Journal of Theoretical and Applied Information Technology
10th October 2013. Vol. 56 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

USING ANSWER SET PROGRAMMING TO FIND MAXIMUM HEIGHT SPANNING TREES

MARCO MANNA
University of Calabria, Department of Mathematics and Informatics, 87036 – Rende, IT
E-mail: manna@mat.unical.it

ABSTRACT

The max tree-height of an undirected graph is the longest possible length of a path among all spanning trees of the graph. A maximum height spanning tree of an undirected graph is a spanning tree that has a path of length equal to the max tree-height of the graph. Finding the max tree-height of a graph, or similarly some spanning tree of maximum height, is an NP-hard optimization problem for which efficient optimal procedures have been proposed only for special classes of graphs, and which is not polynomially approximable within any constant factor unless PTIME = NP. The paper presents an elegant yet efficient and succinct logic program in Answer Set Programming for the identification of both the max tree-height and the maximum height spanning trees of a graph.

Keywords: *Answer Set Programming, Declarative Problem Solving, Artificial Intelligence, Spanning Trees, Optimization Problems*

1. INTRODUCTION

Problems on graphs are at the basis of many real-world applications in every field of Computer Science [18]. Some of these problems are tractable and others are computationally hard. In both cases, most of the research effort has been devoted to the design of more and more scalable algorithms that are able to deal with graphs of large size. However, in the former case, such algorithms are exact, namely they always find the best answer. Conversely, in the latter case, the designed algorithms either work correctly only if suitable restrictions are imposed on the instances of the problem, or they are usually heuristics that find suboptimal solutions. This is the case of the problem of finding the *max tree-height* of an undirected graph, namely the longest possible length of a path among all spanning trees of the graph. This is an NP-hard optimization problem for which efficient optimal procedures have been proposed only for very special classes of graphs (see, e.g., [7]), and which is not polynomially approximable within any constant factor unless PTIME = NP [12].

Consider the graph G depicted in Figure 1, which consists of 9 vertices and 12 edges. It is not difficult to see that G has a max tree-height of 4. In fact, a witness spanning tree T exhibiting this bound can be obtained from G by removing the dashed edges, namely {1,2}, {1,4}, {6,7}, and {8,9}. Moreover, due to the special structure of G, it is not possible to find any spanning tree of G with a path of length greater than 4. For this reason, tree T is said to be a *maximum height spanning tree* of G.

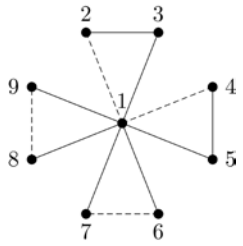

Figure 1: Cactus Graph.

The graph of Figure 1 is actually a *cactus graph*, namely a connected undirected graph in which every block is either an edge or a cycle. For such graphs, for instance, the problem of identifying the max tree-height is tractable and, in particular, it is doable in liner time with respect to the number of vertices of the graph [7].

Unfortunately, there are contexts in which suboptimal solutions have to be avoided as much as possible, especially for those NP-hard optimization problems where no polynomial approximation within any constant factor is guaranteed. As an example, we mention the context of *Ontology Based Data Access* [6], a novel field of research in which the objective is to provide access to data stored in heterogeneous data sources trough a semantic layer in the form of an ontology. In

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

particular, we consider the case in which a query has to be evaluated over a database paired with a set of ontological axioms. In this setting, given a database D, a set of axioms Σ, and a conjunctive query q, the key decision problem is to determine whether q is entailed by D and Σ (namely, whether $D \cup \Sigma \models q$). However, in case Σ is a set of *shy* rules [20, 3], the unique deductive procedure that has been designed so far to solve this problem relies on the size of q. More specifically, this procedure performs a number k of macro iterations which, to ensure correctness, must be at least equal to the max tree-height of the *primal graph* [15] associated with q. Since the time required by this procedure grows exponentially with k, it is clear that k must be as small as possible, namely it should desirably coincide with the max tree-height of the primal graph associated with q.

To easily identify both the max tree-height and the maximum height spanning trees of a graph, we design a disjunctive logic program with constraints in *Answer Set Programming* (ASP), a fully declarative language for Knowledge Representation and Reasoning. ASP has been developed in the field of logic programming and nonmonotonic reasoning, and has been already exploited for solving complex knowledge-based problems in many areas of Knowledge Management, such as Artificial Intelligence and Information Integration (see, e.g., [16, 17, 23, 24, 26]).

The proposed approach has at least three strengths. First, it computes optimal solutions. Second, the designed program is quite elegant and succinct; and this is important especially in contexts where specific adaptations are required. Third, implementation issues are completely demanded to efficient ASP solvers (see, e.g., cmodels [22], DLV [21, 25], clasp [13], and WASP [8, 9]) that have been developed in the last few years to represent and manipulate complex knowledge (see, e.g., [5] for an in-depth comparison among these solvers and [11] for an Integrated Development Environment for ASP supporting the entire life-cycle of ASP development).

2. TECHNICAL BACKGROUND

2.1 Maximum Height Spanning Trees

An *undirected graph* (hereafter just *graph*) G is a pair (V,E) where V are the *vertices* and E are the *edges* of G. Without loss of generality, we assume that vertices are positive integers and that edges are sets of two distinct vertices. Consider a graph $G = (V,E)$. A *path* in G, from a vertex $a \in V$ to another vertex $b \in V$, is any non-empty sequence S of edges such that: (i) both a and b appear in exactly one edge of S; (ii) any two successive edges in the sequence S share a vertex; (iii) any vertex of G other than a and b appears either in exactly two edges of S or in none of the edges of S. The length of S is given by the number of its edges. Graph G is *connected* if there is a path between each pair of its vertices, and it is called a *tree* if there is exactly one path between each pair of its vertices. In the latter case, for a given tree T, the length of the path between two vertices a and b is denoted by $\mathsf{len}_T(a,b)$, which of course is equivalent to $\mathsf{len}_T(b,a)$. By definition, $\mathsf{len}_T(a,a) = 0$.

The *height* (also known as *diameter*) of a tree T over a set V of vertices, denoted by $\mathsf{height}(T)$, is the length of the longest path of T. More precisely, it is defined as follows: $\mathsf{height}(T) = \max(\{\mathsf{len}_T(a,b) \mid a,b \in V\})$. A spanning tree of G is a tree $T = (V,K)$ that shares with G the same set of vertices and where $K \subseteq E$. We are now ready to introduce the notion of maximum height spanning tree of a graph.

Definition 1. Consider a connected graph G. A spanning tree T of G is said to be a *maximum height spanning tree* of G if, for each spanning tree T' of G, it holds that $\mathsf{height}(T) \geq \mathsf{height}(T')$. □

Using the previous definition, we can also introduce the notion of max tree-height of a graph.

Definition 2. The *max tree-height* of a connected graph G is the height of any of the maximum height spanning trees of G. □

2.2 Height of a Tree

Any algorithm for determining the max tree-height of a graph, unavoidably, has to rely on some subroutine that determines the height of a tree. Hence, the design a good algorithm for the first problem should use some efficient subroutine. To this regard, we describe an algorithm that has the status of folklore in the literature, but to the best of our knowledge it has never been explicitly stated. Such an algorithm, called tree-height-finder, is described at the end of this section.

Proposition 1. *Consider a tree T with n vertices. Algorithm tree-height-finder correctly finds the height of T in time $\mathcal{O}(n)$ on a real computer.*

Proof. Let u be a vertex of T, v be one of the farthest vertex from u, and w be one of the farthest vertex from v. Assume now that there exists another path from a to b such that:

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$$\mathsf{len}_T(v,w) < \mathsf{len}_T(a,b). \quad (1)$$

First, observe that such a path contains a vertex c (not necessarily distinct from u) which is reachable from u, and that appears both in the path from u to a and the one from u to b (see Figure 2). Without loss of generality, assume that $\mathsf{len}_T(c,a) \geq \mathsf{len}_T(c,b)$. Expression (1) can now be rewritten as follows:

$$\mathsf{len}_T(u,v) + \mathsf{len}_T(u,w) < \mathsf{len}_T(c,a) + \mathsf{len}_T(c,b). \quad (2)$$

Moreover, to comply with the hypothesis stating that w is one of the farthest vertex from v, also the following inequality must hold (see Figure 2):

$$\mathsf{len}_T(u,c) + \mathsf{len}_T(c,a) \leq \mathsf{len}_T(u,w). \quad (3)$$

In fact, otherwise, if expression (3) were not true, we would have $\mathsf{len}_T(v,w) < \mathsf{len}_T(v,a)$. Now, by combining expression (2) and (3), we have:

$$\mathsf{len}_T(u,v) < \mathsf{len}_T(c,b) - \mathsf{len}_T(u,c),$$

which is a contradiction since we know that $\mathsf{len}_T(c,b) \leq \mathsf{len}_T(u,v)$. Therefore, such a path from a to b cannot exists.

To conclude the proof, since the number of edges of a tree is linear in the number of nodes, each of the two traversals of T is doable in linear time with respect to the number of vertices of T. □

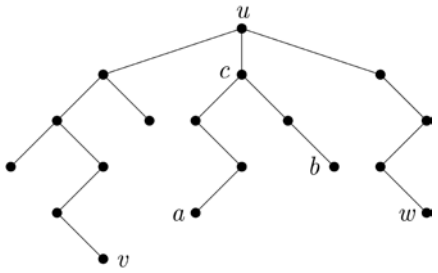

Figure 2: Height of a tree.

Remark. Proposition 1 actually says that to determine the height of a tree T, it is not necessary to consider all the possible $\mathcal{O}(|V|^2)$ paths of T, but it suffices to consider only $2\cdot|V|$ such paths.

ALGORITHM tree-height-finder

Input: A tree T.

Output: The height of T.

1. Pick an arbitrary vertex u of T;
2. Perform a traversal of T from vertex u to compute the distance between u and every other vertex of T;
3. Pick one of the farthest vertex from u, say v;
4. Perform another traversal of T from v to compute the distance between v and every other vertex of T;
5. Return the maximum distance between v and the other vertices of T.

2.3 Answer Set Programming

In this section, we recall some basic notion of ASP [3, 4, 10, 14, 19]. We assume sets of *variables*, *constants*, and *predicates* to be given, variables to be strings starting with uppercase letters, and constants to be non-negative integers or strings starting with lowercase letters. Predicates are strings starting with lowercase letters. An *arity* (non-negative integer) is associated with each predicate. Moreover, the language allows for using built-in predicates (i.e., predicates with a fixed meaning) for the common arithmetic operations.

A *term* is either a variable or a constant. A *standard atom* is an expression $p(t_1,\ldots,t_k)$, where p is a *predicate* of arity k and $t_1,\ldots,t_k$ are terms. If $t_1,\ldots,t_k$ are constants, then $p(t_1,\ldots,t_k)$ is a *ground standard atom*. A *set term* is either a symbolic set or a ground set. A *symbolic set* is a pair $\{Terms : Conj\}$, where *Terms* is a list of terms, and *Conj* is a conjunction of standard atoms. Intuitively, a set term $X : a(X,Y), b(Y)$ stands for the set of X-values making the conjunction $a(X,Y), b(Y)$ true, namely $\{X \mid \exists Y \text{ s.t. } a(X,Y),\ b(Y) \text{ is true}\}$. 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.

An *aggregate function* is of the form $f(S)$, where S is a set term and f is an *aggregate function symbol*. Intuitively, an aggregate function can be thought of as a function, mapping multisets of constants to constants. Hereinafter, we will adopt the following notation of the DLV system [21] for representing aggregate functions: `#min` (minimal term); `#max` (maximal term); and `#count` (number of terms). An *aggregate atom* is a structure of the form $f(S) \odot T$, where $f(S)$ is an aggregate function, $\odot$ is a comparison operator, and T is a term. If T is a constant and S is a ground set term, then $f(S) \odot T$ is a *ground aggregate atom*. A *literal* is a standard atom, or a standard atom preceded by the *negation as failure* symbol `not`, or an aggregate atom.

A rule r is an expression of the form:

$$\alpha_1 \vee \ldots \vee \alpha_n \ :- \ \ell_1, \ \ldots, \ \ell_m,$$

where $\alpha_1, \ldots, \alpha_n$ are standard atoms and $\ell_1, \ \ldots, \ \ell_m$ are literals. The disjunction $\alpha_1 \vee \ldots \vee \alpha_n$ is referred to as the *head* of r, while the conjunction $\ell_1, \ \ldots, \ \ell_m$ is the *body* of r. We denote by $H(r)$ the set of head atoms and by $B(r)$ the set of body literals. If $H(r)$ and all the literals in $B(r)$ are ground, then r is a *ground rule*. A ground rule r is a *fact* if both $B(r)$ is empty and $H(r)$ is a singleton. If

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$H(r)$ is empty, then r is said to be a *strong constraint*. A *program* P is a set of rules; if all rules of P are ground, then P is a ground program.

A program P can be also equipped with a set of *weak constraints* to formulate optimization problems in an easy and natural way. While strong constraints always have to be satisfied, weak constraints express desiderata, i.e., they should be satisfied if it is possible. A weak constraint is an expression of the form:

$$:\sim \ \ell_1, \ \ldots, \ \ell_m. \ [W],$$

where $\ell_1, \ \ldots, \ \ell_m$ are literals, and W is a term representing the *weight* of the constraint.

2.4 Answer Set Semantics

Consider an ASP program P (with no weak constraint). Before defining the semantics of P, we assume that every constraint γ of P of the form $:- \ell_1, \ \ldots, \ \ell_m$ has been replaced by a rule of the form $co_\gamma :- \ell_1, \ \ldots, \ \ell_m,$ `not` co_γ, where co_γ is a symbol that does not occur in any other rule of P. The *universe* of P, denoted by U_P, is the set of constants appearing in P. The *base* of P, denoted by B_P, is the set of standard atoms constructible from predicates of P with constants in U_P. A *substitution* is a mapping from a set of variables to U_P. Given a substitution σ and any structure s containing atoms, the structure obtained by replacing each variable X of s with $\sigma(X)$ is denoted by $\sigma(s)$. A substitution over the set of global variables of rule r is a *global substitution* for r; a substitution over the set of local variables of a set term S is a *local substitution* for S. Given a set term $S = \{Terms : Conj\}$ without global variables, the *instantiation* of S, denoted by $inst(S)$, is the ground set term $\{\langle\sigma(Terms):\sigma(Conj)\rangle \mid \sigma$ is a local substitution for $S\}$. A *ground instance* of a rule r is obtained in two steps: first, a global substitution σ for r is applied; then, every set term S in $\sigma(r)$ is replaced by its instantiation $inst(S)$. The instantiation $Ground(P)$ of program P is the set of instances of all rules in P.

An interpretation I for a program P is a subset of B_P. A standard ground atom α is true with respect to I if $\alpha \in I$; otherwise, α is false with respect to I. A negative standard ground literal `not` α is true with respect to I if $\alpha \notin I$; otherwise, `not` α is true with respect to I. Consider a set term S. Let S^I denote the set $\{\langle t_1,\ldots,t_k\rangle \mid \langle t_1,\ldots,t_k : Conj\rangle \in S$, and all the atoms of $Conj$ are true with respect to $I\}$. The evaluation of S with respect to I is the multiset $I(S) = \{t_1 \mid \langle t_1,\ldots,t_k\rangle \in S^I\}$ obtained as the projection of the tuples of S^I on their first constant. The evaluation $I(f(S))$ of an aggregate function $f(S)$ with respect to I is the result of the application of f on $I(S)$. If the multiset $I(S)$ is not in the domain of f, $I(f(S)) = \perp$ (where $\perp$ is a fixed symbol not occurring in P). A ground aggregate atom $f(S) \odot k$ is true with respect to I if both $I(f(S)) \neq \perp$ and $I(f(S)) - k$ hold; otherwise, $f(S) - k$ is false.

Given an interpretation I, a rule r is satisfied with respect to I if some head atom is true with respect to I whenever all body literals are true with respect to I. An interpretation M is a *model* of a program P if all the rules r of $Ground(P)$ are satisfied with respect to M. A model M of P is (subset) minimal if no model N of P exists such that $N \subset M$. Let P be a ground program, I be an interpretation, and P^I denote the transformed program obtained from P by deleting all rules in which a body literal is false with respect to I. An interpretation M is an answer set of P if it is a minimal model of $Ground(P)^M$. Finally, the answer sets of P plus a set of weak constraints Ω are those answer sets of P which minimize the sum of the weights of the violated weak constraints.

3. ASP-BASED ENCODING

We now propose an ASP-based encoding for the identification of both the max tree-height and the maximum height spanning trees of a graph. More specifically, we define a mapping μ that encodes a graph as a set of facts, and we design a program P and a weak constraint ω such that, for every graph G, T is a maximum height spanning tree of G if and only if T is encoded by an answer set of program $\mu(G) \cup P \cup \{\omega\}$. Moreover, we show how to determine the max tree-height of G from $\mu(G) \cup P \cup \{\omega\}$. For the rest of this section, let $G = (V,E)$ denote a connected graph.

First, we start by defining the mapping μ, which associates a fact to each edge of G as follows: $\mu(G) = \{$`edge(`u`,`v`)` $\mid \{u,v\} \in E, \ u < v\}$. Then, we construct program P by gradually introducing its rules according to role they play.

We start with two rules that derive the vertices of G by exploiting its edges, namely they infer the set of atoms $\{$`vertex(`v`)` $\mid v \in V\}$ (observe that, since G is assumed to be connected, each vertex of G appears in at least one edge of G):

```
vertex(U) :- edge(U,V).
vertex(V) :- edge(U,V).
```

Once we have the vertices of G, we can count them by using the built-in aggregate `#count` to

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

derive the atom `size(n)`, where $n = |V|$. This can be obtained via the following rule:

```
size(N) :- #count{V:vertex(V)} = N.
```

Moreover, it will result useful to fix a node of the graph that will play the role of root in the spanning trees that we are going to construct. Without loss of generality, we choose the smallest node of V (recall that vertices are positive integers). This can be achieved by the following rule, which uses the aggregate function `#min` to derive the atom `first(`u_r`)`, where $u_r = \min(V)$:

```
first(Ur) :- #min{V:vertex(V)} = Ur.
```

All the atoms derived so far are obtained deterministically, and they are part of every answer set of $\mu(G) \cup P \cup \{\omega\}$. Conversely, to identify all maximum height spanning trees of G, we need to derive different sets of atoms, each of which consists of the edges of some maximum height spanning tree of G, and all of which give rise to the answer sets of $\mu(G) \cup P \cup \{\omega\}$. To this end, we introduce the next disjunctive rule to *pick* (or *guess*) from E different sets of edges, each of which forms a candidate maximum height spanning tree of G:

```
pick(U,V) ∨ discard(U,V) :- edge(U,V).
```

Note that we used the term "candidate" since for an arbitrary set of edges, say S, the graph (V,S) could be neither a maximum height spanning tree, nor even a tree of G. Hence, it is clear that we need further rules to *check* whether S satisfies or not our desiderata. In the former case, S gives rise to one of the answer sets of $\mu(G) \cup P \cup \{\omega\}$. In the latter case, S will be ignored instead. Let us now fix one possible set of edges S and consider the set `{pick(`u`,`v`)` $\mid \{u,v\} \in S,\ u<v\}$ consisting of the atoms associated with S that are derived by the above disjunctive rule. We are going to introduce suitable rules and constraints to determine whether graph $C = (V,S)$ is a maximum height spanning tree of G or whether the set S has to be ignored without being part of any answer set of $\mu(G) \cup P \cup \{\omega\}$.

By using the three recursive rules below, we can collect all the nodes that are reachable in C from the node $u_r = \min(V)$.

```
reachable(Ur) :- first(Ur).
reachable(V) :- reachable(U), pick(U,V).
reachable(U) :- reachable(V), pick(U,V).
```

A first reason to ignore S is that C is actually not a tree due to some vertex of V that is not reachable from u_r. To identify such situations, we add to program P the following strong constraint, saying that it is not possible that v is a vertex and v is not reachable from u_r:

```
:- vertex(V), not reachable(V).
```

Moreover, it could also be the case that C is not a tree since the cardinality of S is greater than $n-1$. (Note that if each vertex of V is reachable from u_r, then the cardinality of S is at least $n-1$.) This can be checked by adding to P also the following strong constraint:

```
:- size(N), #count{U,V:pick(U,V)} >= N.
```

Up to this point we have guaranteed that C is actually a spanning tree of G. Now, it remains to enforce that the height of C is equal to the max tree-height of G. To this end, we propose a declarative version of the technique described in Section 2.2 to find the height of a tree. For convenience, for each edge $\{u,v\} \in S$, the next rules derive the two atoms `sEdge(`u`,`v`)` and `sEdge(`v`,`u`)`, which allow us to treat each edge of C as a set of two vertices:

```
sEdge(U,V) :- pick(U,V).
sEdge(V,U) :- pick(U,V).
```

The following rules simulate a traversal of C from u_r to determine the distance between u_r and every other vertex of C:

```
trv1(Ur,V,1) :- first(Ur), sEdge(Ur,V).
trv1(V,W,D1) :- trv1(U,V,D), sEdge(V,W),
                U<>W, D1 = D + 1.
```

In particular, the meaning of each atom of the form `trv1(`u`,`v`,`d`)` is that the distance between u_r and v is d, namely $d = \mathsf{len}_C(u_r,v)$. Moreover, u is the vertex that precedes v in the path from u_r to v. Note that it is important to keep the predecessor of each vertex to avoid, in the second rule, that the traversal extends to vertices that have already been visited.

The next rule selects one of the farthest vertex from u_r:

```
far1(Vr) :- Dr = #max{D:trv1(_,_,D)},
            Vr = #min{V:dist1(_,V,Dr)}.
```

Actually, this rule first determines the maximum distance, say d_r, from u_r to any other vertex. Then, it picks the minimum vertex, call it v_r, that is at distance d_r from u_r.

At this point, the following two rules simulate a second traversal of C from v_r to determine the distance between v_r and every other vertex of C:

```
trv2(Vr,V,1) :- far1(Vr), sEdge(Vr,V).
trv2(V,W,D1) :- trv2(U,V,D), sEdge(V,W),
                U<>W, D1 = D + 1.
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Similarly to the first traversal, the meaning of each atom of the form `trv2(u,v,d)` is that the distance between v_r and v is d and that u is the vertex that precedes v in the path from v_r to v.

The next rule selects the maximum distance, say d_m, between v_r and the other vertices of C:

```
maxDist(Dm) :- Dm = #max{D:trv2(_,_,D)}.
```

Finally, we define the weak constraint ω, which guarantees that S (and therefore C) is not ignored only if its height d_m is the max tree-height of G. Actually, ω allows us to ignores S if the value $n-d_m$ is not the minimum one over all possible different sets of edged guessed by the only disjunctive rule of P:

```
:~ size(N), maxDist(Dm), V=N-Dm. [V]
```

As previously stated, the answer sets of the program $\mu(G) \cup P$ enhanced with ω are representative of all the maximum height spanning tree of G. However, a run of any ASP solver over this program can also return the valued d_m that appears in each answer set that satisfies the weak constraint. In fact, all the maximum height spanning trees of G share the same value for the height d_m. Hence, it suffices to ask for at least one answer set of $\mu(G) \cup P \cup \{\omega\}$, which necessarily contains the atom `maxDist(`d_m`)`.

4. DISCUSSION

We have presented a succinct yet elegant ASP program for the identification of both the max tree-height and the maximum height spanning trees of a graph, which are both NP-hard optimization problem for which efficient optimal procedures have been proposed only for special classes of graphs and which are not polynomially approximable within any constant factor unless PTIME = NP.

The approach has been profitably applied in the context of Ontology Based Data Access, to determine the max tree-height of the primal graph associated with a conjunctive query. In this setting, due to the size of real-world conjunctive queries, which may give rise to graphs with few dozens of vertices, the time required by any ASP solver which interprets the presented program to identify the max tree-height of the graph associated with a conjunctive query is negligible when compared with the time required to evaluate the conjunctive query itself, and that may be drastically reduced by exploiting the max tree-height of the primal graph associated with the query.

Finally, an interesting line for future research is to extend the proposed encoding for dealing with possibly unconnected graphs. In these case, the notion of max tree-height is naturally extended by considering the maximum value among all possible max tree-heights of the different connected components of the graph.

REFRENCES:

[1] M. Alviano, F. Calimeri, W. Faber, S. Perri, and N. Leone, "Unfounded Sets and Well-Founded Semantics of Answer Set Programs with Aggregates", *Journal of Artificial Intelligence Research*, Vol. 42, 2011, p. 487-527.

[2] M. Alviano, W. Faber, N. Leone, S. Perri, G. Pfeifer, and G. Terracina, "The Disjunctive Datalog System DLV", *Datalog Reloaded - First International Workshop (Datalog 2010)*, LNCS, 2011, pp. 282-301.

[3] M. Alviano, N. Leone, M. Manna, G. Terracina, and P. Veltri, "Magic-Sets for Datalog with Existential Quantifiers", *Datalog in Academia and Industry - Second International Workshop (Datalog 2.0)*, LNCS, Vol. 7494, 2012, pp. 31-43.

[4] C. Baral, "Knowledge Representation, Reasoning and Declarative Problem Solving", *Cambridge University Press*, 2003.

[5] F. Calimeri, G. Ianni, F. Ricca, M. Alviano, A. Bria, G. Catalano, S. Cozza, W. Faber, O. Febbraro, N. Leone, M. Manna, A. Martello, C. Panetta, S. Perri, K. Reale, M.C. Santoro, M. Sirianni, G. Terracina, P. Veltri, "The Third Answer Set Programming Competition: Preliminary Report of the System Competition Track", *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, Vancouver, Canada, May 16-19, 2011, pp. 388-403.

[6] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati, "Ontology-based Database Access", *Proceedings of the 15th Italian Symposium on Advanced Database Systems (SEBD)*, Fasano, Italy, June 17-20, 2007, pp. 324-331.

[7] K. Das, and M. Pal, "An Optimal Algorithm to Find Maximum and Minimum Height Spanning Trees on Cactus Graphs", *AMO - Advanced Modeling and Optimization*, Vol. 10, No. 1, 2008, pp. 121-134.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[8] C. Dodaro, M. Alviano, W. Faber, N. Leone, F. Ricca, and M. Sirianni, "The Birth of a WASP: Preliminary Report on a New ASP Solver", *Proceedings of the 26th Italian Conference on Computational Logic (CILC)*, CEUR Workshop Proceedings, 2011, pp. 99-113.

[9] C. Dodaro, M. Alviano, W. Faber, N. Leone, and F. Ricca, "WASP: A native ASP solver based on constraint learning", *Proceedings of the 12th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, LNCS, 2013, to appear.

[10] W. Faber, N. Leone, and G. Pfeifer, "Recursive Aggregates in Disjunctive Logic Programs: Semantics and Complexity", *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA)*, Lisbon, Portugal, September 27-30, 2004, pp. 200-212.

[11] O. Febbraro, K. Reale, and F. Ricca, "ASPIDE: Integrated Development Environment for Answer Set Programming", *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, Canada, May 16-19, 2011, pp. 317-330.

[12] G. Galbiati, A. Morzenti, and F. Maffioli, "On the Approximability of Some Maximum Spanning Tree Problems", *Theoretical Computer Science*, Vol. 181, No. 1, 1997, pp. 107-118.

[13] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub, "Conflict-driven answer set solving", *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, India, January 6-12, 2007, p. 386-392.

[14] M. Gelfond, and N. Leone, "Logic programming and knowledge representation – The A-Prolog perspective", *Artificial Intelligence*, Vol. 138, No. 1-2, 2002, pp. 3-38.

[15] G. Gottlob, N. Leone, and F. Scarcello, "Hypertree Decompositions: A Survey", *Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, Marianske Lazne, Czech Republic, August 27-31, 2001.

[16] G. Grasso, S. Iiritano, N. Leone, and F. Ricca, "Some DLV Applications for Knowledge Management", *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, Potsdam, Germany, September 14-18, 2009, pp. 591-597.

[17] G. Grasso, N. Leone, M. Manna, and F. Ricca, "ASP at Work: Spin-off and Applications of the DLV System", *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, LNCS, Vol. 6565, 2011, pp. 432-451.

[18] J. Gross, and J. Yellen. "Handbook of Graph Theory", CRC Press, 2004.

[19] J. Lee, and Y. Meng, "On Reductive Semantics of Aggregates in Answer Set Programming", *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, Potsdam, Germany, September 14-18, 2009, pp. 182-195.

[20] N. Leone, M. Manna, G. Terracina, and P. Veltri, "Efficiently Computable Datalog∃ Programs", *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, Italy, June 10-14, 2012, pp.13-23.

[21] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello, "The DLV system for knowledge representation and reasoning", *ACM Transactions of Computers Logic*, Vol. 7, No. 3, 2006, pp. 499-562.

[22] Y. Lierler, and M. Maratea, "Cmodels-2: SAT-based Answer Set Solver Enhanced to Non-tight Programs", Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR), FL, USA, January 6-8, 2004, pp. 346-350.

[23] M. Manna, F. Ricca, and G. Terracina, "Consistent query answering via ASP from different perspectives: Theory and practice", *Theory and Practice of Logic Programming*, Vol. 13, No. 2, 2013, pp. 227-252.

[24] F. Ricca, A. Dimasi, G. Grasso, S. M. Ielpa, and S. Iiritano, M. Manna, and N. Leone, "A Logic-Based System for e-Tourism", *Fundamenta Informaticae*, Vol. 105, No. 1-2, 2010, pp. 35-55.

[25] F. Ricca, W. Faber, and N. Leone, "A backjumping technique for Disjunctive Logic Programming", *AI Communications*, Vol. 19, No. 2, 2006, pp. 155-172.

[26] F. Ricca, G. Grasso, M. Alviano, M. Manna, V. Lio, S. Iiritano, and N. Leone, "Team-building with Answer Set Programming in the Gioia-Tauro Seaport", *Theory and Practice of Logic Programming*, Vol. 12, No. 3, 2012, pp. 361-381.