Verification Programs for Abduction

Paolo Liberatore[1] and **Francesco M. Donini**[2]

Abstract. We call *verification* the process of finding the actual explanation of a given set of manifestations. We consider an abductive setting, in which explanations are sets of assumptions. To filter out erroneous explanations, a verification program should propose which assumptions to check. Given the abductive setting of manifestations, assumptions, and a theory relating them, we study the complexity of providing a *minimal set* of assumptions to be checked in order to identify the actual explanation. We study also the case in which assumptions to be checked are given in a tree-like order.

1 INTRODUCTION

Abduction [8] has been frequently advocated as a process apt to formalize *diagnosis* — i.e., the way reasoning is used when looking for (a set of possible) causes for some observed manifestations [9, 10, 2].

In a diagnostic setting, we are given a set of manifestations M observed in the (mal)functioning of a system, a (usually large) set of assumptions H that may or may not explain M, and some relation T linking H to M. In the abductive framework, T is a theory expressed in some logic. Such T can be a logical account of how the system works, or a set of causal relations between H and M [5], etc. Usually, an explanation E is a subset of all assumptions H.

Automated support in diagnosis is helpful for searching and providing some explanation E. However, in a diagnostic setting, E is needed to decide how to *take repair*. Hence, once E is given, one should first check if E is the *actual* explanation, i.e. if the assumptions in E are true. If not, another explanation must be found and checked, and so on.

We believe that automated support to diagnosis should also keep up with *verification* — the process of enumerating and discarding one after the other all possible explanations, till the actual one is isolated. Since we consider explanations to be sets of assumptions, they may share assumptions. Therefore, a system should also provide a *minimal* set of assumptions to check, which, once checked, would determine *the* actual explanation. We call such a set a *Verification Set-Program*. Moreover, the system could suggest which assumptions should be checked first (e.g. most discriminating ones), to come as quickly as possible to the right explanation. In this case, assumptions to be checked could arranged in a *Verification Tree-Program*.

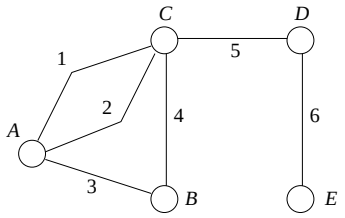
1.1 Abduction

An instance of abduction is a triple $\langle H, M, T\rangle$, where H and M are sets of propositional variables, while T is a theory (set of propositional formulae). The set of explanations of $\langle H, M, T\rangle$ is defined as follows:

$$SOL(H, M, T) = \{E \subseteq H \mid E \cup T \text{consistent, and } E \cup T \models M\}$$

In many cases, not all explanations are equally plausible. It is customary to consider explanations containing the least number of assumptions as the most likely ones. This corresponds to the case in which all assumptions are given the same probability, and assumptions are independent. Given an ordering $\preceq$ over the subsets of H, the set of minimal solutions $SOL_{\preceq}(H, M, T)$ of the problem is defined as $\min(SOL(H, M, T), \preceq)$. In this paper we only refer to the ordering based on cardinality, that is, $E_1 \preceq E_2$ if and only if $|E_1| \leq |E_2|$. Since this corresponds to the special case of equal weight/probability of all assumptions, all of our hardness results carry over to the case of weighted assumptions.

Example 1 *Faults in computer networks are usually easy to detect, but may be hard to locate exactly. Let us consider the layout aside. A, B, C, D, and E denote some computers connected to this network. We are logged on computer A, and see that neither B nor E are reachable, which means that all paths from A to B and from A to E contain a faulty link.*



This problem can be formalized as an instance of abduction $\langle H, M, T\rangle$, as follows. Let h_i denote the fact that link i is faulty, while m_X denotes the unreachability of computer X from A.

$$\begin{aligned} H &= \{h_1, h_2, h_3, h_4, h_5, h_6\} \\ M &= \{m_B, m_E\} \\ T &= \{\neg h_1 \rightarrow \neg m_C, \neg h_2 \rightarrow \neg m_C, \neg h_3 \rightarrow \neg m_B, \\ &\quad \neg h_4 \rightarrow (m_B \equiv m_C), \\ &\quad \neg h_5 \rightarrow (m_C \equiv m_D), \neg h_6 \rightarrow (m_D \equiv m_E)\} \end{aligned}$$

Formula T is a set of clauses like $\neg h_i \rightarrow (m_X \equiv m_Y)$ corresponding to an edge i between computer X and Y, meaning that if the link is not fault, then the reachability of X is the same as the reachability of Y. It is not hard to find that there three possible causes of the problem: 1. links 1, 2, and 3 are fault; 2. links 3, 4, and 5 are fault; 3. links 3, 4, and 6 are fault. These cases correspond to the following set of explanations:

$$SOL_{\preceq}(\langle H, M, T\rangle) = \{\{h_1, h_2, h_3\}, \{h_3, h_4, h_5\}, \{h_3, h_4, h_6\}\}$$

There are other possibilities (for instance, all links are faulty); however, the above three are the ones that presume a minimal number of faults.

[1] Dipartimento di Informatica e Sistemistica, Univ. di Roma "La Sapienza", Via Salaria 113, 00198 Roma, Italy. http://www.dis.uniroma1.it/~liberato

[2] Dipartimento di Elettrotecnica ed Elettronica, Politecnico di Bari, Via Re David 200, 00175 Bari, Italy. http://dee.poliba.it/dee-web/doniniweb/donini.html

The paper is organized as follows. In the next section, we introduce verification programs. Then in Section 3 we study the complexity of verification programs as sets. In Section 4 we study the impact of imposing a tree ordering on verification programs. Finally we draw some conclusions. For definitions of the complexity classes we use, the reader is referred to [7].

2 VERIFICATION PROGRAMS

An instance of abduction may admit more than one possible explanation. Even using some ordering based on plausibility, it is likely we end up with more than one (in some cases, exponentially many) possible explanations — as the above example shows. The aim of verification is to determine which is the actual set of assumptions that causes a given set of manifestations to happen.

We assume that we can directly test whether an assumption is true or not. A possible future research is the study of the case in which assumptions cannot be directly checked, and results of tests are the truth values of complex formulas. In order to decide which explanation is the actual one, we can run a verification program.

Definition 1 (Verification Program) *A* Verification Program *is a process aimed at determining a unique explanation, by checking the truth value of some assumptions.*

We want to minimize the number of assumptions that have to be checked during verification. In this paper we consider two possible ways of formalizing such concept. The first one are verification *set*-programs, which are sets of assumptions that, once checked, allow for uniquely determining an explanation. The second one are verification *tree*-programs, in which we first check an assumption, and then use its truth value to decide the next assumption to check.

Example 1 (Cont.) *The diagnosis of the computer network still left us with the problem of establishing which the actual cause of the problem is. In practice, giving a set of possible explanations is not enough: eventually, the problem should be fixed. In order to accomplishing this aim, we should first uniquely identify the faults (that is, a single explanation).*

This can be obtained by verifying each single link in the network. However, while checking reachability of computers is easy (can be done without moving from the console of A), checking links requires some time. This means that we should try to perform as few tests as possible. In this case, checking link 3 is useless, as it is faulty in all explanations (i.e., it must be faulty). On the other hand, checking links 1, 5, *and* 6 *is sufficient to uniquely determine an explanation. Indeed, if link* 1 *is faulty, the explanation is* $\{h_1, h_2, h_3\}$*. If link* 5 *is faulty, the explanation is* $\{h_3, h_4, h_5\}$*, and similarly for the case in which* h_6 *is true. The question is: are at least three tests necessary, or would two suffice?*

3 VERIFICATION SET-PROGRAMS

Informally, a verification set-program for an instance of abduction is a set of assumptions P such that every truth value of P determines at most one explanation.

Definition 2 *A verification set-program for an instance* $\langle H, M, T\rangle$ *is a subset* $P \subseteq H$ *such that, for every truth evaluation* σ_P *of the variables in* P*, there exists at most one explanation* $E \in SOL_{\preceq}(\langle H, M, T\rangle)$ *such that for all* $h_i \in P$*:*

$h_i \in E \;\Leftrightarrow\; \sigma_P(h_i) = \mathsf{true}$

If we could verify all assumptions, then we would find the real cause of the manifestations without uncertainty, that is, the set of all assumptions is always a verification set-program. However, verifying assumptions is a costly task — if it is not, there is no reason for using abduction, as we can directly verify what happened. As a result, what is needed is a verification set-program containing the minimal number of assumptions to verify. More refined definitions can be given (e.g., assigning weights to assumptions), but for now we consider as measure of optimality the number of assumptions to verify.

In the literature, two special types of assumptions have been studied: necessary (those belonging to every explanation) and relevant (those in at least one explanation) [3]. Let us now relate set-programs with the sets of relevant and necessary assumptions. Clearly, non-relevant assumptions are not in any minimal verification set-program. The same holds for necessary assumptions: a necessary assumption is in every explanation, thus its truth gives no information about which explanation is the right one. The discriminating assumptions are among the relevant, but not necessary, ones.

Theorem 1 *The set of relevant and unnecessary assumptions is a verification set-program.*

However, such a set might not be a minimal verification set-program. Let the set of minimal solutions be:

$$SOL_{\preceq}(\langle H, M, T\rangle) = \{\{h_1, h_2\},\ \{h_2, h_3\},\ \{h_3, h_4\},\ \{h_2, h_4\}\}$$

The set of relevant and unnecessary assumptions is $\{h_1, \ldots, h_4\}$. However, there is exactly one minimal verification set-program: $\{h_3, h_4\}$. It is easy to see that, given the value of these two assumptions, we can uniquely determine the actual explanation. It can also be proved that no other set program composed of two assumptions is a minimal verification set-program. As a result, h_1 and h_2 are not part of any minimal set-program, while they are both relevant and not necessary.

We remark that if non-minimal explanations are taken into account, then the only possible set-program is the set of all non-necessary assumptions.

We show the definition of verification set programs on an example.

Example 1 (Cont.) *Let us consider the faulty network again. What is important is the set of explanations, that are* $E_1 = \{h_1, h_2, h_3\}$*,* $E_2 = \{h_3, h_4, h_5\}$*, and* $E_3 = \{h_3, h_4, h_6\}$*. It is not hard to prove that* $\{h_1, h_5\}$ *is a verification set-program.*

Indeed, once checked whether links 1 *and* 5 *are faulty or not, we can determine the explanation as in the table. The fourth case, in which both* 1 *and* 5 *faulty, is inconsistent with our hypothesis of minimality of explanations.*

h_1	h_5	*explanation*
0	0	E_3
0	1	E_2
1	0	E_1
1	1	*inconsistent*

In all other cases, given a truth value of h_1 *and* h_5*, the explanation is uniquely determined. As a result,* $\{h_1, h_5\}$ *is a verification set-program. Other minimal verification set programs for the same instance are* $\{h_2, h_5\}$*,* $\{h_1, h_6\}$*, and* $\{h_2, h_6\}$*.*

Hitting sets provide an equivalent definition of verification set-programs. This formulation is useful because — apart from giving a better insight — it leads to simpler proofs in some cases. We first recall what is a hitting set.

Definition 3 *Let* $\mathcal{S} = \{S_1, \ldots, S_m\}$ *be a family of sets. A set* P *is a* hitting set *of* $\mathcal{S}$ *if* $P \cap S_i \neq \emptyset$ *for* $i = 1, \ldots, m$*.*

Note that, in spite of the similarity between verification set-programs and hitting sets, these two concepts are not exactly the same. Namely, verification set-programs are not hitting sets of the set of explanations. Indeed, let the set of explanations be $\{\{h_1, h_2\}, \{h_2, h_3\}\}$. The only minimal hitting set is $\{h_2\}$, while there are two minimal verification set-programs, namely $\{h_1\}$ and $\{h_3\}$.

Let Δ be the symmetric difference between two sets, that is, $A\Delta B = A\backslash B \cup B\backslash A$. A verification set-program should distinguish every explanation from every other one. Two explanations are distinguished by those assumptions which are in either one, but not in both of them.

Theorem 2 *For any given set of explanations $\{E_1, \ldots, E_m\}$, a set of assumptions P is a verification set-program if and only if P is a hitting set of the family $\{E_i \Delta E_j : i \neq j\}$, i.e., for any pair of explanations E_i, E_j with $i \neq j$, it holds $P \cap (E_i \Delta E_j) \neq \emptyset$.*

Proof. Let us assume that V intersects any symmetric difference between two explanations. We prove that V is a verification set-program. Given a truth assignment over V we can determine a unique explanation by the following algorithm.

$P = E_1$;
for each i in $\{2, \ldots, m\}$ **do:**
    v is an assumption in $V \cap (P \Delta E_i)$
        (this set is nonempty by assumption);
    the truth value of v allows ruling out one explanation
        between P and E_i: set P to the remaining one.
return P

In other words, we proceed by comparing pairs of explanations. Since V intersects any symmetric difference between two explanations, a truth evaluation over the variables in V allows for choosing an explanation in any pair. Namely, if v is in P, then if it is assigned to true then E_i is ruled out. If it is false then P can be removed. The converse holds if v is in E_i.

We now prove that, if there is a pair of explanations E_i and E_j such that $P \cap (E_i \Delta E_j) = \emptyset$, then P is not a verification set-program. Let us consider the truth assignment that maps each variable in $P \cap E_i \cap E_j$ into true, and any other variable in P into false. Both E_i and E_j are compatible with this assignment, since all elements of P that are in E_i or in E_j are mapped to true, and no variable appearing only in E_1 or in E_2 is in P. As a result, P is not a verification set-program. □

This theorem is useful because it gives a characterization of verification set-programs that is not based on truth assignments and does not involve determining a unique element, but is only based on comparisons between sets.

Let us consider the computational complexity of determining a minimal verification set-program. A simple idea could be just enumerating all minimal explanations, and then, using Theorem 2, finding a hitting set of the symmetric differences of all pairs of explanations. Of course, this might be an overkill if the complexity of finding a minimal verification set-program were less than the one of hitting sets. Hence, we first consider the easiest case — which is almost not even related to abduction — in which we are given explicitly the set of explanations.

In order to determine the complexity of finding minimal verification set-programs, we employ a reduction from the problem of finding a minimal vertex cover of a graph. The following lemmas show that such a reduction is possible.

Given a graph $G = (N, E)$, let $\mathcal{E}$ be the following set of explanations on the set of assumptions $H = N \cup E$.

$$\mathcal{E} = \{\{e_1, e_2\} \mid e_1, e_2 \in E\} \cup \{\{e, n\}, \{e, m\} \mid e = (n, m) \in E\}$$

Lemma 1 *If P is a verification set-program for $\mathcal{E}$ then $P \cap N$ is a vertex cover of G.*

Lemma 2 *If V is a vertex cover of G then $V \cup E$ is a verification set-program for $\mathcal{E}$.*

Note that the above two lemmas *do not* establish a one-to-one correspondence between vertex covers of G and verification set-programs of $\mathcal{E}$. Indeed, a vertex cover V may correspond to many verification set-programs $P_1, P_2, \ldots$ such that $P_i \cap N = V$, but with different elements of E.

With two reasonable restrictions, we can prove that the correspondence between some vertex covers and the corresponding verification set-programs is one-to-one.

Lemma 3 *If V is a vertex cover for a graph $G = (N, E)$ with no isolated vertex and $|N\backslash V| \geq 2$, then for every $E' \subset E$, $V \cup E'$ is not a verification set-program for $\mathcal{E}$.*

In other words, removing elements of E from $V \cup E$, one loses the verification set-program property.

In this case, we are able to prove that minimal vertex covers correspond to minimal verification set-programs.

Theorem 3 *Let $G = (N, E)$ be a graph with no isolated vertex, whose minimal vertex covers contain at most $|N| - 2$ nodes. Then V is a minimal vertex cover of G if and only if $P = V \cup E$ is a minimal verification set-program for $\mathcal{E}$.*

Proof. We divide the proof in two parts.

1. If V is a vertex cover then P is a verification set-program by Lemma 2. If V is also a minimal vertex cover, then no $V' \subset V$ can be a vertex cover. As a result, if $P' = V' \cup E$, then P' is not a verification set-program for $\mathcal{E}$ by Lemma 3. What is left to prove is that $P' = V \cup E'$ is not a verification set-program for every $E' \subset E$. This is a consequence of Lemma 3.
2. If P is a verification set-program then $P \cap N$ is a vertex cover, and $P \cap E = E$. If no subset of P is a verification set-program for $\mathcal{E}$ then no subset of V is a vertex cover.

We can then conclude that there is a one-to-one correspondence between minimal vertex covers of G and minimal verification set programs of $\mathcal{E}$. □

The assumption that G does not contain isolated nodes and that has a vertex cover composed of at most $|N| - 2$ nodes does not change the complexity of the vertex cover problem.

Theorem 4 *Given an integer k and a set of explanations $\mathcal{E}$, deciding whether there exists a verification set-program with no more than k assumptions for $\mathcal{E}$ is* NP *complete.*

Proof. Membership in NP is proved by Theorem 2, which shows a reduction from verification set-programs to hitting sets. Hardness is proved by Theorem 3, showing a reduction from vertex covers to verification set-programs. □

The complexity of finding a minimal verification set-program can be proved in the same manner, still assuming that the set of explanations are given exhaustively.

Theorem 5 *Given a set of explanations $\mathcal{E}$, finding a minimal verification set-program for $\mathcal{E}$ is $F\Delta_2^p[\log n]$ complete.*

The analysis of the case when T is a Horn theory shows an increase of complexity w.r.t. the case in which explanations are given exhaustively. We summarize the results we proved.

Theorem 6 *The complexity of the following problems, when T is a Horn theory, is:*
checking whether a set P is a verification set-program: $\Delta_2^p[\log n]$*-complete;*
existence of a verification set-program composed of at most t assumptions: $\Delta_2^p[\log n]$*-hard, in* Σ_2^p*;*
finding a minimal verification set-program: in $F\Delta_3^p$.

4 TREE-PROGRAMS

Even minimal set-programs may in some case require testing assumptions that are not really useful to the extent of uniquely determining the real explanation. For instance let the explanantions be $\{\{h_1, h_2\}, \{h_2, h_3\}, \{h_1, h_3\}\}$. A minimal verification set-program is $\{h_1, h_2\}$. Indeed, the explanation can be determined from the value of h_1 and h_2 as shown in the table. Given the value of h_1 and h_2, it is possible to determine a single explanation. The fact that $\{h_1, h_2\}$ is a minimal set-program is due to the fact that there are three minimal explanations, while set-programs composed of only one variable have only two possible truth assignment.

h_1	h_2	explanation
false	false	none
false	true	$\{h_2, h_3\}$
true	false	$\{h_1, h_3\}$
true	true	$\{h_1, h_2\}$

Verifying two assumptions we can uniquely determine the explanation. However, there is a case in which one assumption only is enough. Let for instance assume that we check h_1 first and then h_2. If h_1 is true, then h_2 is needed to determine the explanation. However, if h_1 is false, we already know that the explanation is $\{h_2, h_3\}$ without checking the value of h_2.

The idea of verification tree-programs is that assumptions are verified in some order, and the next choice may depend on the result of the previous tests. For instance, we may check h_1 first, and then, if $h_1 =$ true we test h_2, otherwise we test h_3. We can represent such programs with trees whose nodes are labeled with the verified assumptions.

Definition 4 *A* tree-program *is a binary tree whose nodes are labeled with assumptions, and edges are labeled with* true *or* false*, such that, for each path from the root to a leaf, no assumption appears twice, and the outgoing edges of a node have different labels.*

As a result, the leaves of a tree-program are associated to partial interpretations over the alphabet of assumptions. If such interpretations allow one to determine unique explanations, then we have a verification tree-program.

Definition 5 *A* verification tree-program *is a tree-program in which the truth value of the assumptions in each leaf allows identifying exactly one explanation.*

Minimal tree-programs are tree-programs having minimal depth. A tree-program is a decision tree [4] on the set of explanations.

Example 1 (Cont.) *A unique explanation for the computer network can be determined using verification tree-programs. Let us represent trees using the parenthetic notation, assuming that the left subtree is the one in which the root is assigned to* false*. The tree $(h_1(h_5)())$ is a verification tree program for that instance. Indeed, if h_1 is false, we have to check h_5, and the explanation is either E_2 or E_3, depending on the result of this test. If h_1 is true, we conclude that the explanation is E_1.*

It can be proved that the depth of minimal verification tree-programs may be less than the size of minimal verification set-programs, which means that using tree-programs instead of set-programs we always test less assumptions.

Let $H = \{h_1, \ldots, h_{2n}\}$ be a set of assumptions. We consider the following set of explanations, all of cardinality n.

$$\mathcal{E} = \big\{\{h_1, \ldots, h_i, h_{n+i+1}, \ldots, h_{2n}\} \mid 0 \leq i \leq n\big\}$$

In other words, for any i we have an explanation E_i composed of the first i assumptions $h_1, \ldots, h_i$, while the other $n - i$ ones are the last assumptions $h_{n+i+1}, \ldots, h_{2n}$.

Lemma 4 *The minimal verification set-programs of $\mathcal{E}$ have size n.*

Proof. Clearly, $\{h_1, \ldots, h_n\}$ is a verification set-program for $\mathcal{E}$. Let us now prove that no verification set-program can contain less than n assumptions. Assume, by contradiction, that P is a verification set-program, and that $|P| < n$. A consequence of this assumption is that there exists an index i such that $h_i, h_{n+i} \notin P$. Consider the following pair of explanations.

$$\begin{aligned} E_{i-1} &= \{h_1, \ldots, h_{i-1}, h_{n+i}, \ldots, h_{2n}\} \\ E_i &= \{h_1, \ldots, h_i, h_{n+i+1}, \ldots, h_{2n}\} \end{aligned}$$

It holds $E_{i-1} \Delta E_i = \{h_i, h_{n+i}\}$. Since neither h_i nor h_{n+i} is in P, it follows that $(E_{i-1} \Delta E_i) \cap P = \emptyset$, thus contradicting Theorem 2 for verification set-programs. □

On the other hand, there are verification tree-programs for $\mathcal{E}$ having a logarithmic depth.

Lemma 5 *The tree $(h_{n/2}(h_{n/4}\ldots)(h_{3n/4}\ldots))$ is a verification tree-program for $\mathcal{E}$.*

This pair of lemmas prove that the size of verification set-programs may be exponentially larger than the depth of the optimal verification tree-programs.

Corollary 7 *There are sets of explanations for which the size of all verification set-programs is exponentially larger than the depth of its optimal verification tree-programs.*

Finally, we note that, even for very simple theories T, the number of explanations may be exponentially large, which implies that the *total size* of a verification tree-program is exponential (the *depth* is of course bounded by the number of assumptions $|H|$).

The fact that tree-programs may be exponential is not always a problem. What we want is the right explanation of a given set of manifestations, and this can be done iteratively as follows:

while there is more than one explanation **do:**
  find an assumption that is the root of a
    minimal verification tree-program;
  verify whether this assumption is true or false;
  set the value of the assumption in T

Let us now consider the complexity of problems related to verification tree-programs. In the case in which explanations are given explicitly (that is, the input of the problem is a set of explanations), verification tree-programs are binary decision trees. Hence, deciding whether there exists a verification tree-program of a given depth is NP-complete [4].

In the case in which the input is a real abduction instance $\langle H, M, T\rangle$, what makes the problem complex is that even minimal tree-programs may be exponentially large. Let us for example consider the problem of finding the root of an optimal tree-program. Guessing a verification tree-program with a given assumption as root does not work, since it requires guessing a tree that may be exponentially large. Given this premise, the upper bound we proved may look surprisingly simple.

Theorem 8 *Let $\langle H, M, T\rangle$ be an instance of abduction, where T is a propositional theory. Deciding whether an assumption is the root of an optimal verification tree-program for $SOL_{\preceq}(\langle H, M, T\rangle)$ is in* PSPACE.

Note that no condition over T is assumed, that is, T can be any formula (even a non-Horn one). The same upper bound holds for the related problems of deciding whether there exist verification tree-programs of depth k or less, and to actually find the root of an optimal verification tree-program.

Regarding hardness, we show two results. The first one refers to a very simple form for theory T, while the second one is about Horn theories.

Theorem 9 *Deciding whether there exists a verification tree-program for a given instance of abduction $\langle H, M, T\rangle$ whose depth is k or less, is* NP*-hard. This result holds even if T is a conjunction of implications of the form $h \rightarrow m$, where $h \in H$ and $m \in M$.*

A slightly stronger result can be given for the case in which T is assumed to be in Horn form.

Theorem 10 *Deciding whether there exist a verification tree-program for a given instance of abduction $\langle H, M, T\rangle$ whose depth is k or less, is $\Delta_2^p[\log n]$-hard. This result holds even if T is a Horn theory.*

5 DISCUSSION

In this paper we investigated verification, which is the process for uniquely determine the explanation of a given set of manifestations, in an abductive framework. We considered verification programs, to be accomplished by checking the truth of assumptions of candidate minimal explanations. We considered verification set-programs (no hint on which assumptions are to be checked first) and verification tree-programs (a tree tells which assumption to check next, depending on the result of the last check). Verification tree-programs are the natural extension of decision trees to find the right explanation in a diagnostic setting. Since verification tree-programs can have exponential size, it makes sense to study verification set-programs, which have size always less than or equal to the total number of assumptions. We showed the complexity of finding an optimal verification set-program for the cases in which explanations are explicitly enumerated, and found close upper and lower complexity bounds for the case in which the abductive theory is Horn. We also studied the case for verification tree-programs, for which we gave a general polynomial-space algorithm, and intractability results when the theory contains just direct implications between assumptions and manifestations. An impact of our results is that it is now possible to use custom algorithms and techniques for, e.g. vertex cover, to decide the tests to be performed in looking for an explanation. For more complex problems, like finding the root of an optimal tree-program, techniques for solving general PSPACE problems can be used [1]. We are now considering special cases in which providing verification programs could be simpler. However, high complexity in computing a verification program might not be a problem, if we assume that it can be computed off-line.

Finally, let us relate our work with that of McIlraith [6], who introduced a framework in which abduction is used for finding tests in model-based diagnosis. There are three main differences between our definitions and hers. First, while we give a set of definitions of the tests needed to find the correct explanation for an instance of abduction, McIlraith starts from an instance of a model-based diagnosis, and uses abduction as a technical tool for finding the set of tests needed to identify the problem. As such, our framework is suited for problems that can be formalized as abduction, while hers is useful for problems that can be encoded in a model-based diagnosis framework. Second, McIlraith does not consider the minimal number of tests to perform in finding the actual diagnosis, nor the way tests are performed (unordered vs. ordered, as our set-programs and tree-programs): her definition of "minimal" set of test is based on local minimality, that is, irredundance (a set of tests is minimal if no test can be neglected from the set). Third, the way in which test themselves are formalized is different: intuitively, our checks on assumptions would be rephrased in her notation as $(\{\}, h)$, i.e. tests with no achievables.

ACKNOWLEDGMENTS

The authors thank the referees for their comments. This work has been partially supported by CNR.

REFERENCES

[1] M. Cadoli, A. Giovanardi, and M. Schaerf, 'An algorithm to evaluate quantified boolean formulae', in *Proc. of AAAI'98*, pp. 262–267. AAAI Press/The MIT Press, (1998).

[2] L. Console and G. Friedrich, eds. *Model-based Diagnosis*. Ann. of Mathematics and Artificial Intelligence, 1994.

[3] T. Eiter and G. Gottlob, 'The complexity of logic-based abduction', *J. of the ACM*, **42**(1), 3–42, (1995).

[4] L. Hyafil and R. Rivest, 'Constructing optimal binary decision trees is NP-complete.', *Information Processing Letters*, **5**(1), 15–17, (1976).

[5] K. Konolige, 'Using default and causal reasoning in diagnosis', *Ann. of Mathematics and Artificial Intelligence*, **11**, (1994).

[6] S. McIlraith, 'Generating tests using abduction', in *Proc. of KR'94*, pp. 449–460, (1994).

[7] C. H. Papadimitriou, *Computational Complexity*, Addison Wesley, Reading, MA, 1994.

[8] C. S. Peirce, 'Abduction and induction', in *Philosophical Writings of Peirce*, ed., J. Buchler, chapter 11, Dover, New York, (1955).

[9] Y. Peng and J. Reggia, 'Plausibility of diagnostic hypothesis', in *Proc. of AAAI'86*, pp. 140–145, (1986).

[10] R. Reiter, 'A theory of diagnosis from first principles', *Artificial Intelligence*, **32**, 57–96, (1987).