Preliminary Results on Modeling Interdependent Scheduling Games via Answer Set Programming

Giovanni Amendola

Department of Mathematics and Computer Science, University of Calabria, Italy
amendola@mat.unical.it

Abstract. Recently, in the context of planning and coordinating large-scale infrastructures, a model of Interdependent Scheduling Games (ISG) has been proposed. This problem requires interdependent services among players, that control only a limited number of services and schedule independently. A player can schedule his services at any time. However, each service starts to be profitable for the player when all the services on which it depends have been activated. In this paper we approach the ISG by means of Answer Set Programming (ASP). ASP is an established logic-based programming paradigm which has been successfully applied for solving complex combinatorial problems arising in many areas of knowledge. We show how the ISG can be elegantly encoded by means of an ASP program.

1 Introduction

Recently, in the context of planning and coordinating large-scale infrastructures, a model of *Interdependent Scheduling Games* (ISG) in which each player controls a set of services that they schedule independently has been proposed [1]. This problem requires interdependent services among players that control only a limited number of services and schedule independently. A player can schedule his services at any time. However, each service starts to be profitable for the player when all the services on which it depends have been activated. A relevant task is to find a schedule that maximize the most profitable services activated for the longest amount of time.

Complex combinatorial optimization problems, like the one just described, are usually the target for the application of formalisms developed in the area of Artificial Intelligence. Among these, *Answer Set Programming* (ASP) [19, 18], a well-known declarative programming paradigm which has been proposed in the field of logic programming and non-monotonic reasoning, is an ideal candidate. Indeed, ASP combines a comparatively high knowledge-modeling power [19, 3] with a robust solving technology [4, 21, 25, 24, 31, 27, 26, 10, 14–16, 34, 33], that can also be responsive in case of incoherent logic programs [7, 8]. For these reasons ASP has become an established logic-based programming paradigm with successful applications to complex problems in Artificial Intelligence [30, 29, 9, 6], Databases [32], Game Theory [13], Information Extraction [2], and many other fields of knowledge.

The goal of this paper is to provide an assessment of the applicability of ASP to the ISG as a starting point for a more in-depth investigation to address this problem and its variants in comparison with other evaluation approaches.

2 Answer Set Programming

Answer Set Programming (ASP) [19] is a programming paradigm developed in the field of logic programming and nonmonotonic reasoning. In this section, we overview the language of ASP. The reader is referred to [17] for a more detailed introduction.

Syntax. Variables are strings starting with uppercase letter and constants are non-negative integers or strings starting with lowercase letters. A *term* is either a variable or a constant. A *standard atom* is an expression $p(t_1,\ldots,t_n)$, where p is a *predicate* of arity n and $t_1,\ldots,t_n$ are terms. An atom $p(t_1,\ldots,t_n)$ is ground if $t_1,\ldots,t_n$ are constants. A *ground set* is a set of pairs of the form $\langle consts\!:\!conj\rangle$, where $consts$ is a list of constants and $conj$ is a conjunction of ground standard atoms. A *symbolic set* is a set specified syntactically as $\{Terms_1 : Conj_1;\cdots;Terms_t : Conj_t\}$, where $t > 0$, and for each $i = 1,\ldots,t$, $Terms_i$ is a list of terms such that $|Terms_i| > 0$, and each $Conj_i$ is a conjunction of standard atoms. A *set term* is either a symbolic set or a ground set. An *aggregate function*is of the form $f(S)$, where S is a set term, and f is an *aggregate function symbol*. Basically, aggregate functions map multisets of constants to a constant. In the rest of the paper, we will use the following common functions implemented in ASP systems: `#count`, number of terms; and `#sum`, sum of integers. An *aggregate atom* is of the form $f(S) \prec T$, where $f(S)$ is an aggregate function, $\prec\, \in \{<, \leq, >, \geq, =, \neq\}$ is a comparison operator, and T is a term called *guard*. An aggregate atom $f(S) \prec T$ is ground if T is a constant and S is a ground set. An *atom* is either a standard atom or an aggregate atom. A *rule* r is of the form:

$$a_1 \mid \ldots \mid a_n \leftarrow b_1,\ldots, b_k, \texttt{not}\ b_{k+1},\ldots, \texttt{not}\ b_m.$$

where $a_1,\ldots,a_n$ are standard atoms, $b_1,\ldots,b_k$ are atoms, $b_{k+1},\ldots,b_m$ are standard atoms, and $n,k,m \geq 0$. A literal is either a standard atom a or its negation $\texttt{not}\ \ a$. The disjunction $a_1|\ldots|a_n$ is the *head* of r, while the conjunction $b_1,\ldots,b_k,\texttt{not}\ b_{k+1}, \ldots,\texttt{not}\ b_m$ is its *body*. A rule is a *fact* if its body is empty ($\leftarrow$ is omitted), whereas it is a *constraint* if its head is empty. A variable appearing uniquely in set terms of a rule r is said to be *local* in r, otherwise it is *global* in r. An ASP program is a set of *safe* rules. A rule r is *safe* if both the following conditions hold: *(i)* for each global variable X of r there is a positive standard atom ℓ in the body of r such that X appears in ℓ; *(ii)* each local variable of r appearing in a symbolic set $\{Terms\!:\!Conj\}$ also appears in $Conj$.

A *weak constraint* [20] ω is of the form:

$$\leftharpoondown\!\!\sim b_1,\ldots, b_k, \texttt{not}\ b_{k+1},\ldots, \texttt{not}\ b_m.\ [w@l]$$

where w and l are the weight and level of ω. (Intuitively, $[w@l]$ is read "as weight w at level l", where weight is the "cost" of violating the condition in the body of w, whereas levels can be specified for defining a priority among preference criteria). An ASP program with weak constraints is $\Pi = \langle P,W\rangle$, where P is a program and W is a set of weak constraints. A standard atom, a literal, a rule, a program or a weak constraint is *ground* if no variable appears in it.

Semantics. Let P be an ASP program. The *Herbrand universe* U_P and the *Herbrand base* B_P of P are defined as usual [17]. The ground program G_P is the set of all the ground instances of rules of P obtained by substituting variables with constants from U_P. An *interpretation* I for P is a subset I of B_P. A ground atom a is true with respect to I if $a \in I$, and false otherwise. Literal `not` a is true in I if a is false in I, and true otherwise. An aggregate atom is true with respect to I if the evaluation of its aggregate function (i.e., the result of the application of f on the multiset S) with respect to I satisfies the guard; otherwise, it is false. A ground rule r is *satisfied* by I if at least one atom in the head is true with respect to I whenever all conjuncts of the body of r are true with respect to I. A model is an interpretation that satisfies all the rules of a program. Given a ground program G_P and an interpretation I, the *reduct* [23] of G_P with respect to I is the subset G_P^I of G_P obtained by deleting from G_P the rules in which a body literal is false with respect to I. An interpretation I for P is an *answer set* (or stable model [28]) for P if I is a minimal model (under subset inclusion) of G_P^I [23]. We denote the set of all answer sets of an ASP program P by $AS(P)$. A program having an answer set is called coherent, otherwise it is incoherent [5, 12, 11].

Given a program with weak constraints $\Pi = \langle P, W \rangle$, the semantics of Π extends from the basic case defined above. Thus, let $G_\Pi = \langle G_P, G_W \rangle$ be the instantiation of Π; a constraint $\omega \in G_W$ is violated by I if all the literals in ω are true with respect to I. An *optimum answer set* O for Π is an answer set of G_P that minimizes the sum of the weights of the violated weak constraints in a prioritized way. We denote by $OAS(\Pi)$ the set of all optimum answer sets of Π.

3 Interdependent Scheduling Games via ASP

Recently, a model of *Interdependent Scheduling Games* (ISGs) in which each player controls a set of services that they schedule independently has been proposed [1]. An ISG with n players is a tuple of the form $((T_1, \ldots, T_n), G, r)$. Each T_i represents a *set of services* which player i has to schedule, and sets of services are pairwise disjoint and of the same size, i.e., for each $i \neq j$, $T_i \cap T_j = \emptyset$ and $|T_1| = \cdots = |T_n|$. Moreover, let T be the union of all set of services, for each service $u \in T$, there is a (non-negative) *reward* $r(u) \geq 0$, representing payment received in each time period that the service is active. Finally, there is a *dependency relation* among services. Indeed, to activate a service, it and all its prerequisites must be deployed. This is formalized by a transitive acyclic directed graph $G = (T, E)$, where $(u, v) \in E$ if v will generate a reward only after u has been deployed.

Example 1. Consider an ISG with 2 players, say p_1 and p_2, and to each player is associated a set of services: $T_1 = \{u_1, u_2, u_3\}$ and $T_2 = \{v_1, v_2, v_3\}$, respectively. Hence, $T = \{u_1, u_2, u_3, v_1, v_2, v_3\}$. Moreover, the rewards of each service is as follows: $r(u_1) = 10$, $r(u_2) = r(u_3) = r(v_1) = 1$, and $r(v_2) = r(v_3) = 100$. Finally, there are dependency relations from u_1 to v_1; from u_2 to u_3; from u_2 to v_2; and from u_2 to v_3. Hence, $E = \{(u_1, v_1), (u_2, u_3), (u_2, v_2), (u_2, v_3)\}$.

A tuple $\pi = (\pi_1, \ldots, \pi_n)$, where each π_i is a permutation of the services T_i of player i (i.e., $\pi_i : T_i \to \{1, \ldots, |T_i|\}$), is a *schedule* of all services in T. Intuitively, the position $k \in \{1, \ldots, |T_i|\}$ of a service $u \in T_i$ in the permutation π_i denotes the time when

u is deployed. A service u is *active* during a time step if itself and all services v such that $(v,u) \in E$ are deployed at or before that time step. The time when a service u becomes active is denoted by $a(u)$, that is $a(u) = \max\{\pi(v) : v = u \text{ or } (v,u) \in E\}$. Given a schedule $\pi = (\pi_1, \dots, \pi_n)$, the *utility* of player i is

$$R_i(\pi) = \sum_{t=1}^{|T_i|} \sum_{u \in T_i,\ t \geq a(u)} r(u).$$

Finally, the *welfare* of π is defined as $\sum_{i=1}^{n} R_i(\pi)$.

Example 2. Consider the ISG of the Example 1. Let $\pi = (\pi_1, \pi_2)$ be a schedule where $\pi_1(u_1) = 1$, $\pi_1(u_2) = 2$, $\pi_1(u_3) = 3$, and $\pi_2(v_1) = 1$, $\pi_2(v_2) = 2$, $\pi_2(v_3) = 3$. Hence, $R_1(\pi) = 3r(u_1) + 2r(u_2) + r(u_3) = 3 \cdot 10 + 2 \cdot 1 + 1 = 33$, and $R_2(\pi) = 3r(v_1) + 2r(v_2) + r(v_3) = 3 \cdot 1 + 2 \cdot 100 + 100 = 303$, as u_1 and v_1 are active for three time steps, u_2 and v_2 are active for two time steps, and u_3 and v_3 are active for one time step. Therefore, the welfare of π is $R_1(\pi) + R_2(\pi) = 33 + 303 = 336$. Similarly, let $\pi' = (\pi'_1, \pi'_2)$ be a schedule where $\pi'_1(u_2) = 1$, $\pi'_1(u_3) = 2$, $\pi'_1(u_1) = 3$, and $\pi'_2(v_2) = 1$, $\pi'_2(v_3) = 2$, $\pi'_2(v_1) = 3$. Hence, $R_1(\pi') = 3r(u_2) + 2r(u_3) + r(u_1) = 3 \cdot 1 + 2 \cdot 1 + 10 = 15$, and $R_2(\pi') = 3r(v_2) + 2r(v_3) + r(v_1) = 3 \cdot 100 + 2 \cdot 100 + 1 = 501$, as u_2 and v_2 are active for three time steps, u_3 and v_3 are active for two time steps, and u_1 and v_1 are active for one time step. Therefore, the welfare of π' is $R_1(\pi') + R_2(\pi') = 15 + 501 = 516$.

An important task is to find a schedule that maximize the welfare. The corresponding decision problem, denoted by ISG WELFARE, is stated as follows. Given an ISG $((T_1, \dots, T_n), G, r)$ and an integer w, is there a schedule π such that $\Sigma_{i=1}^{n} R_i(\pi) \geq w$? This decision problem is known to be NP-complete [1]. Hence, it can be in theory solved via ASP. We propose an encoding of the problem of computing a welfare greater than the threshold w.

Given an ISG instance $\Gamma = ((T_1, \dots, T_n), G, r)$, and an integer w, the set of services of each player and its corresponding rewards are encoded as facts of the logic program:

$$f_1: \quad \mathit{service}(i, u, r(u)). \qquad \text{if } i \in \{1, \dots, n\} \text{ and } u \in T_i.$$

It means that service u belongs to player i and has a reward $r(u)$. Moreover, we need facts for each dependency of a service from another one, and for the threshold w.

$$\begin{array}{lll} f_2: & \mathit{edge}(u,v). & \text{if } (u,v) \in E. \\ f_3: & \mathit{threshold}(w). & \end{array}$$

Example 3. Considering the ISG of the Example 1, and a threshold $w = 500$. Then, we obtain as facts of the logic program: $\mathit{service}(1, u_1, 10)$, $\mathit{service}(1, u_2, 1)$, $\mathit{service}(1, u_3, 1)$, $\mathit{service}(2, v_1, 1)$, $\mathit{service}(2, v_2, 100)$, $\mathit{service}(2, v_3, 100)$, $\mathit{edge}(u_1, v_1)$, $\mathit{edge}(u_2, u_3)$, $\mathit{edge}(u_2, v_2)$, $\mathit{edge}(u_2, v_3)$, and $\mathit{threshold}(500)$.

First, we consider the following three rules:

$$\begin{array}{ll} \rho_1: & \mathit{numbTotServ}(L) \leftarrow \#\mathit{count}\{C : \mathit{service}(X, C, \_)\} = L. \\ \rho_2: & \mathit{numbServForPlayer}(X, L) \leftarrow \#\mathit{count}\{C : \mathit{service}(X, C, \_)\} = L,\ \mathit{service}(X, \_, \_). \\ \rho_3: & \mathit{maxValueServ}(X, Y, M) \leftarrow \mathit{service}(X, Y, N), \mathit{numbServForPlayer}(X, L), \\ & \qquad M = N * L. \end{array}$$

They encode the total number of services, the number of services for each player, and the possible maximum profit obtainable for each service, respectively.

Example 4. Considering Example 1, by instantiating rules ρ_1, ρ_2, and ρ_3, we will infer the following set of ground atoms: $numbTotServ(6)$, $numbServForPlayer(1, 3)$, $numbServForPlayer(2, 3)$, $maxValueServ(1, u_1, 30)$, $maxValueServ(1, u_2, 3)$, $maxValueServ(1, u_3, 3)$, $maxValueServ(2, v_1, 3)$, $maxValueServ(2, v_2, 300)$, and $maxValueServ(2, v_3, 300)$.

Now, we encode a rule identifying all possible time values that can be used during permutations:

$$\rho_4: \quad pi(X,Y,1..L) \leftarrow service(X,Y,K),\ numbServForPlayer(X,L).$$

Example 5. Considering again the Example 1, by instantiating rule (ρ_4), we will infer the following ground atoms: $pi(1, u_1, 1)$, $pi(1, u_1, 2)$, $pi(1, u_1, 3)$, $pi(1, u_2, 1)$, $pi(1, u_2, 2)$, $pi(1, u_2, 3)$, $pi(1, u_3, 1)$, $pi(1, u_3, 2)$, $pi(1, u_3, 3)$ for the player p_1, and $pi(2, v_1, 1)$, $pi(2, v_1, 2)$, $pi(2, v_1, 3)$, $pi(2, v_2, 1)$, $pi(2, v_2, 2)$, $pi(2, v_2, 3)$, $pi(2, v_3, 1)$, $pi(2, v_3, 2)$, $pi(2, v_3, 3)$ for the player p_2.

The following set of rules guess a possible schedule, by identifying permutations of services for each player.

$$\begin{array}{ll}
\rho_5: inPi(X,Y,Z) \mid outPi(X,Y,Z) & \leftarrow pi(X,Y,Z). \\
\rho_6: & \leftarrow inPi(X,Y,Z),\ inPi(X,V,Z),\ Y > V. \\
\rho_7: & \leftarrow inPi(X,Y,Z),\ inPi(X,Y,W),\ Z > W. \\
\rho_8: & \leftarrow \#count\{Y : inPi(X,Y,Z)\} < L,\ numbTotServ(L).
\end{array}$$

Now, given two services $S1$ and $S2$, such that $S2$ depends on $S1$, it must not happen that the execution time of $S2$, namely $T2$, is strictly lower than the execution time of $S1$, namely $T1$. The following constraint encodes this condition.

$$\rho_9: \quad \leftarrow inPi(P1,S1,T1),\ inPi(P2,S2,T2),\ T2 < T1,\ edge(S1,S2).$$

To compute the profit obtained by a player X on service Y, we have to multiply the reward of the service Y, namely M, by the number of time steps for which Y has been used, that is $L+1-Z$, where Z is the time step associated to Y with respect to the permuation considered. Hence,

$$\begin{array}{lll}
\rho_{10}: & val(X,Y,K) \leftarrow & inPi(X,Y,Z),\ numbServForPlayer(X,L), \\
 & & service(X,Y,M),\ K = M * (L+1-Z).
\end{array}$$

Finally, the welfare is obtained by the following rule:

$$\rho_{11}: \quad welfare(N) \leftarrow \#sum\{M,Y : val(X,Y,M)\} = N.$$

We denote by P_Γ the ASP program so constructed, i.e. $P_\Gamma = \{f_1, f_2, f_3\} \cup \bigcup_{i=1}^{11}\{\rho_i\}$.

The previous enconding allows to easily compute different reasoning tasks. To compute a welfare greater than the threshold w, we just add the following rule.

$$\rho_{12}: \quad \leftarrow welfare(N),\ threshold(W),\ N < W.$$

Rule ρ_{12} is a constraint requiring that the welfare, namely N, must not be less than the threshold, namely W. Therefore, we can formally prove the completeness and soundness of the encoding proposed.

Theorem 1. *Let $\Gamma = ((T_1, \dots, T_n), G, r)$ be an ISG, and let w be an integer. Then, there is a schedule π such that $\sum_{i=1}^{n} R_i(\pi) \geq w$ if, and only if, $AS(P_\Gamma \cup \{\rho_{12}\}) \neq \emptyset$.*

Note that, starting from P_Γ we can easily compute other interesting reasoning tasks. For instance, we can compute the *maximum welfare* for an ISG by adding the following weak constraint.

$$\rho_{13}: \qquad \rightsquigarrow \ val(X,Y,K),\ maxValueServ(X,Y,M),\ L = M - K. \quad [L@1]$$

Indeed, ρ_{13} minimizes the difference between the possible maximum profit obtainable for a given service, namely M, and the current profit of that service, namely K. Also in this case, the completeness and soundness of the approach can be proved.

Theorem 2. *Let $\Gamma = ((T_1, \dots, T_n), G, r)$ be an ISG. Then, μ is the maximum welfare for Γ if, and only if, for each $O \in OAS(\langle P_\Gamma, \{\rho_{13}\}\rangle)$, it holds that $welfare(\mu) \in O$.*

4 Conclusion and Future Work

In the paper, we showed that the ISG can be elegantly encoded by means of an ASP program. In particular, we deal with the problem of maximizing the welfare (ISG WELFARE). However, another interesting question is known as ISG BEST RESPONSE [1]. Here, the goal is to find a best response for a given player by knowing the actions of each other player. More specifically, given an ISG $((T_1, \dots, T_n), G, r)$, a schedule $(\pi_1, \dots, \pi_{i-1}, \pi_{i+1}, \dots, \pi_n)$ for all players except for i, and an integer w, we ask for a permutation π'_i of the services T_i of player i such that $R_i(\pi_1, \dots, \pi_{i-1}, \pi'_i, \pi_{i+1}, \dots, p_n) \geq w$. Moreover, as future work, we want to address the ISG and its possible variants (such as extensions of the model including cyclic interdependencies [22], adding services or misreporting utilities [35]) via ASP in comparison with other modeling approaches, such as the integer linear programming formulation [1].

References

1. Abeliuk, A., Aziz, H., Berbeglia, G., Gaspers, S., Kalina, P., Mattei, N., Peters, D., Stursberg, P., Hentenryck, P.V., Walsh, T.: Interdependent scheduling games. In: IJCAI 2016. pp. 2–9 (2016)
2. Adrian, W.T., Manna, M., Leone, N., Amendola, G., Adrian, M.: Entity set expansion from the web via ASP. In: ICLP (Technical Communications). OASICS, vol. 58, pp. 1:1–1:5. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2017)
3. Alviano, M., Amendola, G., Peñaloza, R.: Minimal undefinedness for fuzzy answer sets. In: AAAI. pp. 3694–3700. AAAI Press (2017)
4. Alviano, M., Dodaro, C., Ricca, F.: A MaxSAT Algorithm Using Cardinality Constraints of Bounded Size. In: IJCAI 2015. pp. 2677–2683 (2015)

5. Amendola, G.: Dealing with incoherence in ASP: split semi-equilibrium semantics. In: DWAI@AI*IA. CEUR Workshop Proceedings, vol. 1334, pp. 23–32. CEUR-WS.org (2014)
6. Amendola, G.: Solving the stable roommates problem using incoherent answer set programs. In: RCRA@AI*IA. p. to appear. CEUR Workshop Proceedings, CEUR-WS.org (2018)
7. Amendola, G., Dodaro, C., Faber, W., Leone, N., Ricca, F.: On the computation of paracoherent answer sets. In: AAAI. pp. 1034–1040. AAAI Press (2017)
8. Amendola, G., Dodaro, C., Faber, W., Ricca, F.: Externally supported models for efficient computation of paracoherent answer sets. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, February 2-7, 2018, New Orleans, Louisiana, USA. pp. 1034–1040 (2018)
9. Amendola, G., Dodaro, C., Leone, N., Ricca, F.: On the application of answer set programming to the conference paper assignment problem. In: AI*IA. Lecture Notes in Computer Science, vol. 10037, pp. 164–178. Springer (2016)
10. Amendola, G., Dodaro, C., Ricca, F.: ASPQ: an asp-based 2qbf solver. In: QBF@SAT. CEUR Workshop Proceedings, vol. 1719, pp. 49–54. CEUR-WS.org (2016)
11. Amendola, G., Eiter, T., Fink, M., Leone, N., Moura, J.: Semi-equilibrium models for paracoherent answer set programs. Artif. Intell. **234**, 219–271 (2016)
12. Amendola, G., Eiter, T., Leone, N.: Modular paracoherent answer sets. In: Logics in Artificial Intelligence - 14th European Conference, JELIA2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings. pp. 457–471 (2014)
13. Amendola, G., Greco, G., Leone, N., Veltri, P.: Modeling and reasoning about NTU games via answer set programming. In: IJCAI 2016. pp. 38–45 (2016)
14. Amendola, G., Ricca, F., Truszczynski, M.: Generating hard random boolean formulas and disjunctive logic programs. In: IJCAI. pp. 532–538. ijcai.org (2017)
15. Amendola, G., Ricca, F., Truszczynski, M.: A generator of hard 2qbf formulas and asp programs. In: KR. AAAI Press (2018)
16. Amendola, G., Ricca, F., Truszczynski, M.: Random models of very hard 2qbf and disjunctive programs: An overview. In: ICTCS. CEUR Workshop Proceedings, CEUR-WS.org (2018)
17. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
18. Bonatti, P.A., Calimeri, F., Leone, N., Ricca, F.: Answer set programming. In: 25 Years GULP. Lecture Notes in Computer Science, vol. 6125, pp. 159–182. Springer (2010)
19. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Com. ACM **54**(12), 92–103 (2011)
20. Buccafurri, F., Leone, N., Rullo, P.: Enhancing disjunctive datalog by constraints. IEEE Trans. Knowl. Data Eng. **12**(5), 845–860 (2000)
21. Calimeri, F., Gebser, M., Maratea, M., Ricca, F.: Design and results of the Fifth Answer Set Programming Competition. Artif. Intell. **231**, 151–181 (2016)
22. Coffrin, C., Hentenryck, P.V., Bent, R.: Last-mile restoration for multiple interdependent infrastructures. In: AAAI. AAAI Press (2012)
23. Faber, W., Pfeifer, G., Leone, N.: Semantics and complexity of recursive aggregates in answer set programming. Artif. Intell. **175**(1), 278–298 (2011)
24. Gebser, M., Leone, N., Maratea, M., Perri, S., Ricca, F., Schaub, T.: Evaluation techniques and systems for answer set programming: a survey. In: IJCAI 2018. pp. 5450–5456 (2018)
25. Gebser, M., Maratea, M., Ricca, F.: The Design of the Sixth Answer Set Programming Competition. In: LPNMR. LNCS, vol. 9345, pp. 531–544. Springer (2015)
26. Gebser, M., Maratea, M., Ricca, F.: What's hot in the answer set programming competition. In: AAAI. pp. 4327–4329. AAAI Press (2016)
27. Gebser, M., Maratea, M., Ricca, F.: The sixth answer set programming competition. Journal of Artificial Intelligence Research **60**, 41–95 (2017)

28. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Comput. **9**(3/4), 365–386 (1991)
29. Grasso, G., Iiritano, S., Leone, N., Lio, V., Ricca, F., Scalise, F.: An asp-based system for team-building in the gioia-tauro seaport. In: PADL. Lecture Notes in Computer Science, vol. 5937, pp. 40–42. Springer (2010)
30. Grasso, G., Iiritano, S., Leone, N., Ricca, F.: Some DLV applications for knowledge management. In: LPNMR. Lecture Notes in Computer Science, vol. 5753, pp. 591–597. Springer (2009)
31. Lierler, Y., Maratea, M., Ricca, F.: Systems, engineering environments, and competitions. AI Magazine **37**(3), 45–52 (2016)
32. Manna, M., Ricca, F., Terracina, G.: Taming primary key violations to query large inconsistent data via ASP. TPLP **15**(4-5), 696–710 (2015)
33. Maratea, M., Pulina, L., Ricca, F.: A multi-engine approach to answer-set programming. TPLP **14**(6), 841–868 (2014)
34. Maratea, M., Ricca, F., Faber, W., Leone, N.: Look-back techniques and heuristics in DLV: implementation, evaluation, and comparison to QBF solvers. J. Algorithms **63**(1-3), 70–89 (2008)
35. Zlotkin, G., Rosenschein, J.S.: Mechanism design for automated negotiation, and its application to task oriented domains. Artif. Intell. **86**(2), 195–244 (1996)