Introducing Real Variables and Integer Objective Functions to Answer Set Programming*

Guohua Liu, Tomi Janhunen, and Ilkka Niemelä

Helsinki Institute for Information Technology HIIT
Department of Information and Computer Science
Aalto University, FI-00076 AALTO, FINLAND
{Guohua.Liu,Tomi.Janhunen,Ilkka.Niemela}@aalto.fi

Abstract. Answer set programming languages have been extended to support linear constraints and objective functions. However, the variables allowed in the constraints and functions are restricted to integer and Boolean domains, respectively. In this paper, we generalize the domain of linear constraints to real numbers and that of objective functions to integers. Since these extensions are based on a translation from logic programs to mixed integer programs, we compare the translation-based answer set programming approach with the native mixed integer programming approach using a number of benchmark problems.

1 Introduction

Answer set programming (ASP) [14], also known as logic programming under *stable model* semantics [8], is a declarative programming paradigm where a given problem is solved by devising a logic program whose *answer sets* capture the solutions of the problem and then by computing the answer sets using *answer set solvers*. The paradigm has been exploited in a rich variety of applications [2].

Linear constraints have been introduced to ASP [1, 7, 11, 12] in order to combine the high-level modeling capabilities of ASP languages with the efficient constraint solving techniques developed in the area of constraint programming. In particular, a language ASP(LC) is devised in [11] which allows linear constraints to be used within the original ASP language structures. The answer set computation for ASP(LC) programs is based on mixed integer programming (MIP) where an ASP(LC) program is first translated into a MIP program and then the solutions of the MIP program are computed using a MIP solver. Finally, answer sets can be recovered from the solutions found (if any).

In this paper, we extend and evaluate the ASP(LC) language in the following aspects. First, we generalize the domain of variables allowed in linear constraints from integers to reals. Real variables are ubiquitous in applications, e.g., timing variables in scheduling problems. However, the MIP-based answer set computation confines the variables in linear constraints to the integer domain. We overcome this limitation by developing a translation of ASP(LC) programs to MIP programs so that constraints over

* The support from the Finnish Centre of Excellence in Computational Inference Research (COIN) funded by the Academy of Finland (under grant #251170) is gratefully acknowledged.

real variables are enabled in the language. Second, we introduce MIP objective functions, i.e., linear functions of integer variables, to the ASP(LC) language. The original ASP(LC) language allows objective functions of Boolean variables only, but integer variables are more convenient than Booleans in many applications [11]. To model optimization problems in these areas, we enable MIP objective functions in ASP by giving semantics for ASP programs with these functions. Third, we compare ASP(LC) to MIP. This is interesting as ASP(LC) provides a richer language than MIP where ASP language structures are extended with linear constraints but the implementation technique is based on translating an ASP(LC) program to a MIP program to solve. We choose some representative problems, study the ASP(LC) and MIP encodings of the problems, and evaluate their computational performance by experiments.

The rest of the paper is organized as follows. Preliminaries are given in Section 2. Then we extend ASP(LC) language with real variables in Section 3, introduce MIP objective functions in Section 4, and compare ASP(LC) and native MIP formulations in Section 5. The experiments are reported in Section 6 followed by a discussion on the related work in Section 7. The paper is concluded by Section 8.

2 Preliminaries

In this section, we review the basic concepts of linear constraints, mixed integer programming, and the ASP(LC) language. A *linear constraint* is an expression of the form

$$\sum_{i=1}^{n} u_i x_i \sim k \tag{1}$$

where the u_i's and k are real numbers and the x_i's are variables ranging over real numbers (including integers). We distinguish the variables to be *real* and *integer* variables when necessary. The operator $\sim$ is in $\{<, \leq, \geq, >\}$. Constraints involving "$<$" and "$>$" are called *strict* constraints. A valuation ν from variables to numbers is a *solution* of (or *satisfies*) a constraint C of the form (1), denoted $\nu \models C$, iff $\sum_{i=1}^{n} u_i \nu(x_i) \sim k$ holds. A valuation ν is a solution of a set of constraints $\mathit{\Pi} = \{C_1, ..., C_m\}$, denoted $\nu \models \mathit{\Pi}$, iff $\nu \models C_i$ for each $C_i \in \mathit{\Pi}$. A set of linear constraints is *satisfiable* iff it has a solution.

A *mixed integer program* (or a *MIP program*), takes the form

$$\texttt{optimize} \quad \sum_{i=1}^{n} u_i x_i \tag{2}$$

$$\texttt{subject to} \; C_1, ..., C_m. \tag{3}$$

where the keyword `optimize` is `minimize` or `maximize`, u_i's are numbers, x_i's are variables, and C_i's are linear constraints. The operators in the constraints are in $\{\leq, =, \geq\}$. The function $\sum_{i=1}^{n} u_i x_i$ is called an *objective function*. The constraints $C_1, ..., C_m$ may be written as a set $\{C_1, ..., C_m\}$. A valuation ν is a solution of a MIP program iff $\nu \models \{C_1, ..., C_m\}$. A solution is *optimal*, iff it minimizes (or maximizes) the value of the objective function. The objective function could be empty (missing from a MIP program), in which case the function is trivially optimized by any solution. The keywords `optimize` and `subject to` may be omitted if the objective function is empty. The goal of MIP is to find the optimal solutions of a MIP program.

An *ASP(LC) program* is a set of rules of the form

$$a \leftarrow b_1, \ldots, b_n, \text{not } c_1, \ldots, \text{not } c_m, t_1, \ldots, t_l \tag{4}$$

where each a, b_i, and c_i is a propositional atom and each t_i, called a *theory atom*, is a linear constraint of the form (1). Propositional atoms and theory atoms may be uniformly called *atoms*. Atoms and atoms preceded by "not" are also referred to as *positive* and *negative literals*, respectively. Given a program P, the set of propositional and theory atoms appearing in P are denoted by $\mathcal{A}(P)$ and $\mathcal{T}(P)$, respectively. For a rule r of the form (4), the *head* and the *body* of r are defined by $\mathrm{H}(r) = \{a\}$ and $\mathrm{B}(r) = \{b_1, \ldots, b_n, \text{not } c_1, \ldots, \text{not } c_m, t_1, \ldots, t_l\}$. Furthermore, the *positive*, *negative*, and *theory* parts of the body are defined as $\mathrm{B}^+(r) = \{b_1, \ldots, b_n\}$, $\mathrm{B}^-(r) = \{c_1, \ldots, c_m\}$, and $\mathrm{B}^t(r) = \{t_1, \ldots, t_l\}$, respectively. The body and the head of a rule could be empty: a rule without body is a *fact* whose head is true unconditionally and a rule without head is an *integrity constraint* enforcing the body to be false.

A set of atoms M satisfies an atom a, denoted $M \models a$, iff $a \in M$, and it satisfies a negative literal 'not a', denoted $M \models \text{not } a$, iff $a \notin M$. The set M satisfies a set of literals $L = \{l_1, \ldots, l_n\}$, denoted $M \models L$, iff $M \models l_i$ for each $l_i \in L$. An *interpretation* of an ASP(LC) program P is a pair $\langle M, T\rangle$ where $M \subseteq \mathcal{A}(P)$ and $T \subseteq \mathcal{T}(P)$, such that $T \cup \bar{T}$ is satisfiable in linear arithmetics where $\bar{T} = \{\neg t \mid t \in \mathcal{T}(P) \text{ and } t \notin T\}$ and $\neg t$ denotes the constraint obtained by changing the operator of t to the complementary one. Two interpretations $I_1 = \langle M_1, T_1\rangle$ and $I_2 = \langle M_2, T_2\rangle$ are *equal*, denoted $I_1 = I_2$, iff $M_1 = M_2$ and $T_1 = T_2$. An interpretation $I = \langle M, T\rangle$ satisfies a literal l iff $M \cup T \models l$. An interpretation I satisfies a rule r, denoted $I \models r$, iff $I \models \mathrm{H}(r)$ or $I \not\models \mathrm{B}(r)$. An integrity constraint is satisfied by I iff $I \not\models \mathrm{B}(r)$. An interpretation I is a *model* of a program P, denoted $I \models P$, iff $I \models r$ for each $r \in P$.

Answer sets are defined using the concept of program *reduct* as follows.

Definition 1 (Liu et al. [11]). *Let P be an ASP(LC) program and $\langle M, T\rangle$ an interpretation of P. The* reduct *of P with respect to $\langle M, T\rangle$, denoted $P^{\langle M,T\rangle}$, is defined as $P^{\langle M,T\rangle} = \{\mathrm{H}(r) \leftarrow \mathrm{B}^+(r) \mid r \in P, \mathrm{H}(r) \neq \emptyset,\ \mathrm{B}^-(r) \cap M = \emptyset,\ \text{and } \mathrm{B}^t(r) \subseteq T\}$.*

Definition 2 (Liu et al. [11]). *Let P be an ASP(LC) program. An interpretation $\langle M, T\rangle$ is an* answer set *of P iff $\langle M, T\rangle \models P$ and M is the subset minimal model of $P^{\langle M,T\rangle}$. The set of answer sets of P is denoted by $AS(P)$.*

Example 1. Let P be an ASP(LC) program consisting of the rules

$$a \leftarrow x - y \leq 2. \quad b \leftarrow x - y \geq 5. \quad \leftarrow x - y \geq 0.$$

The interpretation $I_1 = \langle\{a\}, \{x - y \leq 2\}\rangle$ is an answer set of P since $\{(x - y \leq 2), \neg(x - y \geq 5), \neg(x - y \geq 0)\}$ is satisfiable in linear arithmetics, $I_1 \models P$, and $\{a\}$ is the minimal model of $P^{I_1} = \{a \leftarrow .\}$. The interpretation $I_2 = \langle\{b\}, \{x - y \geq 5\}\rangle$ is not an answer set since $\{(x - y \geq 5), \neg(x - y \leq 2), \neg(x - y \geq 0)\}$ is unsatisfiable. Finally, $I_3 = \langle\emptyset, \{x - y \geq 0\}\rangle$ is not an answer set, since $I_3 \not\models P$. □

Syntactically, theory atoms are allowed as heads of rules in the current implementation [11] for more intuitive reading and thus such rules are used in this paper. As

regards their semantics, a rule with a theory atom as the head is equivalent to an integrity constraint, i.e., a rule $t \leftarrow a_1, \ldots, a_m, \text{not } b_1, \ldots, \text{not } b_n, t_1, \ldots, t_l$ where t is a theory atom is treated as the rule $\leftarrow a_1, \ldots, a_m, \text{not } b_1, \ldots, \text{not } b_n, t_1, \ldots, t_l, \neg t$ in answer set computation. Moreover, the semantics of ASP(LC) programs coincides with that of *normal* logic programs [8, 14] if no theory atoms are present.

Answer set computation for ASP(LC) programs is based on a translation to MIP programs [11]. We will refer to the translation as *MIP-translation* and denote the translation of a program P by $\tau(P)$. Due to space limitations, we skip a thorough review of $\tau(P)$ and focus on a fragment most relevant for this paper, i.e., the rules of the form

$$a \leftarrow t. \tag{5}$$

where a is an propositional atom or not present at all and t is a theory atom. Recall that a rule without head is an integrity constraint.

In the translation, special linear constraints called *indicator constraints* are used. An indicator constraint is of the form $d = v \rightarrow C$ where d is a *binary variable* (integer variable with the domain $\{0, 1\}$), v is either 0 or 1, and C is a linear constraint. An indicator constraint is *strict* if C is strict and *non-strict* otherwise. An indicator constraint can be written as a constraint (1) using the so-called *big-M* formulation.

For a program P consisting of simple rules (5) only, $\tau(P)$ is formed as follows:

1. For each theory atom t, we include a pair of indicator constraints

$$d = 1 \rightarrow t \qquad d = 0 \rightarrow \neg t \tag{6}$$

   where d is a new binary variable introduced for t. The idea is to use the variable d to represent the constraint t in the sense that, for any solution ν of the constraints in (6), $\nu(d) = 1$ iff $\nu \models t$. Thus d can be viewed as a kind of a *name* for t.
2. Assuming that $a \leftarrow t_1, \ldots, a \leftarrow t_k$ are all rules (5) that have a as head, we include

$$a - d_1 \geq 0, \quad \ldots, \quad a - d_k \geq 0, \tag{7}$$

$$d_1 + \ldots + d_k - a \geq 0 \tag{8}$$

   where $d_1, \ldots, d_k$ are the binary variables corresponding to $t_1, \ldots, t_k$ in (6). The constraints in (7) and (8) enforce that the joint head a holds iff some of the bodies $t_1, \ldots, t_k$ holds which is compatible with Clark's completion [3]. If $k = 1$, i.e., the atom a has a unique defining rule, the constraints of (7) and (8) reduce to $a - d_1 = 0$ which makes d_1 synonymous with a. Moreover, if the rule (5) is an integrity constraint, then $d_1 = 0$ is sufficient, as intuitively implied by $k = 1$ and $a = 0$.

In the implementation of $\tau(P)$, more variables and constraints are used to cover the rules of the general form (4). We refer the reader to [11] for details.

The solutions of the MIP-translation of a program capture its answer sets as follows. Let P be an ASP(LC) program and ν a mapping from variables to numbers. We define the *ν-induced interpretation* of P, denoted I^ν_P, by setting $I^\nu_P = \langle M, T \rangle$ where

$$M = \{a \mid a \in \mathcal{A}(P),\ \nu(a) = 1\} \text{ and} \tag{9}$$

$$T = \{t \mid t \in \mathcal{T}(P),\ \nu \models t\}. \tag{10}$$

Theorem 1 (Liu et al. [11]). *Let P be an ASP(LC) program.*

1. *If ν is a solution of $\tau(P)$, then $I^\nu_P \in AS(P)$.*
2. *If $I \in AS(P)$, then there is a solution ν of $\tau(P)$ such that $I = I^\nu_P$.*

Example 2. For the program P from Example 1, the translation $\tau(P)$ consists of:

$$\begin{array}{lll} d_1 = 1 \to x - y \leq 2, & d_1 = 0 \to x - y > 2, & a - d_1 = 0, \\ d_2 = 1 \to x - y \geq 5, & d_2 = 0 \to x - y < 5, & b - d_2 = 0, \\ d_3 = 1 \to x - y \geq 0, & d_3 = 0 \to x - y < 0, & d_3 = 0. \end{array}$$

For any solution ν of $\tau(P)$, we have $\nu(a) = 1$, $\nu(b) = 0$, and $\nu(x) - \nu(y) \leq 2$ which characterize the unique answer set $\langle\{a\}, \{x - y \leq 2\}\rangle$ of P. □

3 Extension with Real Variables

In this section, we first illustrate how strict constraints involved in the MIP-translation prevent the introduction of real variables in ASP(LC) programs. Motivated by these observations, we develop a translation of strict constraints to non-strict ones. Finally, we apply the translation to remove strict constraints from the MIP-translation so that real variables can be allowed in ASP(LC) programs.

3.1 Problems Caused by Real Variables

Real variables are widely used in knowledge representation and reasoning. However, the computation of answer sets based on the MIP-translation becomes problematic in their presence. The reason is that typical MIP systems do not fully support strict constraints involving real variables, e.g., by treating strict constraints as non-strict ones. Consequently, the correspondence between solutions and answer sets may be lost.

Example 3. Consider the condition that Tom gets a bonus if he works at least 8.25 hours and the fact that he works for that long. By formalizing these constraints we obtain an ASP(LC) program P consisting of the following rules:

$$\mathsf{bonus}(tom) \leftarrow h(tom) \geq 8.25. \tag{11}$$

$$\leftarrow h(tom) < 8.25. \tag{12}$$

In the above, the ground term $h(tom)$ is treated as a real variable recording the working hours of Tom and $\mathsf{bonus}(tom)$[1] is a ground (propositional) atom meaning that Tom will be paid a bonus. The MIP-translation $\tau(P)$ of P has the following constraints:

$$d_1 = 1 \to h(tom) \geq 8.25 \tag{13}$$

$$d_1 = 0 \to h(tom) < 8.25 \tag{14}$$

$$\mathsf{bonus}(tom) - d_1 = 0 \tag{15}$$

$$d_2 = 1 \to h(tom) < 8.25 \tag{16}$$

$$d_2 = 0 \to h(tom) \geq 8.25 \tag{17}$$

$$d_2 = 0 \tag{18}$$

[1] We use different fonts for function and predicate symbols, such as "h" and "bonus" in this example, for clarity.

Given $\tau(P)$ as input, CPLEX provides a solution ν where $\nu(\mathsf{bonus}(tom)) = \nu(d_1) = \nu(d_2) = 0$ and $\nu(h(tom) = 8.25)$. However $I_P^\nu = \langle \emptyset, \{h(tom) = 8.25\} \rangle$ is not an answer set of P since it does not satisfy the rule (11). This discrepancy is due to the fact that ν actually does not satisfy the strict constraint (14), but CPLEX treats this as the non-strict one $d_1 = 0 \rightarrow h(tom) \leq 8.25$ and unexpectedly gives ν as a solution. □

The current implementation of the MIP-translation [11] addresses only integer-valued constraints where the coefficients and variables range over integers. Given this restriction, strict constraints of the form $\sum_{i=1}^{n} u_i x_i < k$ (resp. $> k$) can be implemented as non-strict ones $\sum_{i=1}^{n} u_i x_i \leq k - 1$ (resp. $\geq k + 1$).

It might be tempting to convert the domain of a problem from reals to integers, e.g., by multiplying the constraints by 100 and by replacing the variables $h(tom)$ by another holding a hundredfold value. For the program P in Example 3 this would give rise to:

$$\mathsf{bonus}(tom) \leftarrow h'(tom) \geq 825. \tag{19}$$

$$\leftarrow h'(tom) < 825. \tag{20}$$

Thereafter constraints (13) and (14) could be rewritten as non-strict constraints:

$$d_1 = 1 \rightarrow h'(tom) \geq 825. \tag{21}$$

$$d_1 = 0 \rightarrow h'(tom) \leq 824. \tag{22}$$

This approach, however, does not work in general. First, the translated program cannot cover the domain of the original problem due to the continuity of real numbers. For example, the rules (21) and (22) do not give any information about the working hours 8.245 which is covered by (13) and (14). Second, determining the required coefficients is infeasible in general since the real numbers occurring in constraints can be specified up to arbitrary precision which could vary from problem instance to another.

Because CPLEX treats strict constraints as non-strict ones, the MIP-translation becomes inapplicable for answer set computation in the presence of real-valued variables. To enable such computations, a revised translation which consists of non-strict constraints only is needed. Such a translation is devised in sections to come.

3.2 Non-Strict Translation of Strict Constraints

We focus on strict constraints of a restricted form $y > 0$. This goes without loss of generality because any constraint $\sum_{i=1}^{n} u_i x_i > k$ can be rewritten as a conjunction of a non-strict constraint $\sum_{i=1}^{n} u_i x_i - y = k$ and a strict one $y > 0$ where y is fresh. Also, a constraint of the form $\sum_{i=1}^{n} u_i x_i < k$ is equivalent to $-\sum_{i=1}^{n} u_i x_i > -k$.

Lemma 1. *Let Γ be a set of non-strict constraints, $S = \{x_1 > 0, \ldots, x_n > 0\}$, and δ a new variable. Then, the set $\Gamma \cup S$ is satisfiable iff for any bound $b > 0$, the set $\Gamma \cup S_\delta \cup \{0 < \delta \leq b\}$ where $S_\delta = \{x_1 \geq \delta, \ldots, x_n \geq \delta\}$ is satisfiable.*

Proof. We prove the direction "⇒" since the other direction is obvious. Since $\Gamma \cup S$ is satisfiable, there is a valuation ν such that $\nu \models \Gamma$ and $\nu(x_i) > 0$ for each $1 \leq i \leq n$. Let $b > 0$ be any number and $m = \min\{\nu(x_1), \ldots, \nu(x_n)\}$. Then $\nu(x_i) \geq m$ holds for any $1 \leq i \leq n$ and $m > 0$. Two cases arise and need to be analyzed:

Case 1: If $m \leq b$, then $\Gamma \cup S_\delta \cup \{0 < \delta \leq b\}$ has a solution ν' which extends ν by the assignment $\nu'(\delta) = m$.

Case 2: If $m > b$, define ν' as an extension of ν such that $\nu'(\delta) = b$. Thus $b > 0$ implies $\nu' \models 0 < \delta \leq b$. Moreover, for any $1 \leq i \leq n$, $\nu'(x_i) = \nu(x_i) \geq \nu'(\delta) = b$ since $m > b$. Therefore $\nu' \models x_i \geq \delta$ for any $1 \leq i \leq n$.

It follows that $\nu' \models \Gamma \cup S_\delta \cup \{0 < \delta \leq b\}$. □

The result of Lemma 1 can be lifted to the case of indicator constraints since indicator constraints are essentially linear constraints.

Lemma 2. *Let Γ be a set of non-strict constraints,*

$$S = \{d_i = v_i \rightarrow x_i > 0 \mid 1 \leq i \leq n\} \tag{23}$$

a set of strict indicator constraints, and δ a new variable. Then, $\Gamma \cup S$ is satisfiable iff for any bound $b > 0$, $\Gamma \cup S_\delta \cup \{0 < \delta \leq b\}$ is satisfiable where

$$S_\delta = \{d_i = v_i \rightarrow x_i \geq \delta \mid 1 \leq i \leq n\}. \tag{24}$$

Lemma 2 shows that a set of strict indicator constraints can be transformed to a set of non-strict ones by introducing a new bounded variable $0 < \delta \leq b$. Below, we relax the last remaining strict constraint $\delta > 0$ to $\delta \geq 0$ using a MIP objective function.

Definition 3. *Let $\Pi = \Gamma \cup S$ be a set of constraints where Γ is a set of non-strict ones and S is the set of strict indicator constraints (23), δ a new variable, and $b > 0$ a bound. The* non-strict translation *of Π with respect to δ and b, denoted Π^b_δ, is:*

$$\begin{array}{ll} \texttt{maximize} & \delta \\ \texttt{subject to} & \Gamma \cup S_\delta \cup \{0 \leq \delta \leq b\} \end{array} \tag{25}$$

where S_δ is defined by (24).

Given Lemma 2 and Definition 3, the satisfiability of a set of constraints can be captured by its non-strict translation as formalized by the following theorem.

Theorem 2. *Let Π, S, and Π^b_δ be defined as in Definition 3. Then, Π is satisfiable iff Π^b_δ has a solution ν such that $\nu(\delta) > 0$.*

Theorem 2 enables the use of current MIP systems for checking the satisfiability of a set of strict constraints, i.e., by computing an optimal solution for the non-strict translation of the set and by checking if the objective function has a positive value.

3.3 Non-Strict Translation of Programs

Next, we develop the non-strict translation of ASP(LC) programs using Definition 3.

Definition 4. *Let P be an ASP(LC) program, δ a new variable, and $b > 0$ a bound. The* non-strict translation *of P with respect to δ and b, is $\tau(P)^b_\delta$ where $\tau(P)$ is the MIP-translation of P.*

We show that the solutions of $\tau(P)^b_\delta$ and $\tau(P)$ are in a tight correspondence.

Lemma 3. *Let P be an ASP(LC) program that may involve real variables, δ a new variable, and $b > 0$ a bound.*

1. *For any solution $\nu \models \tau(P)$, there is a solution $\nu' \models \tau(P)^b_\delta$ such that $\nu(a) = \nu'(a)$ for each $a \in \mathcal{A}(P)$, $\nu \models t$ iff $\nu' \models t$ for each $t \in \mathcal{T}(P)$, and $\nu'(\delta) > 0$.*
2. *For any solution $\nu \models \tau(P)^b_\delta$ where $\nu(\delta) > 0$, there is a solution ν' of $\tau(P)$ such that $\nu(a) = \nu'(a)$ for each $a \in \mathcal{A}(P)$ and $\nu \models t$ iff $\nu' \models t$ for each $t \in \mathcal{T}(P)$.*

Proof. We prove (i) and omit the proof of (ii) which is similar. Let ν be a solution of $\tau(P)$. Given ν, we extend $\tau(P)$ to $\tau'(P)$ by adding for each atom $a \in \mathcal{A}(P)$, a constraint $a = \nu(a)$, and for each theory atom $t \in \mathcal{T}(P)$ and the variable d introduced for t in (6), $d = \nu(d)$. It is clear that $\nu' = \nu$ is a solution of $\tau'(P)$. Let $\tau''(P)$ be the analogous extension of $\tau(P)^b_\delta$. Applying Theorem 2 to $\tau'(P)$, there is a solution ν'' of $\tau''(P)$ such that $\nu''(\delta) > 0$ and for each a, $\nu''(a) = \nu'(a) = \nu(a)$, and for each d, $\nu''(d) = \nu'(d) = \nu(d)$. The valuation ν'' is also a solution of $\tau(P)^b_\delta$, as $\tau(P)^b_\delta \subset \tau''(P)$. Note that for any $t \in \mathcal{T}(P)$ and the respective atom d, $\nu(d) = 1$ iff $\nu \models t$, and $\nu''(d) = 1$ iff $\nu'' \models t$. Then $\nu \models t$ iff $\nu'' \models t$ due to $\nu(d) = \nu''(d)$. □

Now, we relate the solutions of $\tau(P)^b_\delta$ and the answer sets of P. As a consequence of Lemma 3 and the generalization of Theorem 1 for real variables, we obtain:

Theorem 3. *Let P be an ASP(LC) program that may involve real variables, δ a new variable, and $b > 0$ a bound.*

1. *If ν is a solution of $\tau(P)^b_\delta$ such that $\nu(\delta) > 0$, then $I^\nu_P \in AS(P)$.*
2. *If $I \in AS(P)$, then there is a solution ν of $\tau(P)^b_\delta$ such that $I = I^\nu_P$ and $\nu(\delta) > 0$.*

Example 4. Let us revisit Example 3. By setting $b = 1$ as the bound, we obtain the non-strict translation $\tau(P)^1_\delta$ as follows:

$$
\begin{array}{lll}
\texttt{maximize} & \delta & \\
\texttt{subject to} & 0 \leq \delta \leq 1 & \\
& d_1 = 1 \rightarrow h(tom) \geq 8.25, & d_1 = 0 \rightarrow h(tom) + \delta \leq 8.25, \\
& d_2 = 1 \rightarrow h(tom) + \delta \leq 8.25, & d_2 = 0 \rightarrow h(tom) \geq 8.25, \\
& \mathsf{bonus}(tom) - d_1 = 0, & d_2 = 0.
\end{array}
$$

For any optimal solution ν of $\tau(P)^1_\delta$, we have $\nu(\mathsf{bonus}(tom)) = \nu(d) = \nu(\delta) = 1$ and $\nu(h(tom)) \geq 8.25$ that corresponds to the intended answer set $\langle\{\mathsf{bonus}(tom)\}, \{h(tom) \geq 8.25\}\rangle$. We note that CPLEX provides exactly this solution for $\tau(P)^1_\delta$. □

It can be verified that the non-strict translation $\tau(P)^b_\delta$ reduces to the MIP-translation if the variables in P and the new variable δ are integers and the bound b is set to 1.

4 Extension with Objective Functions

In this section, we define optimal answer sets for ASP(LC) programs enhanced by *objective functions* of the form (2) and illustrate the resulting concept by examples.

Definition 5. *Let P be an ASP(LC) program with an objective function f and $\langle M, T\rangle \in AS(P)$. The answer set $\langle M, T\rangle$ is* optimal *iff there is a solution of $T \cup \bar{T}$ that gives the optimal value to f among the set of valuations*

$$\{\nu \mid \nu \models T \cup \bar{T} \text{ for some } \langle M, T\rangle \in AS(P)\}.$$

Example 5. Let P be an ASP(LC) program

$$\texttt{minimize}\ x. \quad a \leftarrow x \geq 5. \quad b \leftarrow x \geq 7. \quad \leftarrow x < 5.$$

The answer sets of P are $I_1 = \langle\{a\}, \{x \geq 5\}\rangle$ and $I_2 = \langle\{a, b\}, \{x \geq 5, x \geq 7\}\rangle$. Let $T_1 = \{x \geq 5\}$ and $T_2 = \{x \geq 5, x \geq 7\}$. The solutions of $T_1 \cup \bar{T}_1 = \{x \geq 5, x < 7\}$ admit a smaller value of $f(x) = x$ (i.e., 5) than any solution of $T_2 \cup \bar{T}_2 = \{x \geq 5, x \geq 7\}$. Therefore the answer set I_1 is optimal. □

According to Definition 5, each optimal answer set identifies an optimal objective value. In other words, if the objective function is unbounded with respect to an answer set, then the answer set is not optimal. This is illustrated by our next example.

Example 6. Let P be an ASP(LC) program

$$\texttt{minimize}\ x. \quad a \leftarrow x \leq 4. \quad b \leftarrow x > 4.$$

The program P has two answer sets $I_1 = \langle\{a\}, \{x \leq 4\}\rangle$ and $I_2 = \langle\{b\}, \{x > 4\}\rangle$. Let $T_1 = \{x \leq 4\}$ and $T_2 = \{x > 4\}$. We have $T_1 \cup \bar{T}_1 = T_1$ and $T_2 \cup \bar{T}_2 = T_2$. Although $T_1 \cup \bar{T}_1$ admits smaller values of x than $T_2 \cup \bar{T}_2$, I_1 is not optimal, since x may become infinitely small subject to $T_1 \cup \bar{T}_1$. Therefore, P has no optimal answer set. □

For a program that does not involve real variables, we can establish an approach to computing the optimal answer sets as stated in the theorem below. This result essentially follows from Definition 5 and Theorem 1.

Theorem 4. *Let P be an ASP(LC) program involving integer variables only and f the objective function of P. Then $\langle M, T\rangle$ is an optimal answer set of P iff there is a solution $\nu \models \tau(P)$ such that $I^{\nu}_{P} = \langle M, T\rangle$ and ν gives the optimal value to f.*

However, when real variables are involved, the non-strict translation from ASP to MIP programs cannot be employed to compute the optimal answer sets, since the variable δ introduced in the translation may affect the optimal objective function value.

The ASP languages implemented in [6, 16] support objective functions of the form

$$\texttt{\#optimize}\ [a_1 = w_{a_1}, ..., a_m = w_{a_m}, \text{not } b_1 = w_{b_1}, ..., \text{not } b_n = w_{b_n}] \quad (26)$$

where the keyword `optimize` is `minimize` or `maximize`, a_i and "not b_i" are literals, and w_{a_i} and w_{b_i} are integer weights associated with the respective literals. The difference between the functions (2) and (26) is that x_i's in the former are integer variables whereas a_i's and b_i's in the latter are Boolean variables, i.e., propositional atoms from the ASP viewpoint. Proper objective functions facilitate modeling optimization problems as one needs not to encode integer variables in terms of Booleans.

Example 7. Let x be an integer variable taking a value from 1 to n and we want to minimize x. Using an objective function, this can be concisely encoded by:

$$\texttt{minimize}\ x. \quad x \geq 1. \quad x \leq n.$$

Following [14, 17], to encode the same using the function (26), we need:

$$\texttt{\#minimize}\ [x(1) = 1, \ldots, x(n) = n]. \tag{27}$$

$$x(i) \leftarrow \text{not } x(1), \ldots, \text{not } x(i-1), \text{not } x(i+1), \ldots \text{not } x(n). \quad 1 \leq i \leq n \tag{28}$$

where the Boolean variable $x(i)$ represents that x takes value i and the rules in (28) encode that x takes exactly one value from 1 to n. The size of the ASP(LC) encoding is constant, while that of the original ASP encoding is quadratic[2] in the size of the domain of x, i.e., the objective function of length n plus n rules of length n. □

Restriction to Boolean variables affects the performance of ASP systems when dealing with problems involving large domains. A detailed account can be found in [11].

5 A Comparison of ASP(LC) with MIP

In the ASP(LC) paradigm, a problem is solved indirectly, i.e., by first modeling it as an ASP program, then by translating the program to a MIP program, and by solving it. A question is how the approach compares with native MIP. In this section, we study the modeling capabilities of ASP(LC) and MIP languages. We focus on two widely used primitives in modeling: *reachability* and *disjunctivity*. For the former, we study a Hamiltonian Routing Problem (HRP) and for the latter, a Job Shop Problem (JSP). The main observation is that ASP(LC) can provide more intuitive and compact encodings in debt to its capability to model non-trivial logical relations. But, the compactness does not always offer computational efficiency as perceived in the sequel.

In the HRP, we have a network and a set of *critical* vertices. The goal is to route a package along a Hamiltonian cycle in the network so that the package reaches each critical vertex within a given vertex-specific time limit. The network is represented by a set of vertices $V = \{1, \ldots, n\}$ and a set of weighted edges E consisting of elements (i, j, d) where $1 \leq i, j \leq n$ and d is a real number representing that the delay of the edge from i to j is d. The set of critical vertices CV consists of pairs (i, t) where $1 < i \leq n$ and t is a real number representing that the time limit of vertex i is t.

The ASP(LC) encoding of HRP in Figure 1 is obtained by extending the encoding of Hamiltonian cycle problem [14] with timing constraints[3]. The rules (29)–(35) specify a Hamiltonian cycle. To model the timing constraints, we use a real variable $t(X)$ to denote the time when a vertex X is reached. Rules (36) and (37) determine the respective times of reaching vertices. The rule (38) ensures each critical vertex to be reached in time. The MIP program of HRP in Figure 2 is from [15] with extensions of timing constraints[4]. The binary variable x_{ij} represents whether an edge (i, j, d) is on the Hamiltonian cycle ($x_{ij} = 1$) or not ($x_{ij} = 0$) and the integer variable p_i denotes the

[2] Linear and logarithmic encodings can be achieved using *cardinality constraints* [6, 16] and *bit vectors* [13], respectively. But both are more complex than the given ASP(LC) encoding.

[3] A rule with the operator "=" in the head, such as (36) and (37), is a shorthand for a pair of rules with the operators "$\leq$" and "$\geq$" in their heads respectively.

[4] Disequalities and implications can be represented using the operators "$\leq$" and "$\geq$" [15].

$$\mathsf{hc}(X,Y) \leftarrow \mathsf{e}(X,Y,D),\ \text{not}\ \mathsf{nhc}(X,Y). \tag{29}$$
$$\mathsf{nhc}(X,Y) \leftarrow \mathsf{e}(X,Y,D_1),\ \mathsf{e}(X,Z,D_2),\ \mathsf{hc}(X,Z),\ Y \neq Z. \tag{30}$$
$$\mathsf{nhc}(X,Y) \leftarrow \mathsf{e}(X,Y,D_1),\ \mathsf{e}(Z,Y,D_2),\ \mathsf{hc}(Z,Y),\ X \neq Z. \tag{31}$$
$$\mathsf{initial}(1). \tag{32}$$
$$\mathsf{reach}(X) \leftarrow \mathsf{reach}(Y),\ \mathsf{hc}(Y,X),\ \text{not}\ \mathsf{initial}(Y),\ \mathsf{e}(Y,X,D). \tag{33}$$
$$\mathsf{reach}(X) \leftarrow \mathsf{hc}(Y,X),\ \mathsf{initial}(Y), \mathsf{e}(Y,X,D). \tag{34}$$
$$\leftarrow \mathsf{v}(X),\ \text{not}\ \mathsf{reach}(X). \tag{35}$$
$$t(1) = 0. \tag{36}$$
$$t(X) - t(Y) = D \leftarrow \mathsf{hc}(Y,X), \mathsf{e}(Y,X,D), X \neq 1. \tag{37}$$
$$t(X) \leq T \leftarrow \mathsf{critical}(X,T). \tag{38}$$

Fig. 1. An ASP(LC) encoding of HRP

$$\textstyle\sum_{(i,j,d)\in E} x_{ij} = 1 \qquad i \in V \tag{39}$$
$$\textstyle\sum_{(j,i,d)\in E} x_{ji} = 1 \qquad i \in V \tag{40}$$
$$1 \leq p_i \leq n \qquad i \in V \tag{41}$$
$$p_i \neq p_j \qquad i \in V,\ j \in V,\ i \neq j \tag{42}$$
$$p_j \neq p_i + 1 \qquad (i,j,d) \notin E,\ i \neq j \tag{43}$$
$$(p_i = n) \rightarrow (p_j \geq 2) \qquad (i,j,d) \notin E,\ i \neq j \tag{44}$$
$$r_1 = 0 \tag{45}$$
$$x_{ij} = 1 \rightarrow r_j - r_i = d_{ij} \qquad (i,j,d) \notin E \tag{46}$$
$$r_i \leq t_i \qquad (i,t) \in CV \tag{47}$$

Fig. 2. A MIP encoding of HRP

position of the vertex i on the cycle. The real variable r_i denotes the time of reaching vertex i. The constraints (39)–(44) encode a Hamiltonian cycle and (45)–(47) are the counterparts of (36)–(38) respectively.

In comparison, reachability is modeled by the recursive rules (33) and (34) in the ASP(LC) program. Since MIP language cannot express such recursion directly, the reachability condition is captured otherwise—by constraining the positions of the nodes in (41)–(44). Note that the node positions are actually irrelevant for the existence of a cycle. In fact, modeling Hamiltonian cycles in non-complete graphs is challenging in MIP and the above encoding is the most compact one to the best of our knowledge.

In the JSP, we have a set of tasks $T = \{1, ..., n\}$ to be executed by a machine. Each task is associated with an earliest starting time and a processing duration. The goal is to schedule the tasks so that each task starts at its earliest starting time or later, the processing of the tasks do not overlap, and all tasks are finished by a given deadline. Using ASP(LC) we model the problem in Figure 3 where the predicate $\mathsf{task}(I, E, D)$ denotes that a task I has an earliest starting time E and a duration D and the real variables $s(I)$ and $e(I)$ denote the starting and ending times of task I, respectively. The

$$s(I) \geq E \leftarrow \mathsf{task}(I, E, D). \tag{48}$$
$$e(I) - s(I) \geq D \leftarrow \mathsf{task}(I, E, D). \tag{49}$$
$$\leftarrow \mathsf{task}(I, E_1, D_1),\ \mathsf{task}(J, E_2, D_2),\ I \neq J,\ s(I) - s(J) \leq 0,\ s(J) - e(I) \leq 0. \tag{50}$$
$$e(I) \leq \mathit{deadline} \leftarrow \mathsf{task}(I, E, D). \tag{51}$$

Fig. 3. An ASP(LC) encoding of JSP

$$s_i \geq est_i \qquad i \in T \tag{52}$$
$$e_i - s_i \geq d_i \qquad i \in T \tag{53}$$
$$x_{ij} = 1 \rightarrow e_i - s_j < 0 \qquad i \in T,\ j \in T,\ i \neq j \tag{54}$$
$$x_{ij} = 0 \rightarrow e_i - s_j \geq 0 \qquad i \in T,\ j \in T,\ i \neq j \tag{55}$$
$$x_{ij} + x_{ji} = 1 \qquad i \in T,\ j \in T,\ i \neq j \tag{56}$$
$$e_i \leq \mathit{deadline} \qquad i \in T \tag{57}$$

Fig. 4. A MIP encoding of JSP

rule (48) says that a task starts at its earliest starting time or later. The rule (49) ensures each task being processed long enough. The rule (50) encodes the mutual exclusion of the tasks, i.e., for any two tasks, one must be finished before the starting of the other. The rule (51) enforces each task being finished by the deadline. Figure 4 adopts a recent MIP encoding of JSP [9] in CPLEX language, where the variables s_i and e_i represent the starting and ending times of task i, respectively; est_i and d_i are the earliest starting time and processing duration of task i; the binary variable x_{ij} denotes that task i ends before task j starts. The constraints (52), (53), and (57) are the counterparts of (48), (49), and (51) respectively. The constraints (54)–(56) exclude overlapping tasks.

In the ASP(LC) program of JSP, the mutual exclusion of $s(I) - s(J) \leq 0$ and $s(J) - e(I) \leq 0$ is expressed by one rule (50). In contrast, MIP language lacks direct encoding of relations between constraints and therefore, to encode the relation of $e_i < s_j$ and $e_j < s_i$, one has to first represent them by new variables x_{ij} and x_{ji} and then encode the relations of the variables (56). Note that, for computation, the ASP(LC) and the native MIP encodings are essentially the same, since the translation of the rule (50) includes two indicator constraints to represent the constraints $s(I) - s(J) \leq 0$ and $s(J) - e(I) \leq 0$, respectively, and additional constraints to encode the rule.

6 Experiments

We implemented the non-strict translation of ASP(LC) programs and the MIP objective functions by modifying the MINGO system [11]. The new system is called MINGO^r. We tested MINGO^r with a number of benchmarks[5]: the HRP and JSP detailed in Section 5,

[5] A prototype implementation of the MINGO^r system and benchmarks can be found under `http://research.ics.aalto.fi/software/asp/mingoR/`

the Newspaper, Routing Max, and Routing Min problems from [11], and the Disjunctive Scheduling problem from [4]. These problems were selected as they involve either reachability or disjunctivity. The original instances are revised to include real numbers. The objective functions of the optimization versions of HRP and JSP are to minimize the time of reaching some critical node and the ending time of some task, respectively. The optimization problems involve integers only. The experiments were run on a Linux cluster having 112 AMD Opteron and 120 Intel Xeon nodes with Linux 6.1. In each run, the memory is limited to 4GB and the cutoff time is set to 600 seconds.

In Table 1, we evaluate MINGO^r with different values of the parameter b, the bound used in the non-strict translation. The goal is to find a default setting for b. Table 1 suggests that $b = 10^{-6}$ is the best, considering the number of solved instances and the running time. We tested 100 random instances for each problem except Disjunctive Scheduling where we used the 10 instances given in [4]. Each instance was run 5 times and the average number of solved instances and running time are reported. We also include the average sizes of resulting *ground* programs in kilobytes to give an idea on the space complexity of the instances. It is also worth pointing out that none of the problems reported in Table 1 is solvable by existing ASP systems since they involve real variables and thus comparisons of MINGO^r with other ASP systems are infeasible.

Tables 2 and 3 provide comparisons of the translation-based ASP approach with native MIP approach, where both MINGO^r and CPLEX are run with default settings. For the HRP problem in Table 2, we tested 50 randomly generated graphs of 30 nodes for each *density* (the ratio of the number of edges to the number of edges in the complete graph). The results show that the instances with medium densities are unsolvable to CPLEX before the cutoff but MINGO^r can solve them in reasonable time. We also note that CPLEX performs better for the graphs of high densities. This is because the MIP program encodes the positions of nonadjacent nodes. For the JSP problem in Table 3, we tested 50 random instances for each number of tasks and MINGO^r is slower than CPLEX by roughly a order of magnitude but, in spite of this, scaling is similar.

In summary, some observations are in order. On one hand, the ASP(LC) language enables compact encodings in debt to its capability of expressing non-trivial logical relations. Thus some redundant information, such as the order of nodes in a cycle in HRP, can be left out in favor of computational performance. On the other hand, the translation of ASP(LC) programs into MIP is fully general and thus some unnecessary extra variables could be introduced. E.g., in the case of JSP, the structure of the translation is more complex than the native MIP encoding. As a consequence, the translation-based approach is likely to be slower due to extra time needed for propagation.

7 Related Work

Dutertre and de Moura [5] translate strict linear constraints into non-strict ones using a new variable δ in analogy to Lemma 1. However, the variable remains unbounded from above in their proposal. In contrast to this, an explicit upper bound is introduced by our translation. The bound facilitates computation: if the translated set of constraints has no solution ν with $\nu(\delta) > 0$ under a particular bound b, this result is conclusive and no

Benchmark	$b = 10^{-9}$		$b = 10^{-6}$		$b = 10^{-3}$		$b = 1$		$b = 10^{3}$		Size
	Solved	Time	Solved	Time	Solved	Time	Solved	Time	Solved	Time	
Disj. Scheduling	10	0.60	10	0.78	10	0.99	10	0.89	10	12.60	206
Ham. Routing	100	30.79	100	24.52	100	23.16	100	41.63	0	NA	155
Job Shop	100	9.76	100	9.56	100	25.98	100	14.61	10	66.08	387
Newspaper	100	22.64	100	21.61	93	77.90	100	40.76	0	NA	846
Routing Max.	100	0.11	100	0.14	100	0.25	100	0.55	100	0.69	7
Routing Min.	76	109.58	77	102.98	80	127.12	46	95.07	20	79.07	368

Table 1. The effect of the bound $b > 0$

Density	Decision		Optimization	
	MINGO^r	CPLEX	MINGO^r	CPLEX
10	0.03	0.01	0.07	0.01
20	0.05	0.01	0.12	0.01
30	0.92	NA	50.81	NA
40	41.62	NA	NA	NA
50	13.94	NA	NA	NA
60	64.91	NA	NA	NA
70	35.78	NA	NA	NA
80	8.02	95.40	NA	NA
90	181.33	24.74	NA	NA
100	146.18	13.88	NA	NA

Table 2. Hamiltonian Routing Problem

Tasks	Decision		Optimization	
	MINGO^r	CPLEX	MINGO^r	CPLEX
10	0.42	0.14	0.35	0.08
20	4.04	0.18	1.56	0.14
30	6.78	0.40	4.69	0.49
40	13.74	0.72	12.18	1.62
50	27.37	1.36	16.15	1.16
60	45.44	1.72	30.82	2.01
70	51.56	1.57	47.85	1.80
80	88.72	2.34	68.99	2.83
90	114.32	2.97	79.28	6.43
100	192.09	4.19	112.09	8.05

Table 3. Job Shop Problem

further computations are needed. Unbounded variables are problematic for typical MIP systems and thus having an upper bound for δ is important.

Given the extension of ASP(LC) programs with objective functions, MIP programs can be seen as a special case of ASP(LC) programs, i.e., for any MIP program P with an objective function (2) and constraints in (3), there is an ASP(LC) program P' whose objective function is (2) and whose rules simply list the constraints in (3) as theory atoms (facts in ASP terminology). Note that in this setting the non-strict translation of P' is identical to P, since P' involves non-strict constraints only.

In theory, all similar paradigms proposed in [1, 7, 12] cover real-valued constraints. Moreover, a recent ASP system CLINGCON [7] is implemented where constraints over integers are allowed in logic programs. But, to the best of our knowledge, there has not been any system that supports real-valued constraints nor integer-based objective functions. With the non-strict translation of ASP programs into mixed integer programs, we are able to implement these primitives in the context of ASP.

8 Conclusion and Future Work

In this paper, we generalize a translation from ASP(LC) programs to MIP programs so that linear constraints over real variables are enabled in answer set programming.

Moreover, we introduce integer objective functions to ASP(LC) language. These results extend the applicability of answer set programming. Finally, we compare the ASP approach with the native MIP approach and the results show that ASP extensions in question facilitate modeling and offer computational advantage at least for some problems. Our results suggest that MIP and ASP paradigms can benefit mutually. Efficient MIP formulations may be obtained by translating a compact ASP(LC) program. On the other hand, ASP language can be extended using MIP constraints and an objective function to deal with problems that are not directly solvable within standard ASP.

The future work will be focused on system development and experiments. E.g., we will study techniques to reduce the number of extra variables needed in translations and to stop the solving phase early (as soon as δ is positive). These goals aim at more efficient implementation of MINGO^{r}which also lacks a user-friendly front-end parser for the moment. Experiments that compare MINGO^{r}with other constraint logic programming systems [10] will also be conducted to provide insights into improving MINGO^r.

References

1. Balduccini, M.: Industrial-size scheduling with ASP+CP. In: LPNMR. (2011) 284–296
2. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Commun. ACM **54**(12) (2011) 92–103
3. Clark, K.L.: Negation as failure. Logics and Databases (1978) 293–322
4. Denecker, M., Vennekens, J., Bond, S., Gebser, M., Truszczynski, M.: The second answer set programming competition. In: LPNMR. (2009) 637–654
5. Dutertre, B., de Moura, L.M.: A fast linear-arithmetic solver for DPLL(T). In: CAV. (2006) 81–94
6. Gebser, M., Kaminski, R., König, A., Schaub, T.: Advances in *gringo* series 3. In: LPNMR. (2011) 345–351
7. Gebser, M., Ostrowski, M., Schaub, T.: Constraint answer set solving. In: ICLP. (2009) 235–249
8. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: ICLP. (1988) 1070–1080
9. Grimes, D., Hebrard, E., Malapert, A.: Closing the open shop: Contradicting conventional wisdom. In: CP. (2009) 400–408
10. Jaffar, J., Maher, M.J.: Constraint logic programming: A survey. Journal of Logic Programming **19/20** (1994) 503–581
11. Liu, G., Janhunen, T., Niemelä, I.: Answer set programming via mixed integer programming. In: KR. (2012) 32–42
12. Mellarkod, V.S., Gelfond, M., Zhang, Y.: Integrating answer set programming and constraint logic programming. AMAI **53**(1-4) (2008) 251–287
13. Nguyen, M., Janhunen, T., Niemelä, I.: Translating answer-set programs into bit-vector logic. CoRR **abs/1108.5837** (2011)
14. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. AMAI **25**(3-4) (1999) 241–273
15. Niemelä, I.: Linear and integer programming modelling and tools. In Lecture Notes for the course on Search Problems and Algorithms, Helsinki University of Technology (2008)
16. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138**(1-2) (2002) 181–234
17. You, J.H., Hou, G.: Arc-consistency + unit propagation = lookahead. In: ICLP. (2004) 314–328