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 $\{1,2\}, \{1,4\}, \{6,7\}, \text{and} \{8,9\}$. 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 $G$.

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
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 $\Sigma$, and a conjunctive query $q$, the key decision problem is to determine whether $q$ is entailed by $D$ and $\Sigma$ (namely, whether $D \cup \Sigma \models q$). However, in case $\Sigma$ 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 $\text{len}_T(a,b)$, which of course is equivalent to $\text{len}_T(b,a)$. By definition, $\text{len}_T(a,a) = 0$.

The height (also known as diameter) of a tree $T$ over a set $V$ of vertices, denoted by $\text{height}(T)$, is the length of the longest path of $T$. More precisely, it is defined as follows: $\text{height}(T) = \max \{ \text{len}_T(a,b) \mid a,b \in V \}$. 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 $\text{height}(T) \geq \text{height}(T')$.

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 $O(n)$ 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:
loss of generality, assume that
a

with respect to the number of vertices of a tree.

Combining expression (2) and (3), we have:

\[ \text{length}(T,v) + \text{length}(T,u,v) < \text{length}(T,c,a) + \text{length}(T,c,b). \] (2)

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

\[ \text{length}(T,u,c) + \text{length}(T,c,a) \leq \text{length}(T,u,w). \] (3)

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

\[ \text{length}(T,v) < \text{length}(T,c,b) - \text{length}(T,u,c), \]

which is a contradiction since we know that \( \text{length}(T,c,b) \leq \text{length}(T,u,v) \). Therefore, such a path from \( a \) to \( b \) cannot exist.

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 \( T \).

\[ \text{length}(T,v,w) < \text{length}(T,a,b). \] (1)

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 \textit{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 \textit{term} is either a variable or a constant. A \textit{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 standard atoms, then \( p(t_1, \ldots, t_k) \) is a \textit{ground standard atom}. A \textit{set term} is either a symbolic set or a ground set. A \textit{symbolic set} is a pair \{\textit{Terms:Conj}\}, where \textit{Terms} is a list of terms, and \textit{Conj} is a conjunction of standard atoms. Intuitively, a set term \( X:a(Y,b(Y) \text{true}) \) stands for the set of \( X \)-values making the conjunction \( a(X,Y), b(Y) \text{true} \), namely \( \{X \mid \exists Y \text{ s.t. } a(X,Y), b(Y) \text{true} \} \). A \textit{ground set} is a set of pairs of the form \{\textit{consts:conj}\}, where \textit{consts} is a list of constants and \textit{conj} is a conjunction of ground standard atoms.

An \textit{aggregate function} is of the form \( f(S) \), where \( S \) is a set term and \( f \) is an \textit{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 \textit{aggregate atom} is a structure of the form \( f(S) \land T \), where \( f(S) \) is an aggregate function, \( \land \) is a comparison operator, and \( T \) is a term. If \( T \) is a constant and \( S \) is a ground set term, then \( f(S) \land T \) is a \textit{ground aggregate atom}. A \textit{literal} is a standard atom, or a standard atom preceded by the negation as failure symbol \texttt{not}, or an aggregate atom.

A rule \( r \) is an expression of the form:

\[ \alpha_1 \lor \ldots \lor \alpha_n \leftarrow \ell_1, \ldots, \ell_m, \]

where \( \alpha_1, \ldots, \alpha_n \) are standard atoms and \( \ell_1, \ldots, \ell_m \) are literals. The disjunction \( \alpha_1 \lor \ldots \lor \alpha_n \) is referred to as the \textit{head} of \( r \), while the conjunction \( \ell_1, \ldots, \ell_m \) is the \textit{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 \textit{ground rule}. A ground rule \( r \) is a \textit{fact} if both \( B(r) \) is empty and \( H(r) \) is a singleton. If

\[ \text{length}(T,u,c) + \text{length}(T,c,a) \leq \text{length}(T,u,w). \] (3)
local substitution for instances of all rules in $P$ is the ground set term \{$\gamma$\}, where $\gamma$ 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 $\gamma_P$, 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 $\sigma$ 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 $S = \{\text{Terms}: \text{Conj}\}$ without global variables, the instantiation of $S$, denoted by $\text{inst}(S)$, is the ground set term $\{\sigma(\text{Terms}): \sigma(\text{Conj})\} | \sigma$ is a local substitution for $S$. A ground instance of a rule $r$ is obtained in two steps: first, a global substitution $\sigma$ for $r$ is applied; then, every set term $S$ in $\sigma(r)$ is replaced by its instantiation $\text{inst}(S)$. The instantiation $\text{Ground}(P)$ 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 $\alpha$ is true with respect to $I$ if $\alpha \in I$; otherwise, $\alpha$ is false with respect to $I$. A negative standard ground literal not $\alpha$ is true with respect to $I$ if $\alpha \not\in I$; otherwise, not $\alpha$ is true with respect to $I$. Consider a set term $S$. Let $S'$ denote the set $\{\tau_1, \ldots, \tau_k : \text{Conj}\} \in S$, and all the atoms of Con are true with respect to $I$. The evaluation $I(\tau(S))$ 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 $f(S) \bigcirc k$ 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 $\text{Ground}(P)$ are satisfied with respect to $M$. A model $M$ of $P$ is (subset) minimal if no model $N$ of $P$ exists such that $N \subseteq M$. Let $P$ be a ground program, $I$ be an interpretation, and $P'$ 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 $\text{Ground}(P)'$. Finally, the answer sets of $P$ plus a set of weak constraints $\Omega$ 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 $\mu$ that encodes a graph as a set of facts, and we design a program $P$ and a weak constraint $\omega$ such that, for every graph $G$, $\mu(G)$ is a maximum height spanning tree of $G$ if and only if $\mu(G)$ 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 $\mu$, which associates a fact to each edge of $G$ as follows: $\mu(G) = \{\mu(u,v) | (u,v) \in E, u < v\}$. 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 $\{\text{vertex}(v) | v \in V\}$ (observe that, since $G$ is assumed to be connected, each vertex of $G$ appears in at least one edge of $G$):

\[
\text{vertex}(U) := \text{edge}(U,V).
\]

\[
\text{vertex}(V) := \text{edge}(U,V).
\]

Once we have the vertices of $G$, we can count them by using the built-in aggregate $\#\text{count}$ to
derive the atom \( \text{size}(n) \), where \( n = |V| \). This can be obtained via the following rule:

\[
\text{size}(N) :- \#\text{count}(V:\text{vertex}(V)) = N.
\]

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 \( \#\text{min} \) to derive the atom \( \text{first}(u_r) \), where \( u_r = \text{min}(V) \):

\[
\text{first}(Ur) :- \#\text{min}(V:\text{vertex}(V)) = Ur.
\]

All the atoms derived so far are obtained deterministically, and they are part of every answer set of \( \mu(G) U P U \{o\} \). 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 \( \mu(G) U P U \{o\} \). To this end, we introduce the next disjunctive rule to \( \text{pick} \) (or \( \text{guess} \)) from \( E \) different sets of edges, each of which forms a candidate maximum height spanning tree of \( G \):

\[
\text{pick}(U,V) \lor \text{discard}(U,V) :- \text{edge}(U,V).
\]

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 \( \text{check} \) whether \( S \) satisfies or not our desiderata. In the former case, \( S \) will rise to one of the answer sets of \( \mu(G) U P U \{o\} \). In the latter case, \( S \) will be ignored instead. Let us now fix one possible set of edges \( S \) and consider the set \( \{\text{pick}(u,v) \mid \{u,v\} \in S, u < v\} \) 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 \( \mu(G) U P U \{o\} \).

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

\[
\text{reachable}(Ur) :- \text{first}(Ur).
\]

\[
\text{reachable}(V) :- \text{reachable}(U), \text{pick}(U,V).
\]

\[
\text{reachable}(U) :- \text{reachable}(V), \text{pick}(U,V).
\]

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 \( u_r \):

\[
:- \text{vertex}(V), \text{not reachable}(V), \text{not reach from ur}.
\]

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:

\[
:- \#\text{count}(U,V: \text{pick}(U,V)) > N.
\]

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 \( \text{sEdge}(u,v) \) and \( \text{sEdge}(v,u) \), which allow us to treat each edge of \( C \) as a set of two vertices:

\[
\text{sEdge}(U,V) :- \text{pick}(U,V).
\]

\[
\text{sEdge}(V,U) :- \text{pick}(U,V).
\]

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

\[
\text{trvl}(U,V,1) :- \text{first}(U), \text{sEdge}(U,V).
\]

\[
\text{trvl}(V,W,D1) :- \text{trvl}(U,V,D), \text{sEdge}(V,W), U<W, D1 = D + 1.
\]

In particular, the meaning of each atom of the form \( \text{trvl}(u,v,d) \) is that the distance between \( u_r \), and \( v \) is \( d \), namely \( d = \text{length}(u_r,v) \). 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 \( u_r \):

\[
\text{farl}(Vr) :- Dr = \text{max}(D: \text{trvl}(r_,D)), Vr = \#\text{min}(V:\text{dist1}(r_,V,Dr)).
\]

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 \( C \):

\[
\text{trvl}(Vr,V,1) :- \text{farl}(Vr), \text{sEdge}(Vr,V).
\]

\[
\text{trvl}(V,W,D1) :- \text{trvl}(U,V,D), \text{sEdge}(V,W), U<W, D1 = D + 1.
\]
Similarly to the first traversal, the meaning of each atom of the form \( \text{trv2}(u,v,d) \) is that the distance between \( v \) and \( u \) is \( d \) and that \( u \) is the vertex that precedes \( v \) in the path from \( v \) to \( u \).

The next rule selects the maximum distance, say \( d_m \), between \( v \), and the other vertices of \( C \):

\[
\text{maxDist}(Dm) :- \text{Dm} = \#\text{max}\{D:\text{trv2}(\_\_,\_\_,D)\}.
\]

Finally, we define the weak constraint \( \omega \), which guarantees that \( S \) (and therefore \( C \)) is not ignored only if its height \( d_m \) is the max tree-height of \( G \). Actually, \( \omega \) 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 \( P \):

\[
\text{size}(N), \text{maxDist}(Dm), V=N-Dm. [V]
\]

As previously stated, the answer sets of the program \( \mu(G) \cup P \) enhanced with \( \omega \) 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 \( \text{maxDist}(d_m) \).

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 \( \text{PTIME} = \text{NP} \).

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.

REFERENCES:


