Artificial Intelligence 65 (1994) 1-27
Elsevier

ARTINT 1034

A linear constraint satisfaction approach to cost-based abduction

Eugene Santos Jr

Department of Electrical and Computer Engineering, Air Force Institute of Technology, AFIT/ENG, 2950 P Street, Wright-Patterson AFB, OH 45433-7765, USA

Received May 1991
Revised October 1992

Abstract

Santos Jr, E., A linear constraint satisfaction approach to cost-based abduction, Artificial Intelligence 65 (1994) 1-27.

Abduction is the problem of finding the best explanation for a given set of observations. Within AI, this has been modeled as proving the observation by assuming some set of hypotheses. Cost-based abduction associates a *cost* with each hypothesis. The best proof is the one which assumes the least costly set. Previous approaches to finding the least cost set have formalized cost-based abduction as a heuristic graph search problem. However, efficient admissible heuristics have proven difficult to find. In this paper, we present a new technique for finding least cost sets by using linear constraints to represent causal relationships. In particular, we are able to recast the problem as a 0-1 integer linear programming problem. We can then use the highly efficient optimization tools of operations research yielding a computationally efficient method for solving cost-based abduction problems. Experiments comparing our linear constraint satisfaction approach to standard graph searching methodologies suggest that our approach is superior to existing search techniques in that our approach exhibits an expected-case polynomial run-time growth rate.

1. Introduction

Abductive explanation has been formalized in AI as the process of searching for some set of assumptions that can prove the things to be explained [2,3,5,9,10,14-16,19-21]. We call each such set an *explanation* for the

Correspondence to: E. Santos Jr, Department of Electrical and Computer Engineering, Air Force Institute of Technology, AFIT/ENG, 2950 P Street, Wright-Patterson AFB, OH 45433-7765, USA. E-mail: esantos@afit.af.mil.

SSDI 0004-3702(92)00061-D

given evidence. For example, consider the following situation: "John visits his friend Mary's house and finds that the place is dark and quiet. He concludes that Mary is not home." John's conclusion is a form of abductive explanation and cannot be arrived at by purely deductive means.

The information John used to arrive at his conclusion can be described with the following set of propositions:

$$\begin{aligned}
&\text{house-dark} \wedge \text{house-quiet} &&\Longrightarrow \text{house-dark-quiet},\\
&\text{lights-out} &&\Longrightarrow \text{house-dark},\\
&\text{no-one-home} \vee \text{blackout} &&\Longrightarrow \text{lights-out},\\
&\text{tv-off} \wedge \text{radio-off} &&\Longrightarrow \text{house-quiet},\\
&\text{no-one-home} \vee \text{no-shows} \vee \text{blackout} &&\Longrightarrow \text{tv-off},\\
&\text{no-one-home} \vee \text{bad-songs} \vee \text{blackout} &&\Longrightarrow \text{radio-off},
\end{aligned}$$

where "$\wedge$", "$\vee$", and "$\Longrightarrow$" denote conjunction, disjunction, and implication, respectively. The abductive reasoning task can be viewed as a backward-chaining process on the propositions. In essence, we are traveling backwards through the implications in hopes of finding a set of assumptions which can serve as an explanation for the evidence. For example, assuming that no one is home is a possible explanation for the house being dark and quiet.

A basic problem which naturally arises in abductive reasoning is that there may be many different possible explanations available. Using traditional symbolic logic, the only measure of a set's viability as an explanation is whether or not the evidence can be deductively inferred from the set. Thus, even the most far-fetched set of assumptions can be a possible candidate as long as it proves the evidence. In the above example, the house may be dark and quiet because of a blackout which in general is a slightly less plausible possibility. A related, but slightly different problem concerns the explanation whereby John simply assumes that the house is dark and quiet. This is a perfectly legitimate answer but provides no useful information.

We can easily see that some preferential ordering on the explanations is necessary. Early measures based on minimizing the necessary number of hypotheses [5,10] have been shown to be inadequate [3,9,21] suggesting the use of a more sophisticated approach. One such approach proposed by Hobbs and Stickel [1,9,21], called *weighted abduction*, involves levying numerical costs on individual assumptions. The cost of an explanation is a function of the cost of the individual assumptions made in the explanation. These costs are used in an effort to guide the abductive system towards the intended explanations. The final choice for best explanation will be the one with *least cost.*

Here, we will consider a minor variant of weighted abduction called *cost-based abduction* presented in [3]. It has been shown in [3] that belief revision in Bayesian networks [13] can be accurately modeled by cost-based abduction.

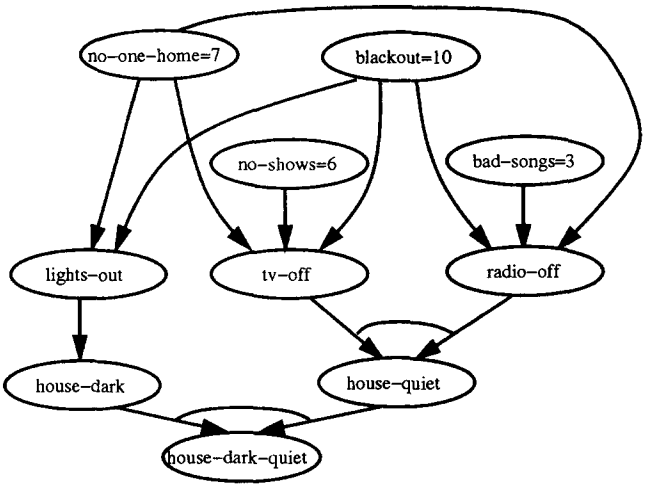

Fig. 1. A simple WAODAG. The AND-node house-dark-quiet is the observation. The nodes no-one-home, no-shows, blackout, and bad-songs are the hypotheses with associated costs 7, 6, 10, and 3, respectively. The assignment of {no-one-home} to true and {bad-songs,blackout,no-shows} to false results in lights-out, radio-off, tv-off, house-dark, and house-quiet to be true. This proof has a cost of 7 and is the minimal cost proof.

In cost-based abduction, hypotheses have associated costs, and the cost of a proof is simply the sum of the costs of the hypotheses required to complete that proof. Examples of such proofs can be found in [2,3]. Central to this approach is the use of *directed acyclic graphs* called WAODAGs (or, weighted AND/OR directed acyclic graphs) [2,3] to represent relationships between hypotheses and the evidence to be explained. Each node represents some proposition, and the connections explicitly detail the relationships between different propositions. Furthermore, each node in a WAODAG corresponds to a logical AND or OR operation on its immediate parents.

An assignment of a truth value to each node is considered a proof if it is consistent with respect to the boolean network and if the items we wish to explain have been explained, i.e., have been assigned a value of true. Consequently, each such proof will have an associated cost. The goal is to find an assignment which has minimal cost (see Fig. 1). However, it has been shown that this problem is NP-hard [3]. NP-hardness is, of course, a worst-case complexity measure. One might nevertheless hope that a technique could be found which works well on the problems which come up in practice. Unfortunately, the current approaches show exponential growth in practice as well as in theory. These current approaches to finding the best proof have centered around using a best-first search technique and *expanding* partial proofs to search for the best proof [2].

In this paper, we present an approach that uses linear constraints to represent causal relationships. Each node in a WAODAG is treated as a

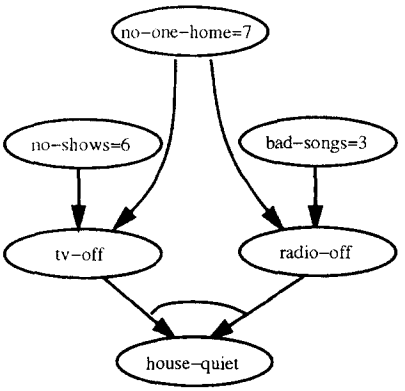

Fig. 2. A simpler WAODAG. The AND-node house-quiet is the observation. The nodes no-shows, blackout, and bad-songs are the hypotheses with associated costs 6, 7, and 3, respectively.

variable and constraints between nodes are represented by linear inequalities. Linear programming techniques are then used to minimize a cost function associated with the WAODAG. This results in a minimal cost proof for the original problem. Occasionally, since we model true and false values with "1" and "0", straight linear programming techniques may not arrive at the proper solution. We thus need to augment our approach. Although the initial solution may not be 0–1, we can use the bounding information it provides in an incremental branch-and-bound search which will guarantee a minimal cost proof. The experimental results (see Section 4) show however that the branch-and-bound is rarely required. Indeed, as opposed to the current search techniques, our linear programming technique shows expected-case polynomial growth rate on typical problems.

In Section 2, we present our formulation of cost-based abduction in terms of linear constraints and show that by working with these constraints, we can effectively determine the minimal cost proof. In Section 3, we describe the branch-and-bound algorithm which occasionally augments our linear constraint system. In Section 4, we describe and analyze the experimental results of our linear constraint satisfaction approach in comparison to existing search heuristics. Finally, in Section 6, we present some concluding thoughts and discuss some future research problems.

2. Cost-based abduction and linear constraints

We now formalize the cost-based abduction problem:

Notation. $\mathbb{R}$ denotes the set of real numbers.

Definition 2.1. A *WAODAG*[1] is a 4-tuple (G, c, r, S), where:

(1) G is a directed acyclic graph, $G = (V, E)$;
(2) c is a function from $V \times \{\text{true}, \text{false}\}$ to $\mathbb{R}$, called the *cost function*;
(3) r is a function from V to $\{\text{AND}, \text{OR}\}$, called the *label*; a node labeled AND is called an AND-*node*, etc.;
(4) S is a subset of nodes in V called the *evidence nodes.*[2]

Notation. V_H is the set of all nodes in V with indegree 0. The nodes in V_H are also called the *hypothesis nodes.*

Clearly, an explanation is the same as a proof for cost-based abduction. We formally define a proof for a set of observations as an assignment of truth values to the different propositions in the knowledge base such that the observations are true and the assignments are consistent with respect to the boolean relationships between the propositions.

In Fig. 2, assume that house-quiet = true is the observation to be explained. One possible proof would be the following assignment of truth values: {house-quiet, radio-off, bad-songs, tv-off, no-shows} are assigned true and {no-one-home} is assigned false. A second possibility is to assign all of them to true. And as a third possibility, we can assign {house-quiet, radio-off, tv-off, no-one-home} to true and {bad-songs, no-shows} to false. As we can easily see, all three assignments are internally consistent with the boolean relationships and assign house-quiet to true.

More formally, we define this as follows:

Definition 2.2. A *truth assignment* for a WAODAG $W = (G, c, r, S)$ where $G = (V, E)$ is a function e from V to $\{\text{true}, \text{false}\}$. We say that such a function is *valid* iff (if and only if) the following conditions hold:

(1) For all AND-nodes q, $e(q) = \text{true}$ iff for all nodes p such that (p, q) is an edge in E, $e(p) = \text{true}$.
(2) For all OR-nodes q, $e(q) = \text{true}$ iff there exists a node p such that (p, q) is an edge in E and $e(p) = \text{true}$.

Furthermore, we say that e is an *explanation* iff e is valid and for each node q in S, $e(q) = \text{true}$.

In propositional logic, $A \wedge B \Longrightarrow C$ corresponds to "If A and B are both true, then C is true". As we mentioned earlier, simply assuming the antecedent C is typically unsatisfactory as an explanation for C. Note that our definition of valid explanations disallows the case where C is true but

[1] Slight generalization of Charniak and Shimony [3]

[2] S represents the set of observations to be explained. Without loss of generality, we only consider positive evidence.

both A and B are false. However, we can straightforwardly model the ability to simply assume C if necessary.

Once we have the different possible explanations for some observation, we must associate a cost with each one to impose some order reflecting the *goodness* of proofs. As we mentioned earlier, the cost of a proof is simply the sum of the individual costs of the hypotheses assumed. [3]

Definition 2.3. We define the *cost of an explanation* e for $W = (G, c, r, S)$ where $G = (V, E)$ as

$$C(e) = \sum_{q \in V} c(q, e(q)). \tag{1}$$

An explanation e which minimizes C is called a *best explanation* for W.

From (1), we find that our three proofs above have costs 9, 16, and 7, respectively. Of the three our best proof is the third one with the cost of 7.

Now we show how to formulate our cost-based abduction as a linear constraint system.

Definition 2.4. A *linear constraint system* is a 3-tuple (Γ, I, ψ) where Γ is a finite set of variables, I is a finite set of linear inequalities based on Γ, and ψ is a function from $\Gamma \times \{\text{true}, \text{false}\}$ to $\mathbb{R}$.

Notation. For each node q in V, let $D_q = \{p \mid (p, q)$ is an edge in E$\}$ be the parents of q. $|D_q|$ is the cardinality of D_q.

From Definition 2.2, for a truth assignment to be a possible explanation, we must guarantee the internal consistency of the assignments required in the definition. This internal consistency is the same consistency required in boolean combinational circuits. We must guarantee the correct assignment of input values versus output values of each AND/OR-node in the WAODAG.

Like values in boolean circuits, we can use numerical assignments instead of true or false. In general, we use 1 for true and 0 for false. By taking this viewpoint, we can now consider the internal consistency as some form of mathematical formulae to be satisfied where each node is actually a variable in the equation. Our purpose is now to show how these equations can be derived and then prove that they guarantee the internal consistency required.

We begin our derivation with the simplest of the requirements. Let q be an evidence node in our WAODAG. Associate the variable x_q with q. Since q

[3] Although we have only discussed having costs associated with hypotheses, our approach permits costs to be associated with any node.

is an evidence node, any explanation for q must assign q to true. This can be modeled by the equation

$$x_q = 1.$$

Next, let q be an AND-node with parents D_q. We have the following: *q is true iff p is true for all nodes p in D_q*. Symmetrically, *q is false iff there exists a p in D_q such that p is false*. We can accomplish this with the equations (2) and (3):

$$x_q \leqslant x_p \quad \text{for each } p \in D_q, \tag{2}$$

which guarantees that

(1) q being true forces all p in D_q to be true, and
(2) some p in D_q being false forces q to be false;

$$\sum_{p \in D_q} x_p - |D_q| + 1 \leqslant x_q \tag{3}$$

guaranteeing that

(1) q being false forces some p in D_q to be false, and
(2) if all p in D_q are true, then q must be true.

Note that at this time we are assuming that our variables may only take values of 0 or 1 although there is no upper or lower bound on the results of evaluating either side of the equation. For example, let $D_q = \{a, b, c, d\}$, $x_a = x_b = x_c = 0$ and $x_d = 1$. This implies that the summation side of equation 3 above yields -2!

Finally, the OR-node can be modeled with the following equations:

$$\sum_{p \in D_q} x_p \geqslant x_q,$$

$$x_q \geqslant x_p \quad \text{for each } p \in D_q,$$

where q is an OR-node with parents D_q.

Together, these equations will guarantee the internal consistency needed for a truth assignment to be an explanation. Also, any explanation is guaranteed to satisfy this set. We formalize our construction as follows:

Definition 2.5. Given a WAODAG $W = (G, c, r, S)$ where $G = (V, E)$, we can construct a linear constraint system $L(W) = (\Gamma, I, \psi)$, where:

(1) Γ is a set of variables indexed by V, i.e., $\Gamma = \{x_q \mid q \in V\}$;
(2) $\psi(x_q, X) = c(q, X)$ for all $q \in V$ and $X \in \{\text{true}, \text{false}\}$;

(3) I is the collection of all inequalities of the forms given below:

$$x_q \leqslant x_p \in I \quad \text{for each } p \in D_q, \quad \text{if } r(q) = \text{AND}, \tag{4}$$

$$\sum_{p \in D_q} x_p - |D_q| + 1 \leqslant x_q \in I, \quad \text{if } r(q) = \text{AND}, \tag{5}$$

$$\sum_{p \in D_q} x_p \geqslant x_q \in I, \quad \text{if } r(q) = \text{OR}, \tag{6}$$

$$x_q \geqslant x_p \in I \quad \text{for each } p \in D_q, \quad \text{if } r(q) = \text{OR}. \tag{7}$$

We say that $L(W)$ is *induced* by W. Furthermore, by including the additional constraints:

$$x_q = 1, \quad \text{if } q \in S \tag{8}$$

we say that the resulting linear constraint system is *induced evidentially* by W and is denoted by $L_{\mathrm{E}}(W)$.

Definition 2.6. A *variable assignment* for a linear constraint system $L = (\Gamma, I, \psi)$ is a function s from Γ to $\mathbb{R}$. Furthermore,

(1) if the range of s is $\{0, 1\}$, then s is a 0–1 *assignment*;
(2) if s satisfies all the constraints in I, then s is a *solution* for L;
(3) if s is a solution for L and is a 0–1 assignment, then s is a 0–1 *solution* for L.

With our formulation of linear constraint systems and variable assignments, we can now prove that 0–1 solutions are equivalent to explanations. Given a 0–1 assignment s for $L(W)$, we can construct a truth assignment e for W as follows:

(1) For all q in V, $s(x_q) = 1$ iff $e(q) = \mathsf{true}$.
(2) For all q in V, $s(x_q) = 0$ iff $e(q) = \mathsf{false}$.

Conversely, given a truth assignment e for W, we can construct a 0–1 assignment s for $L(W)$.

Notation. $e[s]$ and $s[e]$ denote, respectively, a truth assignment e constructed from a 0–1 assignment s, and a 0–1 assignment s constructed from a truth assignment e.

We can show that all explanations for a given WAODAG W have corresponding 0–1 solutions for $L_{\mathrm{E}}(W)$ and vice versa.

Theorem 2.7. *If e is an explanation for W, then $s[e]$ is a* 0–1 *solution for $L(W)$.*

(Proofs can be found in Appendix A.)

Corollary 2.8. *Let L be constructed from $L(W)$ by eliminating all constraints of the forms (5) and (7). If e is an explanation for W, then $s[e]$ is a solution of L.*[4]

Conditions (4) and (6) are called *bottom-up constraints* since they dictate the values of variables from the direction of the evidence nodes. Symmetrically, (5) and (7) are called *top-down constraints.* As we shall see later in this section, Corollary 2.8 will demonstrate certain enhancements and improvements which can be made to our approach.

Theorem 2.9. *If s is a 0–1 solution for $L_E(W)$, then $e[s]$ is an explanation for W.*

From Theorems 2.7 and 2.9, 0–1 solutions for linear constraint systems are the counterparts of explanations for WAODAGs. By augmenting a WAODAG induced linear constraint system with a cost function, the notion of the cost of an explanation for a WAODAG can be transformed into the notion of the cost of a 0–1 solution for the linear constraint system.

To complete the derivation, we must also be able to compute the costs associated with each proof. We can do this as follows:

Definition 2.10. Given a linear constraint system $L(W) = (\Gamma, I, \psi)$ induced (evidentially) by a WAODAG W, we construct a function Θ_L from variable assignments to $\mathbb{R}$ as follows:

$$\Theta_L(s) = \sum_{x_q \in \Gamma} \{s(x_q)\psi(x_q, \mathsf{true}) + (1 - s(x_q))\psi(x_q, \mathsf{false})\}. \qquad (9)$$

Θ_L is called the *objective function* of $L(W)$.

Definition 2.11. An *optimal* 0–1 *solution* for a linear constraint system $L(W) = (\Gamma, I, \psi)$ induced (evidentially) by a WAODAG W is a 0–1 solution which minimizes Θ_L.

As we can clearly see, (9) is identical to (1). From Theorems 2.7 and 2.9 and the relationship between node assignments and variable assignments, an optimal 0–1 solution in $L_E(W)$ is a best explanation for W and vice versa.

Taking the set of linear inequalities I and the objective function Θ_L, we observe that we have the elements known in operations research as a

[4] L is later defined as a WAODAG-*semi-induced linear constraint system.*

linear program [11,12,18]. The goal of a linear program is to minimize an objective function according to some set of linear constraints. Highly efficient methods such as the *simplex method*[5] and *Karmarkar's projective scaling algorithm* are used to solve linear programs [11,12,18]. Empirical studies have shown that the average running time of the simplex method is roughly linear with respect to the number of constraints and the number of variables in the linear program [12].

Proposition 2.12. *Given a* WAODAG $W = (G, c, r, S)$, *where* $G = (V, E)$, *if* $L_E(W) = (\Gamma, I, \psi)$ *is induced evidentially from* W, *then* $|I| = |E| + |V - V_H| + |S|$.

Although our linear constraint systems seem similar in nature to linear programs, linear programs cannot make restrictions which cannot be modeled by linear inequalities. Thus, solutions which minimize the objective function may not be strictly 0 and 1. However, if the solution is a 0–1 solution, then the best explanation is found. (From our experiments, as we shall see later, the optimal solutions for many of these linear programs will in fact be 0–1 solutions. Thus, the best explanation can be found by just using the straight simplex methods on the problems.) If the solution is not a 0–1 solution, the value for the objective function generated by such a solution still provides an excellent lower bound to the cost of an optimal 0–1 solution. This lower bound will be used to direct our search for an optimal 0–1 solution as we shall see below.

For computing the lower bound, WAODAG induced linear programs are well suited for the simplex method. The constraint matrices for these types of linear programs are extremely sparse and consist of only three values: -1, 0, 1. Furthermore, detailed knowledge of the problem structure can be exploited to even further improve performance. The following theorem shows that the number of linear inequalities can be reduced under certain conditions.

Definition 2.13. Given a WAODAG $W = (G, c, r, S)$, where $G = (V, E)$, we can construct a linear constraint system $\hat{L}(W) = (\Gamma, I, \psi)$ where:

(1) Γ is a set of variables indexed by V, i.e., $\Gamma = \{x_q \mid q \in V\}$;

(2) $\psi(x_q, X) = c(q, X)$ for all $q \in V$ and $X \in \{\text{true}, \text{false}\}$;

(3) I is the collection of all inequalities of the forms given below:

$$x_q \leqslant x_p \in I \text{ for each } p \in D_q, \quad \text{if } r(q) = \text{AND}, \tag{4}$$

$$\sum_{p \in D_q} x_p \geqslant x_q \in I, \quad \text{if } r(q) = \text{OR}. \tag{6}$$

[5]For a quick overview of the simplex method, see [8]

We say that $L(W)$ is *semi-induced* by W. Furthermore, by including the additional constraints:

$$x_q = 1, \quad \text{if } q \in S \tag{8}$$

we say that the resulting linear constraint system is *semi-induced evidentially* by W and is denoted by $\hat{L}_E(W)$. (Properties associated with induced linear constraint systems are easily generalizable to semi-induced ones.)

A semi-induced linear constraint system is simply an induced linear constraint system lacking top-down constraints. From Corollary 2.8, the set of possible solutions for $\hat{L}_E(W)$ is a superset of the set of possible explanations for W.

Theorem 2.14. *Let $W = (G, c, r, S)$ be a* WAODAG, *where $G = (V, E)$. An optimal* 0–1 *solution for $\hat{L}_E(W)$ can be transformed into a best explanation for W in $O(|E|)$ steps if $c(p, \text{false}) \leqslant c(p, \text{true})$ for all nodes p in V.*

For transformation, see the proof of Theorem 2.14 in Appendix A. In general, transforming a 0–1 optimal solution for $\hat{L}_E(W)$ requires at most $2|E|$ steps.

Corollary 2.15. *Let $W = (G, c, r, S)$ be a* WAODAG, *where $G = (V, E)$. An optimal 0–1 solution for $\hat{L}_E(W)$ is a best explanation for W if $c(p, \text{false}) < c(p, \text{true})$ for all nodes p in V.*

From the above theorem, a best explanation for W can be found by solving a smaller linear program.

Intuitively, we note that the information required to find an optimal 0–1 solution is propagated from true assignments which originate from the evidence nodes and thus, results in a bottom-up fashion of processing. In terms of our linear constraint system, constraints need only be sensitive to the information from one direction, namely from the evidence nodes.

Proposition 2.16. *Given a* WAODAG *$W = (G, c, r, S)$, where $G = (V, E)$, if $\hat{L}_E(W) = (\Gamma, I, \psi)$ is semi-induced evidentially by W, then*

$$|I| = |\{(p, q) \in E \mid r(p) = \text{AND}\}| + |V_O| + |S|,$$

where V_O is the subset of all nodes in V which are labeled OR *and have nonzero outdegree.*

Many other types of improvements may also be employed. Some arise from the intimate knowledge of our domain while others are techniques used for general linear programming problems.

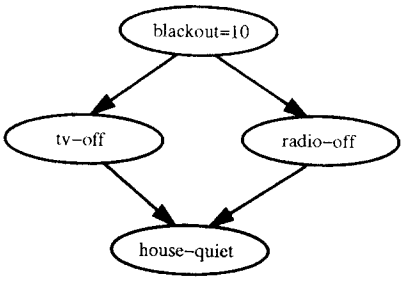

Fig. 3. In this simple WAODAG, the OR-node house-quiet is the observed evidence. blackout is the only hypothesis available.

Although only WAODAGs are used in the preceding discussions, one can easily generalize our linear constraint satisfaction approach to arbitrary boolean gate-only networks.

3. Branch-and-bound

As we mentioned in the previous section, the solution to a WAODAG induced linear program need not consist strictly of 0s and 1s. For example, consider the simple WAODAG in Fig. 3. (Note that we will use the terms linear constraint systems and linear programs interchangeably throughout this section.) From this WAODAG, the following linear program is generated from our semi-induced linear constraint system $L = (\Gamma, I, \psi)$:

$$\begin{aligned} H &= 1, \\ H &\leqslant R + T, \\ B &\geqslant R, \\ B &\geqslant T, \\ 0 &\leqslant H, R, T, B \leqslant 1, \end{aligned}$$

and has objective function:

$$\Theta_L(s) = s(B)\psi(B, \text{true}) + (1 - s(B))\psi(B, \text{false}),$$

where $H, R, T, B \in \Gamma$ respectively stand for house-quiet, radio-off, tv-off, and blackout. Furthermore, assume $\psi(B, \text{false}) = 0$ and $\psi(B, \text{true}) > 0$.

We can easily show that the solution which minimizes the objective function is as follows: $H = 1$, $R = 0.5$, $T = 0.5$, and $B = 0.5$ with $\Theta_L(s) = \psi(B, \text{true})/2$. We call B a *shared node* in our WAODAG. An OR-node such as H with assigned value strictly greater than its parents is called a *divide node*. (It is easy to show that either an OR-node is a divide node, or that all of its parents are 0 or the same value as the OR-node.)

Looking closely at Fig. 3, we could easily remedy this problem by introducing the constraint house-quiet $\leqslant$ blackout. This new constraint reflects

the fact that the value for house-quiet is ultimately determined by blackout. Simple patches like this one could be used to prevent this type of split node problem. However, most are nontrivial to identify and repair.

With small linear programs like the one above, using a brute-force technique of simply trying each possible assignment maybe feasible. Of course, the run-time grows exponentially with respect to the size of the problem.[6]

The technique to be presented avoids the necessity of searching the entire solution space by using the lower bound computed by the linear program. This is a standard technique used in many domains to speed up processing time.

The basic idea is as follows: To find an optimal 0–1 solution, we solve a sequence of linear programs. This sequence can be represented by a tree where each node in the tree is identified with a linear program that is derived from the linear programs on the path leading to the root of the tree. The root of the tree is identified with the linear program induced by our WAODAG. The linear programs along the nodes of the tree are generated using the following schema: Consider s_0, the optimal solution to our initial linear program denoted lp_0. If s_0 is a 0–1 solution, then we are finished. Otherwise, we choose some non-integral variable x_q in s_0 and define two new problems lp_1 and lp_2 as descendants of lp_0. lp_1 is identical to lp_0 except for the additional constraint $x_q = 1$, and lp_2 is identical to lp_0 except for the additional constraint $x_q = 0$. Note that the two new problems do not have s_0 as their optimal solutions. Since we are looking for a 0–1 assignment, the optimal 0–1 solution must satisfy one of the additional constraints. The two new nodes just defined are called *active nodes* and the variable x_q is called the *branching variable.*

Next, we choose one of the problems identified with an active node and attempt to solve it. It is not necessary to run a complete simplex method on the linear program. Using methods such as the *dual simplex algorithm* [12,18], information is used in an incremental manner from other runs resulting in a quick and efficient computation. If the optimal solution is not a 0–1 solution, then two new problems are defined based on the current linear program. These new problems contain all the constraints of the parent problem plus the appropriate additional one.

When a 0–1 solution is found for some active node, the value of its objective function is compared against the current best. If the cost of the new solution is better than the current best, it is then used to prune those active nodes whose computed lower bounds exceed this value. This solution also now becomes the current best solution.

Branching continues in this manner until there are no active nodes in the

[6]See integer programming techniques [11,12,18]

tree. At the end, the current best solution is guaranteed to be the optimal 0–1 solution.

This technique is generally classified as a *branch-and-bound* technique in the area of *integer linear programming* [12,18]. Also, it can be applied to any linear constraint system regardless of whether or not they are WAODAG induced.

Notation. We denote a linear program by an ordered pair (I, Θ), where I is a set of linear constraints and Θ is an objective function.

Algorithm 3.1. Given a linear constraint system $L = (\Gamma, I, \psi)$, find its optimal 0–1 solution.

(1) *Initialization.* Set CurrentBest := $\emptyset$ and *ActiveNodes* := $\{(I, \Theta_L)\}$.
(2) If *ActiveNodes* = $\emptyset$ then go to step (14). Otherwise, let lp be some linear program in *ActiveNodes*.
(3) *ActiveNodes* := *ActiveNodes* − {lp}.
(4) Compute the optimal solution s^{opt} for lp using simplex, etc.
(5) If s^{opt} is a 0–1 solution, then go to step (12).
(6) *Bound.* If *CurrentBest* $\neq \emptyset$ and $\Theta_L(s^{\text{opt}}) > \Theta_L(\textit{CurrentBest})$, then go to step (2).
(7) *Branch.* Choose some variable x_q in lp whose value in s^{opt} is non-integer.
(8) Set $I_1 := I \cup \{x_q = 0\}$ and $I_2 := I \cup \{x_q = 1\}$.
(9) Create two new linear programs $\text{lp}_1 := (I_1, \Theta_L)$ and $\text{lp}_2 := (I_2, \Theta_K)$.
(10) *ActiveNodes* := *ActiveNodes* $\cup \{\text{lp}_1, \text{lp}_2\}$.
(11) Go to step (2).
(12) 0–1 *solution.* If *CurrentBest* = $\emptyset$ or $\Theta_L(s^{\text{opt}}) < \Theta_L(\textit{CurrentBest})$, then set *CurrentBest* := s^{opt} and prune *ActiveNodes* using *CurrentBest*.
(13) Go to step (2).
(14) *Solution.* Print *CurrentBest*.

To solve Fig. 3, we choose tv-off to be our first branching variable. Following Algorithm 3.1 above, we generate two new linear programs:

$$
\begin{aligned}
L_1: \quad & H = 1, \\
& H \leqslant R + T, \\
& B \geqslant R, \\
& B \geqslant T, \\
& 0 \leqslant H, R, T, B \leqslant 1, \\
& T = 0,
\end{aligned}
\qquad\qquad
\begin{aligned}
L_2: \quad & H = 1, \\
& H \leqslant R + T, \\
& B \geqslant R, \\
& B \geqslant T, \\
& 0 \leqslant H, R, T, B \leqslant 1, \\
& T = 1,
\end{aligned}
$$

both with objective function:

$$\Theta_L(s) = s(B)\psi(B, \text{true}) + (1 - s(B))\psi(B, \text{false}).$$

We first note that tv-off is now a fixed value in both L_1 and L_2. Since the original optimal value of tv-off was 0.5, both L_1 and L_2 now exclude this possibility which effectively eliminates the original optimal solution from their respective feasible solutions space. From simple observation, we find that the optimal solutions for L_1 and L_2 are $\{H = R = B = 1, T = 0\}$ for L_1 and $\{H = T = B = 1, R = 0\}$ for L_2. The cost for both assignments is $\psi(B, \text{true})$. We can easily show that both assignments are optimal 0–1 solutions.

In this algorithm, two points were left deliberately vague: the choice of the next active node and the choice of branching variable. Several different options exist for both.

For the choice of the next active node, we have the following:

N1: depth-first search of the branch-and-bound tree;
N2: breadth-first search of the branch-and-bound tree;
N3: choose the active node whose parent node has the best lower bound;
N4: choose the active node whose parent's solution s^{opt} is *closest* to a 0–1 solution according to

$$\sum_{q \in \text{lp}_k} \min\{s^{\text{opt}}(x_q), (1 - s^{\text{opt}}(x_q)\}.$$

For the choice of the next branching variable:

V1. choose only variables corresponding to hypotheses nodes since a 0–1 assignment to these variables guarantees a 0–1 assignment throughout the remaining variables;
V2. order the choice of variables as shared nodes, then divide nodes, then hypothesis nodes, and so on;
V3. choose the variable x_q which minimizes

$$\{\min\{s^{\text{opt}}(x_q), (1 - s^{\text{opt}}(x_q))\}\};$$

V4. choose the variable x_q which maximizes

$$\{\min\{s^{\text{opt}}(x_q), (1 - s^{\text{opt}}(x_q))\}\};$$

V5. choose the variable which has maximum associated cost;
V6. use any combination of the above.

Obviously, many other techniques exist for making our choices, some based on the knowledge of our problem and others based on general techniques. Combinations of active node heuristics N1 and N3 together with branching variable heuristics V2 and V3 seem most promising.[7]

[7]Techniques V1 and V2 seem to work best when values can propagate in a top-down fashion. Thus, using the induced linear constraint system as opposed to the semi-induced linear constraint system is desired.

Finally, since the success of branch-and-bound techniques depends on the ability to prune the active nodes early and as many as possible, we observe that pruning occurs whenever we have a 0–1 solution for the linear program. Although not every linear program results in a 0–1 solution, it is possible to build a 0–1 solution from the non-integer optimal solution. In fact, the construction is fairly straightforward and computationally cheap.

Theorem 3.2. *Let s be the optimal solution of some* WAODAG-*(semi-)induced linear program. Construct a variable assignment s' from s by changing all nonzero values in s to* 1. *s' is a* 0–1 *solution for the (semi-)induced linear constraint system.*

(For intuitions on why this theorem holds, see the proof in Appendix A.)

For both WAODAG-induced as well as semi-induced linear constraint systems L, we consider the following construction from s: s' is constructed from s by changing all nonzero hypothesis node values in s to 1. Now, enforcing the boolean gate-like properties of the WAODAG, we propagate the boolean values from the hypothesis nodes up this boolean graph.

Theorem 3.3. *s' constructed from s above is a* 0–1 *solution for L.*

Proof. Follows from Theorems 2.7 and 2.9. □

4. Experimental results

We performed two experiments to measure the efficiency of our linear constraint satisfaction approach. The first involves a real application of our technique to solve cost-based abduction problems created by the story understanding system WIMP [6,7]. This allows us to make a comparison against the search-style heuristic described in [2] to solve these graphs. Our second experiment involves testing our approach on randomly generated WAODAGs as a gauge on how well the technique applies to the general class of WAODAGs. Also, WAODAGs larger than those found in the first experiment are used.

For both experiments, we employed active node method N1 and branching variable technique V3 above.

Experiment 1: *WIMP WAODAGs*

WIMP is a natural language story comprehension system for parsing and understanding written English sentences [6,7]. It uses belief networks to perform the abductive inference tasks necessary to solve problems like

pronoun reference and word-sense disambiguation. The belief networks can then be transformed into equivalent cost-based abductions problems as shown in [3].

The algorithm for determining the minimal cost proof in [2] is based on a best-first search of the WAODAG. The basic idea is that one starts with the partial proof consisting only of the evidence nodes in the WAODAG and then creating alternative partial proofs.[8] In each iteration, a partial proof is chosen to be expanded. It is expanded by adding some new nodes and edges to the existing partial proof which takes into consideration how one of its goals can be achieved locally according to the current partial proof and nearby nodes. This continues until all the goals (such as evidence) are satisfied and results in a minimal cost proof. How this is actually done is outlined in [3].

Naturally, the success of this algorithm depends on having a good heuristic function for deciding which partial proof should be worked on next. Furthermore, the heuristic function must be admissible to guarantee that the first proof generated is the minimal cost proof. Efficient heuristics have been difficult to find. Prior to [2] the only basic admissible heuristic that had been used was *cost-so-far* in [3,21]. Simply put, the partial proofs are weighted according to the costs of the hypotheses which they currently contain. The difficulty in the cost-so-far approach is that no estimation on the "goodness" of the partial proof can be made until hypothesis nodes have been reached. Recently, a more efficient heuristic was introduced which used a more sophisticated cost estimator [2]. In brief, the new heuristic propagates the costs of the hypothesis nodes down the network to give some estimation of the "goodness" of each partial proof. The admissibility of this heuristic is guaranteed by the special care it takes in expanding partial proofs. (For precise details and the admissibility proof of this heuristic, see [2]).

Since the problem of finding the minimal cost explanations is NP-hard, we are naturally interested in the expected-case growth rates of the heuristic search method versus our linear constraint systems. Unfortunately, performing average-case analyses is a rather difficult task. This has certainly been the case when studying either heuristic search methods or linear programming. Thus, we are only left with making empirical studies on the two approaches as the basis for our comparisons. However, this technique is wrought with pitfalls unless we are extremely careful. If we employ a floating-point optimization in our linear programming implementation which enhances the efficiency of our approach, does this imply that our linear constraint approach has improved relative to the search heuristic? Typically, if the two

[8] A *partial proof* is a subgraph of the WAODAG.

methods were somewhat more isomorphic, a similar optimization can also be applied to the other method. However, as we can easily see, the floating-point optimization is not very useful to the heuristic search method.

Our goal is to compare the two methods above without being influenced by "approach-independent" factors, such as compiler optimizations, machine type, etc., which can directly alter the empirical results. Often, this can be done by comparing approaches at a more abstract level. For example, two search routines could be compared by measuring how many search steps each took, independent of the time for each step. Unfortunately, the radical differences between the two approaches we are comparing defeats us. There is no obvious common ground for comparison except the most obvious one—how long the systems took. However, suppose that given a collection of ordered pairs of the form

(WAODAG complexity, CPU usage),

we attempt to perform a least-squares fit of the data on the function $t = e^{a+bx}$, where t is the CPU seconds used and x is the complexity of the WAODAG to be solved. We can now compare the relative efficiency of each approach by comparing the constant b for both fits. In this way, we hope to eliminate the approach-independent factors.

There are several complexity measures available for WAODAGs. We chose as our measure the number of edges in the WAODAG as it seemed to us to have the most direct impact on both the graph search heuristic and our linear constraints approach. An expansion of a partial proof for the search heuristic necessitates the traversal of the graph along its edges. For our linear constraint systems, Proposition 2.12 stipulates that the number of constraints required to solve a WAODAG is roughly the number of edges in the WAODAG.[9]

In this experiment, 140 WAODAGs generated by WIMP ranging in size from 7 nodes to 158 nodes and from 12 edges to 375 edges were presented to both approaches. Table 1 summarizes the set of WAODAGs generated by WIMP for our experiment. Figures 4 and 5 show the semi-logarithmic plot of our timings for the WIMP heuristic and for our linear constraint satisfaction approach.

We found that in our linear constraint satisfaction approach, roughly 61% of the WAODAGs generated by WIMP were solved using only linear programming without resorting to branch-and-bound. Furthermore, the number of active nodes actually used during branch-and-bound cases were only a small fraction of the total number of nodes involved. On average for the branch-and-bound cases alone, the number of active nodes was 6.2% of the nodes

[9]Generally, the number of edges is some multiple of the number of nodes.

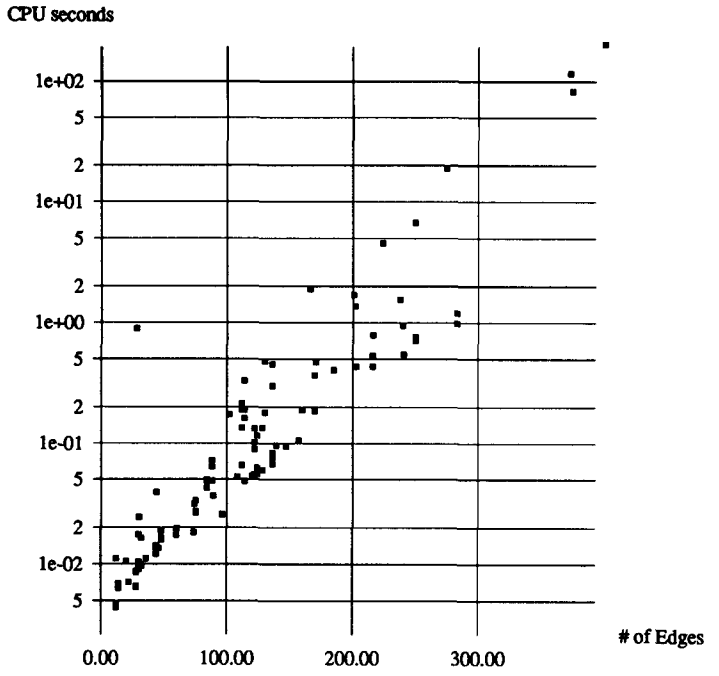

Fig. 4. Semi-logarithmic plot of WIMP heuristic timings.

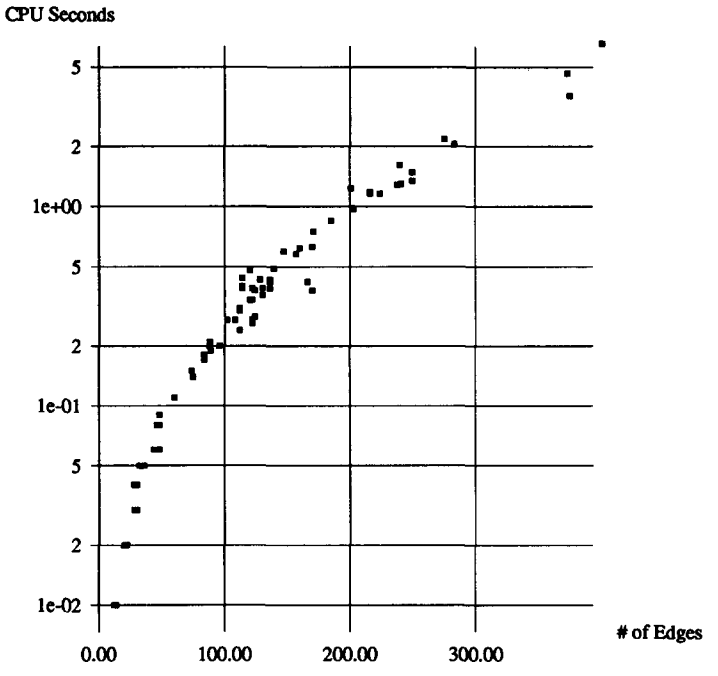

Fig. 5. Semi-logarithmic plot of linear constraint satisfaction timings.

Table 1
WIMP WAODAG summary.

	Nodes	Edges	Hypotheses	OR-nodes
Min	7	12	2	3
Max	158	375	41	59
Average	42.54	93.49	10.88	16.76
Median	34	75	9	14
Total	5955	13089	1523	2346

in the graph, and overall, the average number for all WAODAGs was 2.4%.

Performing a least-squares fit on the timings gives us the following:

$$\text{WIMP heuristic:} \quad e^{-5.32+0.0245x},$$
$$\text{Constraint system:} \quad e^{-3.80+0.0187x},$$

which serves to verify some of our expectations on the expected growth rate of our linear constraint satisfaction approach as compared to the search technique.

Although we have just shown that our approach is better than the search heuristic found in WIMP, the best exponential fit does not actually describe our timings very well. Consider again the semi-logarithmic plot of our linear constraint satisfaction approach in Fig. 5.

As we can clearly see, our linear constraint satisfaction approach actually exhibits an expected subexponential growth rate. By further attempting to fit our data to ax^b, we get $0.0001371x^{1.6484}$ as our expected growth curve. To show that the polynomial fit is better, we compare the two least square error fits, that is,

$$\sum_{e \in \Lambda} |\chi_e - F(e)|^2$$

where Λ is the set of all WAODAGs, χ_e is the amount of CPU seconds taken to solve WAODAG e and $F(e)$ is the amount of time predicted by the least-squares fit. Taking the error of the exponential fit and dividing it by the error of the polynomial fit, we roughly find a 10000% improvement of the polynomial fit over the exponential. When we attempted to perform a polynomial fit on the search heuristic, we found that the error actually tripled. Although the search heuristic is slightly faster than our approach on the very small (in terms of edges) WAODAGs, our approach seems to be quite fast and practical at solving all the WAODAGs generated by WIMP.

Experiment 2: random WAODAGs

Our purpose in this experiment is to further test the efficiency of our linear constraint satisfaction approach when faced with more general and larger

Table 2
Random WAODAG summary.

	Nodes	Edges	Hypotheses	OR-nodes
Min	36	39	8	7
Max	387	699	154	135
Average	224.2	328.4	74.2	75.6
Median	225	323	76	79
Total	22424	32844	7423	7559

WAODAGs than those generated by WIMP. In particular, we are interested in whether the expected subexponential growth rate exhibited for the WIMP generated WAODAGs remains to be the case for these randomly generated graphs. We also considered testing the search heuristic on these graphs. However, the explosive growth rate of the heuristic made it infeasible to attempt these much larger graphs.

100 WAODAGs were generated ranging from 36 to 387 nodes and from 39 to 699 edges. They were generated randomly[10] by first determining the number of nodes n from 1 to 400 and then instantiating said nodes. Next, the number of edges from n to 800 to be included in this graph is determined and the edges were randomly instantiated between two nodes.[11] Finally, hypothesis nodes are identified and are arbitrarily assigned some non-negative cost. Table 2 summarizes the set of randomly generated WAODAGs for this experiment.

We found that 97% of the randomly generated WAODAGs were solved using only linear programming without branch-and-bound.[12] Performing a least-squares exponential fit gives us $e^{-5.49+0.0614x}$.

Consider the logarithmic plot of our linear constraint satisfaction approach in Fig. 6. Again, we can clearly see that our linear constraint satisfaction approach actually exhibits an expected subexponential growth rate. By further attempting to fit our data to ax^b, we get $0.0079188x^{2.0308}$ as our growth curve. Again, the error fit actually improved roughly 2300%.

5. Conclusions

From the two experiments above, our linear constraint satisfaction approach seems very promising. A very surprising result is that the optimal solution found for the linear program without branch-and-bound was either

[10]By random, we mean uniform distribution.

[11]To guarantee that our resulting graph is acyclic, we initially imposed a random topological ordering on the nodes.

[12]We are currently investigating why this differs so much from WIMP-generated WAODAGs.

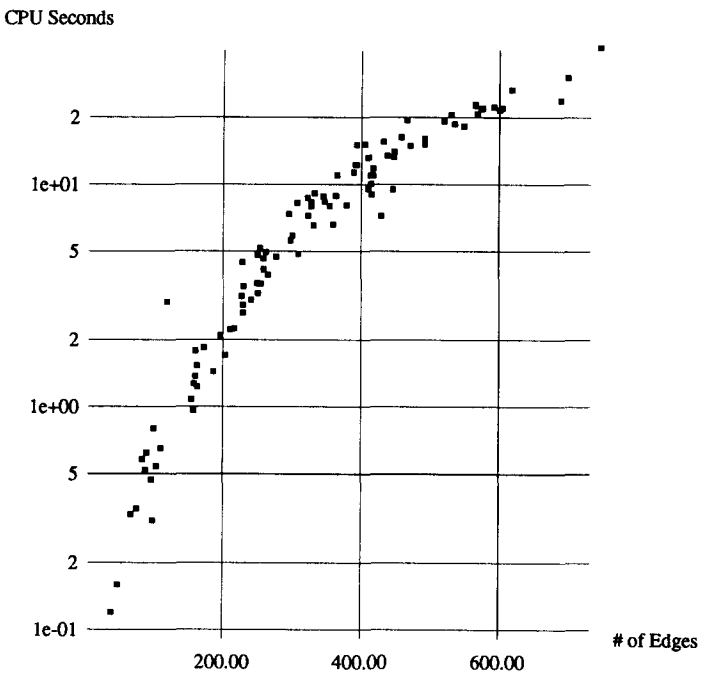

Fig. 6. Logarithmic plot of linear constraint satisfaction timings on random WAODAGs.

already a 0–1 optimal solution or a very close approximation as indicated by the relatively small number of active nodes used. Furthermore, branch-and-bound can be performed incrementally by using methods such as the dual simplex algorithm. Thus, the additional computational effort required beyond solving the initial linear program was rather minimal.

Instinctively, we would have guessed that the bulk of our problems would have centered around the branch-and-bound process since it seems unlikely that our linear program should have an optimal solution which is also integral. Why they are so often integral is a puzzling problem. We are currently studying this phenomenon, however, it seems to be a very difficult problem. We suspect that abduction may fall into a class of problems considered to be "easy" integer programming problems. There seems to be a link between abduction and set-covering problems and it has been frequently observed that matching and set covering problems on graphs are very amenable to linear programming formulations in that they very often have integral optimal solutions [4].

As for the association between abduction and set-covering, it is admittedly a weak one. We offer only two suggestive points. First, the early abduction model for medical diagnoses presented in [14] is such a set-covering approach. Cost-based abduction is a generalization of [14]. Second, in the original formulation of cost-based abduction presented in [3], the proof

of NP-completeness was accomplished by transforming the *vertex covering problem* into cost-based abduction.

6. Further research

In conclusion, the linear constraint satisfaction approach can be used to solve the minimum cost-based abduction problem in an efficient manner superior to existing search-style heuristics. The formalism of linear constraint satisfaction provided a natural framework for finding minimal cost proofs.

It seems likely that our approach can be extended to model other problems in explanation and reasoning. One of the extensions being currently explored involves generating the other alternative explanations. In abductive explanation, having alternative explanations is often useful and sometimes necessary. Having the second best, third best, and so on, can provide a useful gauge on the *quality* of the best explanation. (Details concerning the generation of alternative explanations can be found in [17].) Another extension being explored involves modeling *belief revision* in *Bayesian networks* [13] using our linear constraint systems [17]. Finally, other extensions currently being incorporated into our approach include handling partial explanations.

Appendix A. Proofs

Proof of Theorem 2.7. Assume that $s[e]$ is not a solution of $L(W) = (\Gamma, I, \psi)$. This implies that there exists a constraint Q in I which is violated. (For notational convenience, we will denote $s[e](x_p) = a$ by $x_p = a$.)

Case 1: *Q is of the form $x_p \leqslant x_q$.* Since $s[e]$ is a 0–1 assignment, $x_p = 1$ and $x_q = 0$. From (4) and (7), we get can conclude that either $r(p) =$ AND or $r(q) =$ OR. If $r(p) =$ AND, then q is a parent of p and x_q must equal 1 in $s[e]$. If $r(p) =$ OR, then p is a child of q and x_p must equal 0 in $s[e]$. Since neither is the case, Q cannot be violated.

Case 2: *Q is of the form*

$$\sum_{q \in D} x_q - |D| + 1 \leqslant x_p,$$

where D is some set of nodes. For Q to be violated,

$$\sum_{q \in D} x_q - x_p > |D| - 1.$$

This implies that $x_p = 0$ and for all $q \in D$, $x_q = 1$. From (5), we can conclude that $r(p) =$ AND and $D = D_p$. Since $r(p)$ is an AND node, if x_p

equals 0, then all the parents of p must also equal zero. Thus, Q cannot be violated.

Case 3: *Q is of the form*

$$\sum_{q \in D} x_q \geqslant x_p,$$

where D is some set of nodes. This implies that $x_p = 1$ and for all $q \in D$, $x_q = 0$. From (6), we conclude that $r(p) =$ OR and $D = D_p$. Since $r(p)$ is an OR node, x_p equals 1 implies that there exists an $q \in D_p$ such that $x_q = 1$. Thus, Q cannot be violated.

Cases 1 to 3 cover every type of violations of the linear constraint system possible. Therefore, e cannot be an explanation for W. □

Proof of Theorem 2.9. Assume $e[s]$ is not an explanation for W. This implies that one or more of the following conditions hold: (For notational convenience, we will denote $s(x_p) = a$ by $x_p = a$ and $e[s]$ by e.)

(a) There exists an AND-node p in W such that $e(p) =$ true and there exists a $q \in D_p$ such that $e(q) =$ false.
(b) There exists an AND-node p in W such that $e(p) =$ false and for all $q \in D_p$, $e(q) =$ true.
(c) There exists an OR-node p in W such that $e(p) =$ true and for all $q \in D_p$, $e(q) =$ false.
(d) There exists an OR-node p in W such that $e(p) =$ false and there exists a $q \in D_p$ such that $e(q) =$ true.
(e) There exists an evidence node p in S such that $e(p) =$ false.

Case 1. From (4), $r(p) =$ AND implies that the constraint $x_p \leqslant x_q$ is in I. Since, $x_p = 1$ and $x_q = 0$, condition (a) does not hold.

Case 2. From (5), $r(p) =$ AND implies that $x_p \geqslant 1$. Thus, condition (b) does not hold.

Case 3. From (6), $r(p) =$ OR implies that $x_p \leqslant 0$. Thus, condition (c) does not hold.

Case 4. From (7), $r(p) =$ OR implies that $x_p \geqslant 1$. Thus, condition (d) does not hold.

Case 5. From (8), $p \in S$ implies that $x_p = 1$. Thus, condition (e) does not hold.

Therefore, s is not a 0–1 solution in $L_E(W)$. □

Before we can prove the next theorem, we present the definition of AND-DAGS.

Definition A.1. An *AND-DAG* is a WAODAG whose nodes which are labeled OR have at most outdegree one. Given a WAODAG $W = (G, c, r, S)$, construct $W' = (G', c', r', S)$ from W by removing all but one of the parent from every OR-node. Now, remove from W', all nodes and associated edges which are not reachable from any evidence node in S. The resulting AND-DAG W' is said to be *induced* by W.

Proposition A.2. *Let W' be an AND-DAG induced by W. For any truth assignment e, if $e(p) =$ true for all nodes p in W', then e is an explanation for W.*

Proof of Theorem 2.14. Let s be any optimal 0–1 solution for $\hat{L}_E(W)$. Assume $e[s]$ is not an explanation for W. This implies that one or more of the following conditions hold: (For notational convenience, let e denote $e[s]$.)

(a) There exists an AND-node p in W such that $e(p) =$ true and there exists a $q \in D_p$ such that $e(q) =$ false.
(b) There exists an AND-node p in W such that $e(p) =$ false and for all $q \in D_p$, $e(q) =$ true.
(c) There exists an OR-node p in W such that $e(p) =$ true and for all $q \in D_p$, $e(q) =$ false.
(d) There exists an OR-node p in W such that $e(p) =$ false and there exists a $q \in D_p$ such that $e(q) =$ true.
(e) There exists an evidence node p in S such that $e(p) =$ false.

From Definition 2.13, conditions (a), (c), and (e) cannot hold. We now only consider conditions (b) and (d).

We can view the process of finding a suitable 0–1 solution as the propagation of information from evidence nodes through AND/OR-nodes to hypothesis nodes. The remaining conditions, (b) and (d), indicate that zero assignments do not propagate. Let M be the set of nodes p in W such that $e(p) =$ false and whose childrens' assignment permits either condition (b) or (d) to hold. Let M' be the subset of M such that each node in M' does not have a descendant also in M.

Let p be any node in M' and D_p^{-1} be the immediate children of node p. For each q in D_p^{-1}, one of the following holds:

(i) $e(q) =$ false and $r(q) =$ AND.
(ii) $e(q) =$ false and $r(q) =$ OR.
(iii) $e(q) =$ true and $r(q) =$ OR.

$e(q) =$ true and $r(q) =$ AND cannot both be true since it would violate the definition of M'. For the third combination, there exists some node $p' \neq p$ in D_q such that $e(p') =$ true.

Let W_M be the resulting WAODAG obtained by first removing all nodes and associated edges from W in M and then removing those which are not reachable from any evidence node in S. We can easily see that any AND-DAG obtained from W_M is an AND-DAG for W.

Since s is an optimal 0–1 solution for $\hat{L}_E(W)$ and $c(q, \text{false}) \leqslant c(q, \text{true})$ for any node q, for each hypothesis node p in W but not in W_M, $e(p)$ can be set to false. (It can be easily shown that if $e(p) = \text{true}$, then $c(p, \text{false}) = c(p, \text{true})$.) Now, by propagating the truth values from the hypothesis nodes, another optimal 0–1 solution will be generated. This new solution will be a best explanation for W.

We can easily determine M and the final 0–1 solution in $O(|E|)$ steps. □

Proof of Theorem 3.2. Assume that s' is not a 0–1 solution for the semi-induced linear constraint system. This implies that there exists a nonzero variable x_q such that by setting x_q to 1, this violates some constraint Q in the linear constraint system.

Case 1: $x_q \leqslant x_p$ is *violated.* Since x_q is nonzero, this implies that x_p was also nonzero to begin with. Thus x_p would have also been set to 1. Thus, this constraint is not violated.

Case 2: $\sum_{p \in D_q} x_p \geqslant x_p$ is *violated.* Since x_q is nonzero, this implies that some x_p where $p \in D_q$ is also nonzero. Thus, x_p would have also been set to 1. Thus, this constraint is not violated.

All the cases for constraint violations have been considered. No constraint is violated. Contradiction. We can similarly proved this for induced systems. □

Acknowledgement

This work has been supported by the National Science Foundation under grant IRI-8911122 and by the Office of Naval Research, under contract N00014-88-K-0589. This paper extends and improves the earlier work presented in [15,16]. Special thanks to Eugene Charniak for important pointers and critical review of this paper. Also, thanks to the anonymous reviewers whose suggestions improved this paper.

References

[1] D.E. Appelt, A theory of abduction based on model preference, in: *Proceedings AAAI Symposium on Abduction* (1990).

[2] E. Charniak and S. Husain, A new admissible heuristic for minimal-cost proofs, in: *Proceedings AAAI-91*, Anaheim, CA (1991).

[3] E. Charniak and S.E. Shimony, Probabilistic semantics for cost based abduction, in: *Proceedings AAAI-90*, Boston, MA (1990) 106–111.
[4] R.S. Garfinkel and G.L. Nemhauser, *Integer Programming* (Wiley, New York, 1972).
[5] M.R. Genesereth, The use of design descriptions in automated diagnosis, *Artif. Intell.* **24** (1984) 411–436.
[6] R.P. Goldman, A probabilistic approach to language understanding, Ph.D. Thesis, Department of Computer Science, Brown University, Providence, RI (1990).
[7] R.P. Goldman and E. Charniak, Probabilistic text understanding, in: *Proceedings Third International Workshop on AI and Statistics*, Fort Lauderdale, FL (1991).
[8] F.S. Hillier and G.J. Lieberman, *Introduction to Operations Research* (Holden-Day, San Fransisco, CA, 1967).
[9] J.R. Hobbs, M.E. Stickel, P. Martin and D. Edwards, Interpretation as abduction, in: *Proceedings 26th Annual Meeting of the Association for Computational Linguistics*, Buffalo, NY (1988).
[10] H.A. Kautz and J.F. Allen, Generalized plan recognition, in: *Proceedings AAAI-86*, Philadelphia, PA (1986).
[11] C. McMillan, *Mathematical Programming* (Wiley, New York, 1975).
[12] G.L. Nemhauser, A.H.G. Rinnooy Kan and M.J. Todd, *Optimization: Handbooks in Operations Research and Management Science, Vol. 1* (North-Holland, Amsterdam, 1989).
[13] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference* (Morgan Kaufmann, San Mateo, CA, 1988).
[14] Y. Peng and J.A. Reggia, Plausibility of diagnostic hypothese: the nature of simplicity, in: *Proceedings AAAI-86*, Philadelphia, PA (1986).
[15] E. Santos Jr, Cost-based abduction, linear constraint satisfaction, and alternative explanations, in: *Proceedings AAAI-workshop on abduction* (1991).
[16] E. Santos Jr, Cost-based abduction and linear constraint satisfaction, Tech. Report CS-91-13, Department of Computer Science, Brown University, Providence, RI (1991).
[17] E. Santos Jr, On the generation of alternative explanations with implications for belief revision, in: *Proceedings Seventh International Conference on Uncertainty in AI*, Los Angeles, CA (1991).
[18] A. Schrijver, *Theory of Linear and Integer Programming* (Wiley, New York, 1986).
[19] B. Selman and H.J. Levesque, Abductive and default reasoning: a computational core, in: *Proceedings AAAI-90*, Boston, MA (1990) 343–348.
[20] M. Shanahan, Prediction is deduction but explanation is abduction, in: *Proceedings IJCAI-89*, Detroit, MI (1989).
[21] M.E. Stickel, A Prolog-like inference system for computing minimum-cost abductive explanations in natural-language interpretation, Technical Note 451, SRI International, Menlo Park, CA (1988).