0-1 Integer Programming for Computing Semi-Stable Semantics of Argumentation Frameworks

Mauricio Osorio[1], Juan Díaz[1], Alejandro Santoyo[2]

[1] Universidad de las Américas Puebla, Cholula, Puebla, Mexico

[2] Universidad Autónoma de Querétaro, Querétaro, Mexico

osoriomauri@gmail.com, juana.diaz@udlap.mx, alejandro1.santoyo@gmail.com

Abstract. Dung's abstract argumentation has been an object of intense study not only due to its relationship with logical reasoning but also because of its uses within artificial intelligence. One research branch in abstract argumentation has focused on finding new methods for computing its different semantics. We present a novel method, to the best of our knowledge, for computing semi-stable semantics using 0-1 integer programming. This approach captures the notions of conflict freeness, acceptability, maximality with regard to set inclusion, etc., by 0-1 integer constraints. Additionally, this work also presents an empirical experiment to compare our novel approach with an answer set programming approach. Our results indicate that the new method performed well, and it has a great opportunity space for improving.

Keywords. Argumentation frameworks, binary programming, answer set programming, semi-stable semantics.

1 Introduction

Argumentation theory has become an increasingly important and exciting research topic in Artificial Intelligence (AI), with research activities ranging from developing theoretical models, prototype implementations, and application studies [3], [28]. The main purpose of argumentation theory is to study the fundamental mechanism humans use in argumentation and to explore ways to implement this mechanism on computers.

Argumentation theory is related to debate, dialogue, negotiation, conversation, and persuasion; it searches conclusions through logical reasoning [25]. In fact, argumentation theory has been successfully used in a variety of applications such as organ transplantation [24], democratic decision support [1], multi-agent systems [21], legal reasoning [22] [29], machine learning [23], as well as debate and argument mining [19].

Currently, formal argumentation research has been strongly influenced by abstract argumentation theory of Dung [15]. This approach is mainly orientated to manage the interaction of arguments by introducing a single structure called Argumentation Framework (AF). An AF basically is a pair of sets: a set of arguments and a set of disagreements between arguments called attacks. Indeed, an AF can be regarded as a digraph in which the arguments are represented by nodes and the attack relations are represented by arcs. In Figure 1, one can see an example of an AF and its graph representation.

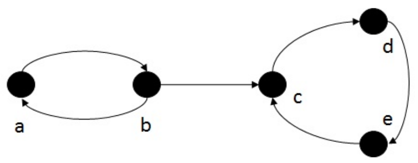

Fig. 1. Graph Rep. $AF := \langle\{a,b,c,d,e\},\{(a,b),(b,a),(b,c),(c,d),(d,e),(e,c)\}\rangle$

In [15], four argumentation semantics were introduced: grounded, preferred, stable, and complete semantics. The central notion of Dung's semantics is the acceptability of the arguments. Even though each of these argumentation semantics represents different patterns of selection of arguments, all of them are based on the basic concept of admissible set. Informally speaking, an admissible set presents a coherent and defendable point of view in a conflict between arguments. For instance, by considering the AF of Figure 1, one can find the following admissible sets: $\emptyset, \{a\}, \{b\}, \{b, d\}$.

One research branch in abstract argumentation has been to find new methods for computing its different semantics, *i.e.* the search for acceptable (*w.r.t.* certain criteria) sets of arguments. Charwat *et al.* [12] survey the approaches that have been used so far for computing AF semantics, and divide them into reduction and direct approaches. The direct approach consists in developing new algorithms for computing AF semantics, and the reduction approach consists in using the software that was originally developed for other formalisms [12].

Our interest is in the reduction approach. Thus, a given AF has to be formalized in the targeted formalism such as constraint-satisfaction [13], constraint-programming [6], propositional logic [4], and answer-set programming [7, 30]. But, our interest is particularly in the answer-set-programming and binary integer programming approaches.

To the best of our knowledge, there is only one previous work [27] where authors indirectly used 0-1 integer programming for computing preferred semantics. Their approach was based on a mapping from an argumentation framework AF into a logic program with negation as failure Π_{AF}. After that, authors computed the Clark's completion $Comp(\Pi_{AF})$ [26], and finally they created the 0-1 integer program $lc(\Pi_{AF})$ [2] which then was solved by a mathematical programming solver.

In this work, we present a novel method, to the best of our knowledge, for directly computing semi-stable (SS) semantics using 0-1 integer programming with no mapping. Our approach is based on 0-1 integer programming *i.e.* a mathematical programming formulation which models the semi-stable semantics. This work presents the mathematical model and explains it in terms of Dung's abstract argumentation notions [15], *i.e.* the mathematical constraints model conflict-freeness, acceptability, and the semi-stable extension definition itself [10]. This work explains the objective function and each constraints' role.

It is worth mentioning that semi-stable semantics has been regarded as an alternative for stable semantics by Caminada *et al.* [10] [9] because of the following:

— Every stable extension is also a semi-stable extension.

— There exists at least one semi-stable extension, while it is possible to have no stable extensions.

— Preferred semantics has also been proposed as an alternative [15], but additional non-stable extensions can be introduced, even in situations where stable extensions already exist.

— Additionally, Caminada and Gabbay [11] also state that every semi-stable extension is also a preferred extension.

Therefore the semi-stable semantics is closer and more adequate for being an alternative for stable semantics.

In this work, we also experimentally compare our approach against an answer set programming (ASP) approach: the ASPARTIX approach [17] using its general encoding for SS semantics[1]. The results we obtained indicate that our new method performed well.

However, we must say that ASPARTIX has been proved against tools from other approaches [5] such as Dung-O-Matic[2] which is based on well-known algorithms (direct approach), and ConArgs2[3] which is based on constraint-programming (reduction approach). The results showed that ASPARTIX outperformed Dung-O-Matic, but ConArg2 outperformed ASPARTIX.

The selection of Clingo in our research is due to the following reasons. Clingo is an answer set solver for (extended) normal and disjunctive logic

[1] http://www.dbai.tuwien.ac.at/research/project/argumentation/systempage/

[2] `http://www.arg.dundee.ac.uk/?page_id=279`

[3] `http://www.dmi.unipg.it/conarg/`

programs. It combines the high-level modeling capacities of ASP with state-of-the-art techniques from the area of Boolean constraint solving. The group that developed Clasp won the 2014 Artificial Intelligence Dissertation Award[4]. Furthermore, the software has several trophies[5].

The paper is organized as follows. Section 2 gives some background on argumentation. Section 3 presents a procedure based on solving a series of 0-1 integer programming problems for computing SS semantics. Section 4 presents the preliminary results as well as the used methodology. Section 5 presents our approach for addressing different decision problems. Finally, Section 6 presents some conclusions and future work.

2 Background

We assume that readers are familiar with the basic notions of answer set programming, otherwise readers can find good introductions in [7], and additionally in [30] one can find a survey of ASP approaches for computing argumentation semantics.

2.1 0-1 Integer Programming [31]

A linear programming problem (LP) is a class of mathematical programming problem, a constrained optimization problem, in which we seek to find a set of values for continuous variables $(x_1, x_2, ..., x_n)$ that maximizes or minimizes a linear objective function z, while satisfying a set of linear constraints (a system of simultaneous linear equations and/or inequalities) [14]. Mathematically, an LP is expressed as follows:

$$max\{cx : Ax \leq b, x \geq 0\},$$

where A is an m by n matrix, c an n-dimensional row vector, b an m-dimensional column vector, and x an n-dimensional column vector of variables or unknowns.

[4]http://www.eccai.org/diss-award/current.shtml

[5]http://potassco.sourceforge.net/trophy.html

If all variables are integer, we have an *Integer Program (IP)* written as

$$\max\left\{cx : Ax \leq b, x \in \mathbb{Z}_+^n\right\}$$

and If all variables are restricted to 0-1 values, we have a 0-1 or *Binary Integer Program (BIP)*:

$$\max\left\{cx : Ax \leq b, x \in \{0,1\}^n\right\}.$$

2.2 Formulation of a 0-1 Integer Program

Translating a problem description into a formulation should be done systematically, and a clear distinction should be made between the data of the problem instance, and the variables (or unknowns) used in the model: (1) Define what appear to be the decision variables, (2) Define a set of constraints which will define the feasible solutions space, (3) Using the decision variables, define the objective function.

If something goes wrong, define an alternative set of decision variables and iterate. Once the formulation is ready, it can be coded in any mathematical programming modeling language such as AMPL.

2.3 Abstract Argumentation

Readers with no background on argumentation semantics can find a gentle introduction in [25]. We will use some concepts of Dung's argumentation approach, the main of them is an Argumentation Framework (AF), which captures the relationships between arguments.

Definition 1. *[15] An AF is a pair $AF := \langle AR, attacks\rangle$, where* AR *is a finite set of arguments, and* attacks *is a binary relation on* AR*, i.e. attacks $\subseteq AR \times AR$.*

Any AF can be regarded as a digraph. For instance, if $AF := \langle\{a, b, c, d, e\}, \{(a, b), (b, a), (b, c), (c, d), (d, e), (e, c)\}\rangle$, then AF is represented as it is shown in Figure 1. We say that *a attacks b* (or b is attacked by a) if $attacks(a, b)$ holds. Similarly, we say that a set S of arguments attacks b (or b is attacked by S) if b is attacked by an argument in S.

Dung defined his argumentation semantics based on the basic concept of *admissible set*, which can be understood in terms of *defense* of arguments and in terms of *conflict-free* sets, as follows:

Definition 2. *[10] Let $AF := \langle AR, attacks \rangle$ be an argumentation framework, $A \in AR$ and $S \subseteq AR$, then:*

1. *A^+ as $\{b \in AR \ : \ a \ attacks \ b\}$ and*
2. *S^+ as $\{b \in AR \ : \ a \ attacks \ b \ for \ some \ a \in S\}$.*
3. *A^- as $\{b \in AR \ : \ b \ attacks \ a\}$ and*
4. *S^- as $\{b \in AR \ : \ b \ attacks \ a \ for \ some \ a \in S\}$.*
5. *S is conflict-free iff $S \cap S^+ = \emptyset$.*
6. *S defends an argument a iff $A^- \subseteq S^+$.*
7. *$F \ : \ 2^{AR} \rightarrow 2^{AR}$ as $F(S) = \{a \in AR \ : \ a \ is \ defended \ by \ S\}$.*

It is possible to define the semantics in terms of admissible sets as follows:

Definition 3. *[10] Let $AF := \langle AR, attacks \rangle$ be an argumentation framework and $S \subseteq AR$ be a conflict-free set of arguments, then:*

1. *S is admissible iff $S \subseteq F(S)$.*
2. *S is a complete extension iff $S = F(S)$.*
3. *S is a preferred extension iff S is a maximal (w.r.t. set inclusion) complete extension.*

The SS semantics is similar to the preferred semantics [10], but instead of maximizing S it is required to maximize $S \cup S^+$, as the following definition states:

Definition 4. *[10] Let $AF := \langle AR, attacks \rangle$ be an argumentation framework and $S \subseteq AR$ be a conflict-free set of arguments, then: S is called a SS extension iff S is a complete extension where $S \cup S^+$ is maximal.*

The semi-stable semantics accepts an equivalent statement, as follows:

Definition 1. *[10] Let $AF := \langle AR, attacks \rangle$ be an argumentation framework and let $S \subseteq AR$. The following statements are equivalent:*

1. *S is a complete extension such that $S \cup S^+$ is maximal (Definition 4).*
2. *S is an admissible set such that $S \cup S^+$ is maximal.*

3 Computing Semi-Stable Semantics by 0-1 Integer Programming

In this section we show the 0-1 integer programming formulation for the SS semantics problem and its encoding in Mosel[6], which is the modeling language of the mathematical solver *Xpress*[7] that we used for this work. This section also explains the objective function and constraints, as well as the iterative process for computing all the extensions of the SS semantics.

3.1 Semi-Stable Semantics Problem Formulation

A mathematical programming solver works with mathematical formulations which in turn work with decision variables. These formulations are coded using the solver's modeling language. In our case, the values that decision variables can take on are restricted to $\{0, 1\}$ since we use 0-1 integer programming.

The task of the solver is to determine the values that the decision variables should take on in order to optimize (maximize or minimize) the objective function of its underlying mathematical model. It is possible to have several feasible solutions, but the solver has to find the optimal one, if it exists. Therefore, the first step is to define the required binary decision variables.

Considering that Definition 4 and Proposition 1 state an SS extension in terms of an *admissible* set S such that $S \cup S^+$ is maximal, then it is required

[6]`http://www.fico.com/en/wp-content/secure_upload/Xpress-Mosel-User-Guide.pdf`

[7]FICO XPRESS: http://www.fico.com/en/products/fico-xpress-optimization-suite

a decision variable for S and another for S^{+} as follows:

$$S_i = \begin{cases} 0 & \text{if } i \notin S \\ 1 & \text{if } i \in S \end{cases} \qquad \forall\, i \in AR. \tag{1}$$

When the solver executes the mathematical model's program and finds an optimal solution, we should interpret the values of the decision variables in terms of a solution of the modeled problem. Thus, in our case, if the variable $S_i = 1$, the associated argument i is in the set S, otherwise it is not. Once we have this solution we define

Definition 5. *The set* $M = \{i \in AR : S_i = 1$ *in the optimal solution*$\}$, *and* $C = \{i \in AR : S_i = 0$ *in the optimal solution*$\}$ *is M's complement.*

Accordingly, we define the decision variable for S^{+} as follows:

$$S_i^{+} = \begin{cases} 0 & \text{if } i \notin S^{+} \\ 1 & \text{if } i \in S^{+} \end{cases} \qquad \forall\, i \in AR. \tag{2}$$

Definition 6. *The set* $M^{+} = \{i \in AR : S_i^{+} = 1$ *in the optimal solution*$\}$, *and* $C^{+} = \{i \in AR : S_i^{+} = 0$ *in the optimal solution*$\}$ *is* M^{+}*' complement.*

In this way, the optimal solution of the 0-1 integer problem formulation for the SS semantics can be stated in terms of maximizing $S \cup S^{+}$. Additionally, it is required to have a decision variable for the union of these sets as follows:

$$U_i = \begin{cases} 0 & \text{if } i \notin S \cup S^{+} \\ 1 & \text{if } i \in S \cup S^{+} \end{cases} \qquad \forall i \in AR. \tag{3}$$

Definition 7. *The set* $Mu = \{i \in AR : U_i = 1$ *in the optimal solution*$\}$, *and* $Cu = \{i \in AR : U_i = 0$ *in the optimal solution*$\}$ *is Mu's complement.*

Now, in order to have a mechanism to work with attacks more suitable than working with the adjacency matrix of a given AF, we define the following:

Definition 8. *Let* $AF := \langle AR, attacks \rangle$ *be an argumentation framework, then* $R_i^{-} = \{j \in AR : (j,i) \in attacks\}$ $\forall i \in AR$, *is the set of nodes attacking node* i, *and the set of sets* $R^{-} = \{R_i^{-} : i \in AR\}$.

Considering Definitions 2 and 3, we restate the admissible set definition in terms of Definition 8 in order to be able to derive the linear constraint that assures admissibility, thus we have the following Lemma:

Lemma 1. Let $AF := \langle AR, attacks \rangle$ be an argumentation framework, and set $S \subseteq AR$, then S is admissible iff $\forall i \in S, \forall j \in R_i^{-}, \exists k \in S,\ ((k,j) \in attacks))$.

It is important to take into account that a mathematical solver searches in the solution space for just one optimal solution for a given problem formulation, *i.e.* just one extension. Therefore, if we want all the extensions of a given AF, it is required to solve a series of binary programming models, one for each extension. Therefore, we can think of the SS semantics problem formulation in terms of a series of solutions that the solver finds in a series of iteration.

Since SS semantics is made up of several sets, we use M^{t} to denote the solution of the binary subproblem in iteration t and q to denote the amount of SS extensions that a given argumentation framework has such that $q \geq 1$, thus:

$$\{Mu^{1}, Mu^{2}, ..., Mu^{q}\} : \\ |Mu^{1}| \geq |Mu^{2}| \geq ... \geq |Mu^{q}|. \tag{4}$$

This expression states that the Mu^{1} has the largest possible cardinality, and that Mu^{q} has the smallest possible cardinality. Therefore, it is clear that in order to get all SS extensions it is required to iterate q times, and the problem formulation will use t to denote a given iteration.

Additionally, it is also required to have a decision variable for a couple of either/or constraints [20] which will be added per each iteration (see Section 3.3) in order to assure $S \cup S^{+}$ maximality *w.r.t.* set inclusion, *i.e.* they help us just to avoid that solution Mu^{t+1} be the same than the solutions found in iterations $t, t-1, \ldots, 1$ or any subset of them as follows:

$$Y_r = \begin{cases} 1 & \text{if } |Mu^{t}| > |Mu|^{t+1} \\ 0 & \text{otherwise} \end{cases} \qquad r = 1, \ldots, t-1. \tag{5}$$

This way, the following 0-1 integer problem formulation to compute the t^{th} semi-stable extension of a given AF is the following (including (1), (2), (3), (5)):

$$max\ f(S) = \sum_{i \in AR} (S_i + S_i^+), \tag{6}$$

subject to:

$$S_i + S_j \leq 1, \forall i \in AR, \forall j \in R_i^-, \tag{7}$$

$$\sum_{k \in R_j^-} S_k \geq S_i, \forall i \in AR, \forall j \in R_i^-, \tag{8}$$

$$S_i^+ \geq S_j, \forall i \in AR, \forall j \in R_i^-, \tag{9}$$

$$S_i^+ \leq \sum_{j \in R_i^-} S_j, \forall i \in AR, \tag{10}$$

$$U_i = S_i + S_i^+, \forall i \in AR, \tag{11}$$

$$\sum_{i \in C^r} S_i \geq 1, \forall r = 1, \ldots, t-1, \tag{12}$$

$$-(\sum_{i \in Mu^r} U_i) + |Mu^r| \leq |AR| * Y_r, \forall r = 1, \ldots, t-1, \tag{13}$$

$$-(\sum_{i \in Cu^r} U_i) + 1 \leq |AR| * (1 - Y_r), \forall r = 1, \ldots, t-1, \tag{14}$$

$$S_i \in \{0, 1\}, i = 1, ..., |AR|, \tag{15}$$

$$S_i^+ \in \{0, 1\}, i = 1, ..., |AR|, \tag{16}$$

$$U_i \in \{0, 1\}, i = 1, ..., |AR|, \tag{17}$$

$$Y_i \in \{0, 1\}, i = 1, ..., t-1. \tag{18}$$

Note that (1), (2), (3) and (5) are the decision variables definition, and constraints (15), (16), (17), (18) define the problem's domain.

Note also that constraints (12), (13), and (14) are going to be added in iteration t, once the model was solved, in order to be active in iteration $t+1$, while the remaining constraints are always active. The following paragraph explains the objective of these constraints, but they are explained with more detail in Section 3.3.

Now, let us explain each constraint within the context of Dung's abstract argumentation notions and the semi-stable definition:

1. ***Maximality with regard to set inclusion.*** The model's objective function (OF)(6) guarantees us that we will find a *maximum cardinality set*, which will be the solution Mu^t. This set is made up of $S \cup S^+$. Constraints (12), (13), and (14) along with the objective function will avoid that M^{t+1} and Mu^{t+1} be any subset of M^t and Mu^t respectively, thus they guarantee us *maximality with regard to set inclusion*. Subsection 3.3 explains how constraints (13) and (14) were defined and why they are added after the computation of each additional extension in order to compute the whole semantics.

2. ***Conflict-Freeness.*** Note that the definition of a conflict-free set in Definition 2 item 5 is not stated in terms of attacks's directions but just in terms of attacks between arguments, without considering the directions of them. In this way, such a definition considers an arc just as an edge, and therefore the whole AF can be regarded as an undirected graph, at least with regard to the conflict-free set problem.

   Note that the expression $S_i + S_j \leq 1$ in constraint (7) will be fulfilled only when $S_i = 1$ or $S_j = 1$ but not both and when $S_i = 0$ and $S_j = 0$, therefore at most one argument will be selected. Thus, this constraint guarantees us that solution will be a *conflict-free set*.

3. ***Admissibility.*** The intuition of Definitions 1, 2, and 3 is that an *admissible set S* should defend each of its arguments, and Lemma 1 only restates it in terms of Definition 8. Note that in Lemma 1 the existential quantifier suggests that constraint (8) should be $\sum_{k \in R_j^-} S_k \geq 1$, but we used $\sum_{k \in R_j^-} S_k \geq S_i$ since the constraint must be fulfilled $\forall i \in AR$ [8]. This way, the translation from this definition to constraint (8) is a straightforward task, this constraint guarantees us that the set M^t is admissible.

[8]There are two special cases: $S_i = 0$ and $S_i = 1$. In the first one, the total of $\sum_{k \in R_j^-} S_k$ does not matter because $S_i \notin Solution$, thus in the second case we will have $\sum_{k \in R_j^-} S_k \geq 1$ which means that there will be at least one argument defending argument i since $S_i \in Solution$.

4. ***Creation of** S^+**.*** So far, we have a conflict-free and admissible set, and it is still required to build S_i^+ as in Definition 2 item 2 $\forall i \in M^+$. It should be done by decision variables (2) such that the objective function can be executed. This way, S_i^+ should take on value 1 if argument i is attacked by some argument $j \in R_i^-$ such that $S_j = 1$. This is the same that $d = d_i \vee d_2 \ldots \vee d_n$ as a logical expression which can be linearized as follows [20]:

$$d \geq d_i \qquad i = 1, \ldots, n \qquad (19)$$

$$d \leq \sum_i d_i \qquad i = 1, \ldots, n \qquad (20)$$

   Note that (19) and (20) become (9) and (10) respectively. Additionally to (19) and (20) we should have $d \leq 1$, but it is redundant due to (16). Thus, constraints (9), (10), and (16) will determine the values of decision variables (2) from values on decision variables (1).

   Thus, M^1 is a conflict-free and admissible set, considering that (6) guarantees a *maximum cardinality set*, and according to Definition 4 and Preposition 1, M^1 is an SS extension.

5. ***Construction of** $U = S \cup S^+$**.*** In order to avoid that Mu^{t+1} be a subset of Mu^t, it is required to have as decision variables the union of (1) and (2), as defined in (3). To this end, and in order to ease this process, consider that it is not possible that $S_i = 1$ and $S_i^+ = 1$ at the same time, since it would mean that M is not a conflict-free set. Thus, the value that U_i will take on should be $S_i + S_i^+$. Considering Definition 7, constraint (11) guarantees that Mu will have the whole solution.

3.2 Semi-Stable Extensions Program

Notice that the problem formulation was made up of (1)-(11), and (15)-(18) already can be used for computing the first SS extension of a given AF. To this end, this program should be coded using a mathematical programming language like *Mosel*[9], this is a straightforward task, since the mathematical language was developed for expressing mathematical formulas. The following code stands for the whole mathematical model:

```
z:= sum(i in arguments) (S(i) + Sp(i))
forall(i in arguments, j in R(i))
                     S(i) + S(j) <= 1
forall(i in arguments, j in R(i))
           sum(k in R(j)) S(k) >= S(i)
forall(i in arguments, j in R(i))
                         Sp(i) >= S(j)
forall(i in arguments)
          Sp(i) <= sum(j in R(i)) S(j)
forall(i in arguments)
                   U(i) = S(i) + Sp(i)
forall(i in arguments) S(i)  is_binary
forall(i in arguments) Sp(i) is_binary
forall(i in arguments) U(i)  is_binary
forall(i in t-1)   Y(i)  is_binary
maximize(z).
```

We will denote this program as BIP in order to make reference to it.

3.3 Semi-Stable Semantics

As it was stated, once the model is implemented in a mathematical programming language, the program computes only one SS extension. In order to compute an additional extension it is required to iterate, but adding additional constraints to avoid getting previous solutions, these constraints will force to get another different extension. In this setting, it is required to iterate to find all the extensions of a given AF until no feasible solution exists. Thus, we have to take care of getting no subsets of M^t (Case No. 1), and no proper subsets of Mu^t (Case No. 2).

Case No. 1. Then, in order to find the constraint that we have to add to avoid that $M^{t+1} \subseteq M^t$, consider Definitions 5, and let P be the solution in iteration $t+1$, and M the solution in iteration t, thus:

$$P \subseteq M \leftrightarrow \forall x (x \in P \rightarrow x \in M)$$
$$\leftrightarrow \forall x (x \notin P \vee x \in M), \qquad (21)$$

[9]http://www.fico.com/en/products/fico-xpress-optimization-suite/

$$P \nsubseteq M \leftrightarrow \exists x(x \in P \wedge x \notin M)$$
$$\leftrightarrow \exists x(x \in P \wedge x \in C). \quad (22)$$

The intuition of this result is that it is required that the solution in iteration $t+1$ has at least one element from solution's complement in iteration t. Constraint (12) is defined from this intuition. Note that the Mosel code must take care of the special case where $|M| = |AR|$.

Case No. 2. Additionally, we have to add another constraint to avoid that $Mu^{t+1} \subset Mu^t$. In order to find such a constraint(s), consider Definition 7 and let P be the solution in iteration $t+1$ and Mu the solution in iteration t, thus:

$$P \subset Mu$$
$$\leftrightarrow \forall x(x \in P \rightarrow x \in Mu) \wedge \exists x(x \notin P \wedge x \in Mu) \quad (23)$$

$$\leftrightarrow \forall x(x \notin P \vee x \in Mu) \wedge \exists x(x \notin P \wedge x \in Mu), \quad (24)$$

$$P \not\subset Mu$$
$$\leftrightarrow \exists x(x \in P \wedge x \notin Mu) \vee \forall x(x \in P \vee x \notin Mu) \quad (25)$$

$$\leftrightarrow \exists x(x \in P \wedge x \in Cu) \vee \forall x(x \in P \vee x \notin Mu). \quad (26)$$

Note that the intuition of the expression in (22) should be the same as that of the first part of the disjunction in (26). Consider also that we wanted to find an expression that led us to a linearized constraint to avoid getting proper subsets of previous solutions. Thus, we can have a proper subset when $|P| < |Mu|$ holds, and therefore we must apply the expression $\exists x(x \in P \wedge x \in Cu)$, otherwise apply the expression $\forall x(x \in P \vee x \notin Mu)$, whose intuition is that the new solution P can have any element, this means that it requires no constraint. Thus, we have just to work with the first part of the disjunction. Therefore, we have to linearize the expression

$$if\ |P| < |Mu|\ then\ \exists x(x \in P \wedge x \in Cu)$$
$$\leftrightarrow if\ |P| < |Mu|\ then \sum_{i \in Cu} U_i \geq 1,$$

as follows [14]:

$$if\ |P| < |Mu|\ then \sum_{i \in Cu} U_i \geq 1$$
$$\leftrightarrow not\ (|P| < |Mu|)\ \vee \sum_{i \in Cu} U_i \geq 1,$$

that in turn can be transformed as follows:

$$\leftrightarrow |P| \geq |Mu| \vee \sum_{i \in Cu} U_i \geq 1$$
$$\leftrightarrow \sum_{i \in Mu^r} U_i \geq |Mu^r| \vee \sum_{i \in Cu^r} U_i \geq 1.$$

This means that only either $\sum_{i \in Mu^r} U_i \geq |Mu^r|$ or $\sum_{i \in Cu} U_i \geq 1$ will be active, but to satisfy the simultaneousness assumption of binary integer programming, they must be transformed considering the following general format [20]:

$$f(x_1, x_2, \ldots, x_n) \leq By,$$
$$g(x_1, x_2, \ldots, x_n) \leq B(1-y).$$

where B is a big number, in our case $B = |AR|$, and y is the binary variables defined in (18). This transformation becomes constraints (13) and (14).

Now, let SSE be a set of all the SS extensions of a given AF, and MC a set of additional (12), (13), and (14) constraints, then the algorithm for computing the q extensions of a given AF is Algorithm 1:

```
1  [ht] Set $SSE = \emptyset$, $MC = \emptyset$;
2  Solve $BIP \cup MC$;
3  **while** *optimal solution found* **do**
4      Let $M, M^+, and\ Mu$ be the optimal solution, and $C, C^+, and\ Cu$ its complements respectively;
5      Add M to SSE;
6      Add $\sum_{i \in C} S_i \geq 1$ to $MC$;
7      Add $-(\sum_{i \in Mu^r} U_i) + |Mu^r| \leq |AR| * Y(r) \forall r = 1, \ldots, t-1$ to $MC$;
8      Add $-(\sum_{i \in Cu^r} U_i) + 1 \leq |AR| * (1 - Y(r)) \forall r = 1, \ldots, t-1$ to $MC$;
9      Solve $BIP \cup MC$;
10 **end**
```

Algorithm 1: Computing all semi-stable extensions of a given AF

Now it is possible to state the following theorem:

Theorem 1. Let AF be an argumentation framework, R^- as in Definition 8, $M, M^+, and\ Mu$ is a solution of BIP, and SSE is computed as described in Algorithm 1, then SSE is the set of all SS extensions of AF.

Proof. Sketch: From the above discussion consider the following items:

1. By constraints (7), (8) we know that M is a conflict-free and admissible set. See Section 3.1 item 2 and 3.
2. By constraints (9) and (10) we know that M^+ is made up from M. See Section 3.1 item 4.
3. By constraint (11) we know that $Mu = M \cup M+$. See Section 3.1 item 5.
4. By the Objective Function (6) we know that $M \cup M^+$ is a maximum cardinality set. See Section 3.1 item 1.
5. By Definition 4 and Proposition 1 we know that if a set M is a conflict-free and admissible set and $M \cup M^+$ is maximal, then M is an SS extension.
6. Now, notice that in each iteration, due to the constraints added to MC in steps 6, 7, and 8 in iteration t, the solution Mu (if exists) obtained in step 4 must not be a superset or subset of any previous solutions already in SSE, and Mu must be of maximum cardinality among the solutions that satisfy $BIP \cup MC$, therefore Mu is maximal *w.r.t.* set inclusion.
7. By previous items and according with (4), there is no any possible solution between solutions found in iteration t and $t+1$, therefore SSE is the set of all SS extensions of AF.

4 Preliminary Results

In order to measure the performance of the 0-1 integer programming approach, it was compared with an ASP approach: the ASPARTIX [18] which we will refer to as ASP1. Additionally, the approach based on 0-1 integer programming will be called BIP (Binary Integer Programming).

4.1 Experiment Description

For the readers interested in the code used to compute the SS extensions, it is available at ASPARTIX's web page[10]. It is worth mentioning that ASPARTIX is the *de facto* benchmark for argumentation systems.

The solver used by the ASP approach was Clingo[11] due to its great performance in several ASP competitions [8], while the 0-1 integer programming approach used the *ad-hoc* Xpress[12] solver. Both solvers were used without any special configuration parameter. The computers used in the experiment had the following configuration: An AMD Phenom II X3 2.80 Ghz processor, 4GB of RAM, and 32-bit Windows 7 professional operating system.

The given time for solving each instance was 1000 seconds. In order to compute the global time when a solver fails solving a given instances, the time assigned is 1000 seconds.

The instances that were used during all the experiments were taken from the ASPARTIX web page[13]. The name of each instance gives us some information about its inherent difficulty to be solved and it has the form inst_G_n_p1_p2_i, that should be interpreted as follows [16]:

— **G:** Generator used: 2: arbitrary AFs (does not use p2); 4: 4-grid AFs; 8: 8-grid AFs.

— **n:** Number of arguments

— **p1:** Probability for each pair of arguments (a,b) that the attack (a,b) is present for the arbitrary graphs. For the grid graphs this parameter indicates one dimension of the grid. The other is calculated with n.

— **p2:** Present only for grid AFs. It indicates the probability that a given attack is a mutual attack.

[10]http://www.dbai.tuwien.ac.at/research/project/argumentation/systempage/

[11]`http://potassco.sourceforge.net`

[12]http://www.fico.com/en/Products/DMTools/Pages/FICO-Xpress-Optimization-Suite.aspx

[13]http://www.dbai.tuwien.ac.at/research/project/argumentation/systempage

— **i:** Index for AFs with the same parameters, i.e. AFs generated with the same parameters are distinguished with this index.

ASPARTIX project's goal was to find "new methods for analyzing, comparing, and solving argumentation problems"[14], and with this goal in mind the project's team created the instances that we used.

4.2 Performance Results

A total of 1,118 instances were used, the instances ranged from 20 arguments to 100 arguments with increments of 10 arguments. There were 132 instances of 20 arguments, 44 of each kind of instances, and so on for the rest of the instances.

In Figure 2 we present average computation times for each approach and for each kind of instances (arbitrary, 4-grid, and 8-grid). Figure 2 shows also that ASP1 had the best performance (near zero) for arbitrary instances, while the Binary Integer Program (BIP) was very close to it for the same kind of instances.

The performance of both solvers solving 4-grid and 8-grid instances were close to zero until 60-arguments instances, where the BIP approach started to have difficulties to solve them. The ASP1 approach started to have problems until the 70-arguments instances for 4-grid instances, and 80-arguments instances for 8-grid instances. Both solvers had no problem solving arbitrary instances, even with the 100-argument ones.

Figure 3 shows timeouts per approach and per kind of instance. Note that there are no timeouts for arbitrary instances and that the behavior is similar to that of average execution times. Note also that though BIP's timeouts started at 60 arguments instances and ASP's at 70 arguments, the rate of growth is almost parallel from 90 arguments instances.

Figures 2 and 3 seem to show that the rate of growth of both solvers is similar with a constant difference between the performance of them, and though we have observed that the execution time for BIP approach for arbitrary instances at n arguments $t(n)_{BIP,2}$ is never larger than 2.691 times the time of the ASP approach $t(n)_{ASP,2}$ (*i.e.* $t(n)_{BIP,2} \leq 2.691 * t(n)_{ASP,2}$), Figure 4 suggests that at some point both lines are going to cross. With regard to the other instances, it is difficult to say if there is also a constant difference since timeout actually indicate a lack of reliable times to determine if this constant exists or not.

[14]The goal is stated at the ASPERTIX project web page.

On the other hand, Figure 5 shows average execute times for finding just the first extension for arbitrary instances, note that the ASP solver was faster than the BIP solver, while Figure 6 shows that for 4-Grid and 8-Grid instances it was the other way around, *i.e.* the BIP solver had a better performance.

It is important to say that the average amount of extensions corresponding to arbitrary instances (2-Grid) is very small, while the average amount of extensions for 4-Grid and 8-Grid instances is larger. Note the great similarity of Figures 4 and 5 which also suggest that the BIP performance is as good as the ASP' performance, but there is a difference in how solvers compute the remaining extensions: the ASP solver keeps enough information to ease the search of the remaining extensions, while the BIP solver starts with no information from previous searches.

At this moment, we can say that this problem could be addressed either by programming the whole solution using *ad-hoc* libraries or by configuring the solver for keeping information of previous searches and adding additional *Mosel* code.

5 Addressing Decision Problems

In order to have a more complete view of our approach to calculate the semi-stable semantics, we decided to analyze its behavior with the decision problems that Caminada *et al.* [10] described in his paper. Such a description is as follows.

Given an argumentation framework $\mathcal{H} = (AR, attacks)$, let $\mathcal{E}$ be its SS semantics. Table 1 describes a number of general decision problems relative to $\mathcal{E}$.

Let $\mathcal{H}$ be the argumentation framework shown in Figure 1, which has $\{b, d\}$ as the only SS extension. Let $S_1, ... S_5$ be the set of decision variables as in

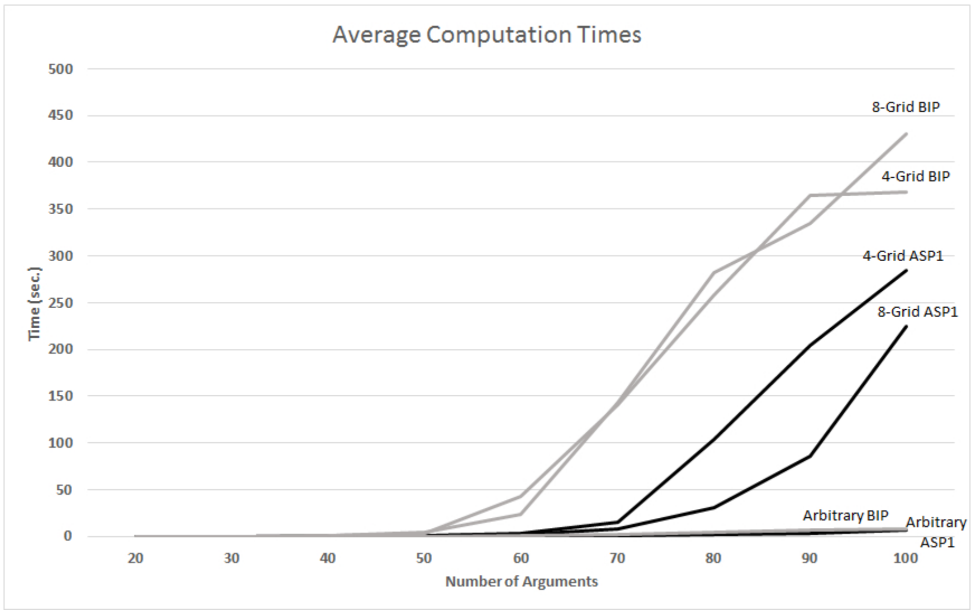

Fig. 2. Average computation times per approach and per kind of AF

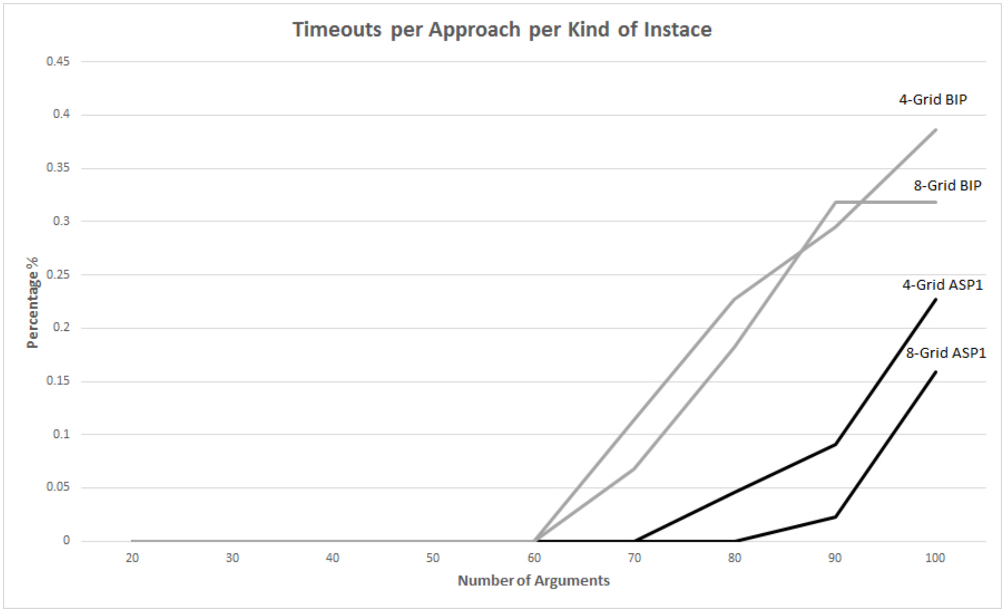

Fig. 3. Timeouts per approach and per kind of AF

Definition 1, such that the set $\{b, d\}$ corresponds to the set $S = \{0, 1, 0, 1, 0\}$.

1. ***Verification.*** In order to decide if $\{b, d\} \in \mathcal{E}(\mathcal{H})$ just add the following constrains:

$$S_1 = 0;\ S_2 = 1;\ S_3 = 0;\ S_4 = 1;\ and\ S_5 = 0.$$

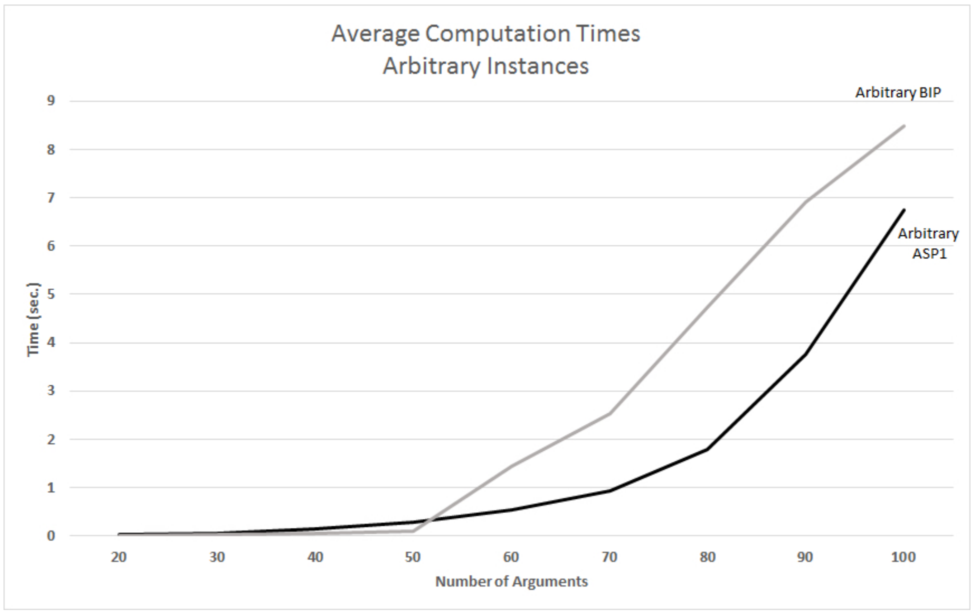

Fig. 4. Average computation times for arbitrary instances (2-grid)

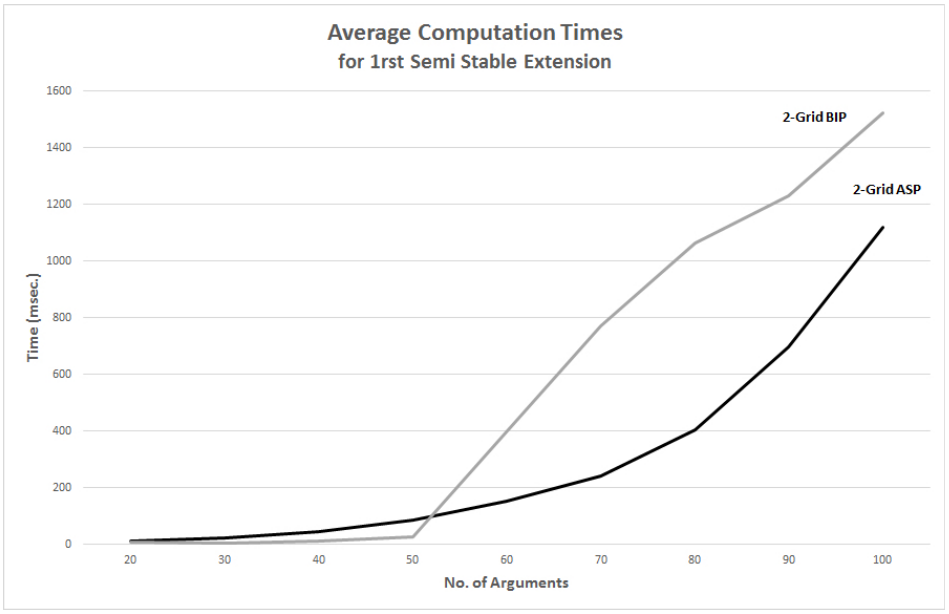

Fig. 5. Average computation times for finding the 1rst extension for arbitrary instances (2-grid)

In this case, if the solver finds an optimal solution, then $\{b, d\}$ is an SS extension, otherwise it is not. Algorithm 1 is not required since we are asking for the first extension, *i.e.* the one the solver finds with no iteration.

2. ***Credulus Acceptance.*** Let $x = b$. In order to decide if there is any $S \in \mathcal{E}(\mathcal{H})$ for which

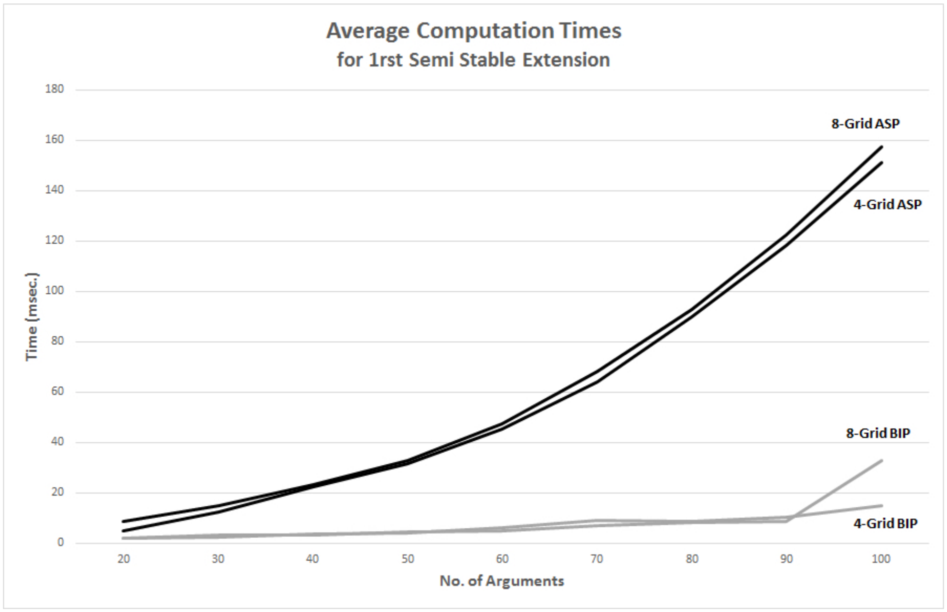

Fig. 6. Average computation times for finding the 1rst extension for 4-grid and 8-grid instances

Table 1. Decision problems described in [10]

Problem Name	Instance	Question
Verification $(VER_{\mathcal{E}})$	$\mathcal{H} = (AR, attacks)$ $S \subseteq AR$	Is $S \in \mathcal{E}(\mathcal{H})$?
Credulus Acceptance $(CA_{\mathcal{E}})$	$\mathcal{H} = (AR, attacks)$ $x \in AR$	Is there any $S \in \mathcal{E}(\mathcal{H})$ for wich $x \in S$?
Sceptical Acceptance $(SA_{\mathcal{E}})$	$\mathcal{H} = (AR, attacks)$ $x \in AR$	Is x member of every $S \in \mathcal{E}(\mathcal{H})$?
Non-emptiness $(EXISTS_{\mathcal{E}}^{\neg\emptyset})$	$\mathcal{H} = (AR, attacks)$	Is there any $S \in \mathcal{E}(\mathcal{H})$ for wich $S \neq \emptyset$?

$x \in S$, add the following constraint:

$$S_2 = 1.$$

In this case, if the solver finds an optimal solution, then the condition we are asking for holds, otherwise it is not. Again, Algorithm 1 is not required.

3. ***Sceptical Acceptance.*** In this case it is not required to add more constraints, but it is required to use Algorithm 1.
Additionally, it is required to add code to the algorithm in order to test if $x \in S$. On the other hand, it is not required to compute all SS extensions of $\mathcal{H}$, since it is enough to find the first extension where $x \notin S$ holds in order to stop.
4. ***Non-emptyness.*** It is not required to add more constraints neither Algorithm 1 is required. It is enough to attempt to compute the first extension and test if the solver found an optimal solution.

6 Conclusions

We have presented a novel method for computing SS semantics using binary integer programming. The performance of the new method was good although the ASP approach outperformed it at finding all the SS extensions of a given argumentation framework.

However, the BIP approach had a performance that was as good as that of the ASP approach at finding just the first extension. It means that our approach is good for solving decision problems, at least in those cases where it is enough to compute the first extension.

The reason for the difference between both approaches for computing all extensions of the SS semantics lies in their designs. The mathematical solver was designed to efficiently compute only the first optimal solution, while the ASP solver was designed to compute all models of a logic program. Despite this difference, it is possible that a mathematical solver can take advantage of the information used for computing the first optimal solution in order to compute the remaining ones more efficiently.

It is well known that binary integer programs can be improved in order to compute more efficiently its objective function by using mathematical programming techniques such as relaxation or adding strong valid inequalities. Therefore, there is a great opportunity space for improving this approach for computing the semi-stable semantics.

This new approach constitutes an alternative for computing AF semantics using mathematical programming, and even though we used a state of the art mathematical programming solver, there exist several libraries for Java, C++, and other general purpose languages.

References

1. **Atkinson, K. (2006).** Value-based argumentation for democratic decision support. *Proceedings of the 2006 Conference on Computational Models of Argument: Proceedings of COMMA 2006*, IOS Press, Amsterdam, The Netherlands, The Netherlands, pp. 47–58.
2. **Bell, C., Nerode, A., Ng, R. T., & Subrahmanian, V. S. (1994).** Mixed integer programming methods for computing nonmonotonic deductive databases. *Journal of the ACM*, Vol. 41, No. 6, pp. 1178–1215.
3. **Bench-Capon, T. & Dunne, P. E. (2007).** Argumentation in artificial intelligence. *Artificial Intelligence*, Vol. 171, No. 10-15, pp. 619 – 641. Argumentation in Artificial Intelligence.
4. **Biere, A., Biere, A., Heule, M., van Maaren, H., & Walsh, T. (2009).** *Handbook of Satisfiability: Volume 185 Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, The Netherlands, The Netherlands.
5. **Bistarelli, S., Rossi, F., & Santini, F. (2014).** A fisrt comparison of abstract argumentation reasoning tools. *21st European Conference on Artificial Intelligence*, pp. 969–979.
6. **Bistarelli, S. & Santini, F. (2011).** Conarg: A constraint-based computational framework for argumentation systems. *IEEE 23rd International Conference on Tools with Artificial Intelligence, ICTAI'11*, IEEE Computer Society, Eashington, DC, USA, pp. 605–612.
7. **Brewka, G., Eiter, T., & Truszczynski, M. (2011).** Answer set programming at a glance. *Commun. ACM*, Vol. 54, No. 12, pp. 92–103.
8. **Calimeri, F., Ianni, G., & Ricca, F. (2012).** The third open Answer Set Programming competition. *CoRR*, Vol. abs/1206.3.
9. **Caminada, M. (2006).** Semi-stable semantics. *Computational Models of Argument: Proceedings of COMMA 2006, September 11-12, 2006, Liverpool, UK*, pp. 121–130.
10. **Caminada, M. W. A., Carnielli, W. A., & Dunne, P. E. (2012).** Semi-stable semantics. *J. Log. Comput.*, Vol. 22, No. 5, pp. 1207–1254.
11. **Caminada, M. W. A. & Gabbay, D. M. (2009).** A logical account of formal argumentation. *Studia Logica*, Vol. 93, No. 2-3, pp. 109–145.
12. **Charwat, G., Dvořák, W., Gaggl, S. A., Wallner, J. P., & Woltran, S. (2015).** Methods for solving reasoning problems in abstract argumentation – a survey. *Artificial Intelligence*, Vol. 220, pp. 28 – 63.
13. **Dechter, R. (2003).** *Constraint Processing*. Morgan Kaufmann Publishers Inc.
14. **Der-San Chen, Y. D., Robert G. Batson (2010).** *Applied Integer Programming: Modeling and Solutions*. John Wiley & Sons, Inc., Hoboken, New Jersey, USA.
15. **Dung, P. M. (1995).** On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, Vol. 77, No. 2, pp. 321–358.

16. **Dvorák, W., Gaggl, S. A., Wallner, J. P., & Woltran, S. (2011).** Making use of advances in answer-set programming for abstract argumentation systems. *CoRR*, Vol. abs/1108.4942.

17. **Egly, U., Alice Gaggl, S., & Woltran, S. (2010).** Answer-set programming encodings for argumentation frameworks. *Argument & Computation*, Vol. 1, No. 2, pp. 147–177.

18. **Egly, U., Gaggl, S. A., & Woltran, S. (2008).** Aspartix: Implementing argumentation frameworks using answer-set programming. **de la Banda, M. G. & Pontelli, E.**, editors, *International Conference of Logic Programming (ICLP)*, volume 5366 of *Lecture Notes of Computer Science*, Springer, pp. 734–738.

19. **Evripidou, V., Toni, F., & Carstens, L. (2014).** From argumentation theory to real life debating: Building social web applications. *Arguing on the Web 2.0, Amsterdam, June 30 - July 1, 2014*.

20. **FICO (2009).** *MIP Formulations and Linearizations*. Fair Isaac Corporation.

21. **Kraus, S., Sycara, K., & Evenchik, A. (1998).** Reaching agreements through argumentation: a logical model and implementation. *Artificial Intelligence*, Vol. 104, No. 1–2, pp. 1 – 69.

22. **McCarty, L. T. (1995).** An implementation of eisner v. macomber. *Proceedings of the 5th International Conference on Artificial Intelligence and Law*, ICAIL '95, ACM, New York, NY, USA, pp. 276–286.

23. **Mozina, M., Zabkar, J., & Bratko, I. (2007).** Argument based machine learning. *Artificial Intelligence*, Vol. 171, No. 10-15, pp. 922–937.

24. **Nieves, J. C., Cortés, U., & Osorio, M. (2006).** Supporting decision making in organ transplanting using argumentation theory. *Latin-American Workshop on Non-Monotonic Reasoning, Proceedings of the LA-NMR06 Workshop, Facultad de Ingeniería de la Universidad Auónoma de San Luis Potí, San Luis Potosí, México, September 18, 2006*.

25. **Nieves, J. C., Osorio, M., & Cortés, U. (2008).** An overview of argumentation semantics. *Computación y Sistemas*, Vol. 12, No. 1.

26. **Nieves, J. C., Osorio, M., & Cortés, U. (2008).** Preferred Extensions as Stable Models. *Theory and Practice of Logic Programming*, Vol. 8, No. 4, pp. 527–543.

27. **Osorio, M. & Santoyo, A. (2013).** Preferred extensions as minimal models of clark's completion semantics. *Research in Computing*, Vol. 68, pp. 57–68.

28. **Rahwan, I. & Simari, G. R.**, editors **(2009).** *Argumentation in Artificial Intelligence*. Springer.

29. **Rissland, E. L., Skalak, D. B., & Friedman, M. T. (1993).** Bankxx: A program to generate argument through case-base research. *Proceedings of the 4th International Conference on Artificial Intelligence and Law*, ICAIL '93, ACM, New York, NY, USA, pp. 117–124.

30. **Toni, F. & Sergot, M. (2011).** Argumentation and answer set programming. In **Balduccini, M. & Son, T. C.**, editors, *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*. Springer-Verlag, Berlin, Heidelberg, pp. 164–180.

31. **Wolsey, L. A. (1998).** *Integer Programming*. Discrete Mathematics and Optimization. John Wiley & Sons, Inc.

Article received on 15/04/2015; accepted on 15/09/2015.
Corresponding author is Alejandro Santoyo.