural properties of the ASP program according to the splitting theorem, does not require domain-specific knowledge to define cuts, and preserves optimality.

Our contributions rely on the notions of MUSes and MCSes [Alviano *et al.*, 2023], which are concepts widely studied in SAT/MaxSAT, and in ASP have been used in unsatisfiability-based reasoning [Alviano *et al.*, 2024; Dodaro *et al.*, 2019] and the notion of *unsatisfiable sets* (or *cores*) has been widely used to support the solving of programs with weak constraints [Andres *et al.*, 2012; Alviano *et al.*, 2020]. Saikko *et al.* [2018] adapted to ASP an Implicit Hitting Set (IHS) approach for evaluationg programs with weak constraints, building on the techniques used in MaxSAT [Davies and Bacchus, 2011]. In more details, their approach collects cores using an ASP solver and then computes a (minimal) hitting set (MHS) of such cores, using a dedicated integer programming (IP) solver, in an attempt to derive an optimal solution. This work is related to this paper, since an MCS can be seen as an MHS of all MUSes [Reiter, 1987]. Moreover, both approaches aim to minimise unsatisfiable sets under a given resource limit, and return a non-minimal core when this limit is exceeded [Alviano and Dodaro, 2016]. Nevertheless, there are also important differences. First, our work is more general, as we allow a broader notion of unsatisfiable core, and not only a subset of weak constraints that appear in the program. This generality comes at a higher technical cost, since correctness of the decomposition must be ensured; weak (and hard) constraints represent a special case within our theoretical framework. Second, while Saikko *et al.* [2018] rely on an ASP solver to extract cores and on an IP solver to compute an MHS, we use the same ASP solver to compute both MUSes and MCSes. In particular, we rely on optimisation statements to directly compute a minimum-size correction set of the program, rather than an external solver that computes hitting sets over a collection of previously identified cores. Third, our method does not require the computation of a minimal unsatisfiable/correction set, as it suffices to produce an unsatisfiable set and a correction set, whereas the approach of Saikko *et al.* [2018] must compute an MHS over the collected unsatisfiable sets to guarantee optimality. Fourth, Saikko *et al.* [2018] use MCSes to relax weak constraints, while we use MCSes as a heuristic to guide the search.

Finally, our method uses unsatisfiable cores as domain-agnostic cuts, similarly to Davies *et al.* [2017] who introduced automatic LBBD in MiniZinc [Nethercote *et al.*, 2007]. There are, however, several differences. First, our use of correction sets to handle multiple unsatisfiable cores is novel. Second, our framework natively supports multiple subproblems, in contrast with their single-subproblem formulation. Third, our implementation allows users to explicitly declare the objective atoms associated with the subproblem(s), while Davies *et al.* [2017] perform this step automatically.

7 Conclusion

In this paper, we formalised conditions, relying on the splitting theorem, to guarantee that a decomposition with ASP reflects the properties of the LBBD approach. Moreover, we perform a theoretical analysis of the conditions under which LBBD cuts in ASP reflect feasibility/optimality cuts and extend them to more subprograms. Exploiting MUSes, we generate safe cuts in a fully automated, problem-independent way. We proposed an algorithm that employs the concepts above and also correctly handles unsafe cuts generated from MCSes. We tested it on scheduling instances, showing advantages for our proposal in terms of CPU time over a direct approach while preserving the optimality. Future work includes extending our theoretical analysis to settings in which the subprograms are also optimisation programs, and supporting parallelism so that other domains can be analysed.

Acknowledgements

This work has been partially supported by the Italian "Ministero delle Imprese e Sviluppo Economico" (MIMIT) under projects "Ei-TWIN - Energy Digital Twin" (CUP B29J2400068000); by Regione Calabria under PR Calabria FESR FSE 2021–2027 through the projects "IOS4OS – Intelligent Optimization System for Optimization Solutions" (CUP J89I24002920005), "MOZART - Modelli e tecniche di mOdernizzazione di applicaZioni e data silos a supporto di clienti esterni, field force e impiegati di backoffice basati su AI GeneRativa e apprendimento auTomatico" (CUP J49I24001740005), and "SIGENERA - Large Language Models (LLM) per il controllo di SIstemi informativi, la GENERazione automatica di testo, e l'accesso a basi di conoscenzA" (CUP J29I24001770005); and by the Austrian Science Fund (FWF) projects 10.55776/PAT1599524. Marco Maratea was supported by the European Union - NextGenerationEU and by Italian Ministry of Research (MUR) under PNRR project FAIR "Future AI Research", CUP H23C22000860006.

References

[Alviano and Dodaro, 2016] Mario Alviano and Carmine Dodaro. Anytime answer set optimization via unsatisfiable core shrinking. *Theory Pract. Log. Program.*, 16(5-6):533–551, 2016.

[Alviano *et al.*, 2020] Mario Alviano, Carmine Dodaro, João Marques-Silva, and Francesco Ricca. Optimum stable model search: algorithms and implementation. *J. Log. Comput.*, 30(4):863–897, 2020.

[Alviano *et al.*, 2023] Mario Alviano, Carmine Dodaro, Salvatore Fiorentino, Alessandro Previti, and Francesco Ricca. ASP and subset minimality: Enumeration, cautious reasoning and muses. *Artif. Intell.*, 320:103931, 2023.

[Alviano *et al.*, 2024] Mario Alviano, Susana Hahn, Orkunt Sabuncu, and Hannes Weichelt. Answer set explanations via preferred unit-provable unsatisfiable subsets. In *Proc. of LPNMR*, volume 15245 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 2024.

[Andres *et al.*, 2012] Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in clasp. In *Proc. of ICLP Technical Communications*, volume 17 of *LIPIcs*, pages 211–221. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.

[Brewka *et al.*, 2011] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

[Calimeri *et al.*, 2020] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Marco Maratea, Francesco Ricca, and Torsten Schaub. ASP-Core-2 input language format. *Theory Pract. Log. Program.*, 20(2):294–309, 2020.

[Cappanera *et al.*, 2023] Paola Cappanera, Marco Gavanelli, Maddalena Nonato, and Marco Roma. Logic-based benders decomposition in answer set programming for chronic outpatients scheduling. *Theory Pract. Log. Program.*, 23(4):848–864, 2023.

[Caruso *et al.*, 2025] Simone Caruso, Carmine Dodaro, Marco Maratea, Cinzia Marte, and Marco Mochi. An lbbd approach for solving the chemotherapy treatment scheduling problem. In *Proc. of HC@AIxIA + HYDRA 2025*, 2025.

[Davies and Bacchus, 2011] Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In *Proc. of CP*, volume 6876 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2011.

[Davies *et al.*, 2017] Toby O. Davies, Graeme Gange, and Peter J. Stuckey. Automatic logic-based benders decomposition with minizinc. In *Proc. of AAAI*, pages 787–793. AAAI Press, 2017.

[Dodaro *et al.*, 2019] Carmine Dodaro, Philip Gasteiger, Kristian Reale, Francesco Ricca, and Konstantin Schekotihin. Debugging non-ground ASP programs: Technique and graphical tools. *Theory Pract. Log. Program.*, 19(2):290–316, 2019.

[Dodaro *et al.*, 2026] Carmine Dodaro, Giuseppe Galatà, Marco Maratea, Cinzia Marte, and Marco Mochi. Asp-based approaches for solving the nuclear medicine scheduling problem. *J. Log. Comput.*, 1, 2026.

[Erdem *et al.*, 2016] Esra Erdem, Michael Gelfond, and Nicola Leone. Applications of answer set programming. *AI Mag.*, 37(3):53–68, 2016.

[Falkner *et al.*, 2018] Andreas A. Falkner, Gerhard Friedrich, Konstantin Schekotihin, Richard Taupe, and Erich Christian Teppan. Industrial applications of answer set programming. *Künstliche Intell.*, 32(2-3):165–176, 2018.

[Gebser *et al.*, 2018] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Patrick Lühne, Philipp Obermeier, Max Ostrowski, Javier Romero, Torsten Schaub, Sebastian Schellhorn, and Philipp Wanko. The potsdam answer set solving collection 5.0. *Künstliche Intell.*, 32(2-3):181–182, 2018.

[Gelfond and Lifschitz, 1991] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Gener. Comput.*, 9(3/4):365–386, 1991.

[Geoffrion, 1972] Arthur M Geoffrion. Generalized benders decomposition. *Journal of optimization theory and applications*, 10:237–260, 1972.

[Hooker and Ottosson, 2003] John N Hooker and Greger Ottosson. Logic-based benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003.

[Hooker, 2024] John Hooker. Logic-based benders decomposition: Theory and applications. *Synthesis Lectures on Operations Research and Applications*, 2024.

[Lifschitz and Turner, 1994] Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In *Proc. of ICLP*, pages 23–37. MIT Press, 1994.

[Marques-Silva *et al.*, 2021] João Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning SAT solvers. In *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 133–182. IOS Press, 2021.

[Nethercote *et al.*, 2007] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In *CP*, volume 4741 of *Lecture Notes in Computer Science*, pages 529–543. Springer, 2007.

[Rahmaniani *et al.*, 2017] Ragheb Rahmaniani, Teodor Gabriel Crainic, Michel Gendreau, and Walter Rei. The benders decomposition algorithm: A literature review. *Eur. J. Oper. Res.*, 259(3):801–817, 2017.

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

[Saikko *et al.*, 2018] Paul Saikko, Carmine Dodaro, Mario Alviano, and Matti Järvisalo. A hybrid approach to optimization in answer set programming. In *Proc. of KR*, pages 32–41. AAAI Press, 2018.