

Optimal Models of Disjunctive Logic Programs: Semantics, Complexity, and Computation

Nicola Leone, Francesco Scarcello, and V.S. Subrahmanian

Abstract—Almost all semantics for logic programs with negation identify a set, $SEM(P)$, of models of program P , as the intended semantics of P , and any model M in this class is considered a possible meaning of P with regard to the semantics the user has in mind. Thus, for example, in the case of stable models [10], choice models [30], answer sets [11], etc., different possible models correspond to different ways of “completing” the incomplete information in the logic program. However, different end-users may have different ideas on which of these different models in $SEM(P)$ is a reasonable one from their point of view. For instance, given $SEM(P)$, user U_1 may prefer model $M_1 \in SEM(P)$ to model $M_2 \in SEM(P)$ based on some evaluation criterion that she has. In this paper, we develop a logic program semantics based on *Optimal Models*. This semantics does not add yet another semantics to the logic programming arena—it takes as input an existing semantics $SEM(P)$ and a user-specified objective function Obj , and yields a new semantics $Opt(P) \subseteq SEM(P)$ that realizes the objective function *within the framework of preferred models identified already by $SEM(P)$* . Thus, the user who may or may not know anything about logic programming has considerable flexibility in making the system reflect her own objectives by building “on top” of existing semantics known to the system. In addition to the declarative semantics, we provide a complete complexity analysis and algorithms to compute optimal models under varied conditions when $SEM(P)$ is the stable model semantics, the minimal models semantics, and the all-models semantics.

Index Terms—Disjunctive logic programming, computational complexity, nonmonotonic reasoning, knowledge representation, optimization problems.



1 INTRODUCTION

THERE are now a vast number of semantics for logic programs and extensions of logic programs. All these semantics identify a set of models of logic program P as the intended semantics, $SEM(P)$, of P . From the point of view of a user of a logic programming application, she is stuck with the semantics assigned by the logic programming interpreter, but cannot specify that some of these models are “better” from her point of view, than others. Thus, the end-user (not the logic programmer building applications!) cannot make the program select and answer queries with regard to models that she deems appropriate. We show via a “cooking example” that different users may have different preferences, even when the same program and the same semantics are considered. In this paper, we shall make the following contributions:

- First, we develop the notion of an Herbrand Objective Function that allows the user to specify an expression e that can be evaluated with regard to each model. Without loss of generality, we will always assume the user wishes to find a model M in $SEM(P)$, such that e 's value is *minimized*.¹

1. As maximizing e is the same as minimizing $-e$, this framework supports maximization as well.

- N. Leone is with the Department of Mathematics, Via P. Bucci 30/B, University of Calabria, I-87030 Rende, Italy. E-mail: leone@unical.it.
- F. Scarcello is with D.E.I.S., Via P. Bucci 41/C, University of Calabria, I-87030 Rende, Italy. E-mail: scarcello@deis.unical.it.
- V.S. Subrahmanian is with the Department of Computer Science, A.V. Williams Building, University of Maryland, College Park, MD 20742. E-mail: vs@cs.umd.edu.

Manuscript received 8 Nov. 2001; revised 19 Sept. 2002; accepted 12 Dec. 2002.

For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 115344.

- A model M is optimal according to semantics $SEM(P)$ if $M \in SEM(P)$ and there is no model $M' \in SEM(P)$ such that the value of the expression e on M' is strictly less than the value of e with regard to M .
- We consider the cases when $SEM(P)$ is the set of stable models, minimal models, and all models of a *disjunctive* logic program (DLP). We look at two classes of programs—general DLPs and positive DLPs, which have no negations in rule bodies. Finally, we identify two kinds of objective functions—those that satisfy a “monotonic aggregation strategy” property, defined later in the paper, and those that may or may not do so. This leads to 12 possible combinations of semantics, DLP syntax, and aggregation-strategy type.
- For each of the above combinations, we study the complexity of four problems:
 - Checking whether a given model is optimal;
 - Brave reasoning, which checks if a given atom is true in some optimal model;
 - Cautious reasoning, which checks if a given atom is true in all optimal models;
 - Approximation, which checks if there exists a model M containing a given set of ground atoms such that the value of the Herbrand Objective Function on M is less than or equal to a given bound.

This leads to a total of 24 complexity results that are neatly summarized in Table 1 of Section 5. In particular, we show that, in general, the complexity depends upon the properties of the Herbrand Objective Functions in question, monotonicity properties decrease the complexity.

TABLE 1
Two Atomic Weight Assignments for the Program in Example 1

Atom A	$\wp_1(A)$	$\wp_2(A)$
entree(E)	C if $dish(E, entree, C, T)$ holds.	T if $dish(E, entree, C, T)$
dessert(D)	C if $dish(D, dessert, C, T)$ holds.	T if $dish(D, dessert, C, T)$
appetizer(A)	C if $dish(A, appetizer, C, T)$ holds.	T if $dish(A, appetizer, C, T)$
other atoms	0	0

- We develop a comprehensive set of algorithms that may be used to compute optimal models, both under the monotonicity assumption, and without it, for stable models, minimal models, and all models of disjunctive logic programs.

The rest of this paper is organized as follows: Section 2 motivates the paper through two examples—one is a simple “cooking” example, and the other is a more serious combinatorial auction (cf. Sandholm [31]) example. Section 3 provides basic definitions in disjunctive logic programming. In Section 4, we formally define what it means for a model M drawn from a family of models to be optimal. Such families of models can include stable models, minimal models, and all models. In Section 5, we develop the 24 complexity results discussed earlier. In Section 6, we develop algorithms to compute optimal models of disjunctive logic programs under all the cases described above. In Section 7, we compare our work with existing results in the research literature.

2 MOTIVATING EXAMPLES

In this section, we present two simple examples motivating our research: planning a dinner according to different criteria, and determining the winner of an auction.

2.1 Planning a Dinner

Suppose a chef is planning a three-course dinner (appetizer, entree, dessert) for a group of people. Different guests have different likes and dislikes. The chef would like to devise a dinner which satisfies the tastes of the guests, while at the same time, optimizing various criteria for the chef (e.g., minimizing cost, minimizing preparation time, etc.).

The relation $dish$ is a 4-ary predicate consisting of a name field specifying the name of the dish, a type field specifying whether it is an appetizer, an entree, or a dessert, a cost field estimating the unit (i.e., per serving) cost of the dish, and a time field estimating the time (in minutes) required to prepare the dish. The relation $dislikes$ is a binary predicate containing the name of a person and the name of a dish, indicating whether the person dislikes the dish. This information allows the chef to propose only menus that are compatible with all guests. Appendix A1 contains an extensional database showing the $dish$ and $dislikes$ relation and a unary guest relation.

Models: For any given collection of guests expected at dinner, one or more *menus* may be feasible. Two possible menus may be:

```
Menu1 : appetizer = caprese; entree =
        spaghetti_alla_carbonara; dessert = tiramisu
Menu2 : appetizer = samosa; entree = matar_paneer;
        dessert = rasgulla.
```

In this paper, we will argue that different *models* of a (possibly disjunctive) logic program neatly capture different menus in this example. In general, models represent scenarios that *could* be true, given certain information about the domain of discourse. Different scenarios could have different utilities to different people.

Utilities/Costs of Models: For example, the cost of preparing Menu1 may be \$6.00 per person, while the cost of preparing Menu2 may be \$5.00 per person. However, preparing Menu1 takes 25 minutes, while preparing Menu2 takes 60 minutes. A lazy chef who wishes to minimize cooking time may prefer Menu1. On the other hand, the chef who wishes to minimize cost may prefer Menu2.

Optimal Models: Informally speaking, a model is a way of satisfying the requirements encoded in a problem specification. As we have seen above, different solutions to the problem may have different costs/benefits to an end-user (i.e., to the person interested in solving the problem). However, what is important to a person is subjective and may vary from one individual to another. Consequently, any way of defining optimality of a model (or solution) must take the evaluation criterion (e.g., minimize cost of dinner, minimize preparation time) as an input.

2.2 Determining Winners in Combinatorial Auctions

A combinatorial auction is one where a set \mathcal{O} of objects are for sale and there is a set \mathcal{B} of bidders. Bidders may offer bids on a set of items. For example, bidder b_1 may bid \$500 for items a , b , and c together, and \$200 for item a alone. Formally, a *bid* is a triple of the form (b, X, p) , where $b \in \mathcal{B}$ is a bidder, $X \subseteq \mathcal{O}$ is a set of items, and $p > 0$ is a price. The auctioneer receives a set **bids** of bids. Without loss of generality, we may assume that **bids** does not contain two triples of the form $(-, X, -)$. This is because if two bids are received for the same set X of objects, the one with the lower price can be eliminated.²

Models: The task of the auctioneer is to determine which bids to “accept.” Given a set **bids** of bids, a *potential winner* is a subset **win** \subseteq **bids** such that $(b_1, X_1, p_1), (b_2, X_2, p_2) \in$

2. It is still possible that there are two bids of the form $(b_1, X, p_1), (b_2, X, p_2)$ with $p_1 = p_2$. In such a case, most auctioneers eliminate one of the two bids using some preannounced protocol—e.g., the bid received at a later time may be discarded.

$\text{win} \rightarrow X_1 \cap X_2 = \emptyset$. Each potential winner set represents a possible scenario, i.e., a possible outcome of the auction.

Utilities/Costs of Models: Potential winners may be very different according to the auctioneer's point of view. Indeed, she clearly wants to maximize her revenue and, thus, evaluates a model win according to the following measure: $R(\text{win}) = \sum_{(b,x,p) \in \text{win } P}$.

Optimal Models: The auctioneer is interested in the potential winners win such that her revenue $R(\text{win})$ is maximized. The *winner determination problem* is to find such an optimal potential winner.

In this paper, we will formalize the intuitive discussion given above so as to allow users to select models that are optimal from their point of view.

3 PRELIMINARIES

We assume the existence of an arbitrary logical language L generated by a finite set of constant, function, and predicate symbols, and an infinite set of variable symbols.

Definition 1. If $A_1, \dots, A_n, B_1, \dots, B_m, B_{m+1}, \dots, B_{m+k}$ are atoms, where $n \geq 1$ and $m, k \geq 0$, then $A_1 \vee \dots \vee A_n \leftarrow B_1 \& \dots \& B_m \& \text{not}(B_{m+1}) \& \dots \& \text{not}(B_{m+k})$ is a rule.³ A rule is normal if $n = 1$ and positive if $k = 0$. A rule with an empty body (i.e., $m = k = 0$) is called a fact. A (normal, positive) logic program is a finite set of (normal, positive) rules. For a rule r , we denote by $H(r)$ (respectively, $B(r)$) the set of literals occurring in its head (respectively, body).

A DLP has two kind of rules: 1) facts that define the so-called extensional database predicates (EDB), and 2) rules (typically with a nonempty body) that define other predicates, representing the intensional database (IDB). EDB predicates are often implemented as relations in a relational database. We often use the words "relation" and "predicate" interchangeably. Thus, if we say that a relation r is an EDB relation for a program P , we mean that for each tuple $(v_1, \dots, v_n) \in r$, a fact $r(v_1, \dots, v_n) \leftarrow$ belongs to P , even if it is not explicitly listed. These facts provide the definition of the EDB predicate r in the program P .

Given a DLP P , the Herbrand Base B_P of P is the set of all ground (i.e., variable-free) atoms that can be constructed by using the constants, predicates, and function symbols appearing in P . Given a set X of literals, we write:

$$\begin{aligned} \neg X &= \{\neg l \mid l \in X\}. \\ X^+ &= \{l \mid l \in X \text{ is a (positive) atom}\}. \\ X^- &= \{a \mid \neg a \in X \wedge a \text{ is a (positive) atom}\}. \end{aligned}$$

An interpretation I for DLP P is any subset of $B_P \cup \neg B_P$. I satisfies a ground rule r if either $I \cap H(r) \neq \emptyset$ or $B(r) \not\subseteq I$. I satisfies a nonground rule r' if it satisfies each ground instance of r' . I is consistent if $I^+ \cap I^- = \emptyset$. I is a total interpretation if $I^+ \cup I^- = B_P$. A set M of atoms is a model for P if the total consistent interpretation $M \cup \neg(B_P - M)$ satisfies each rule in P . Model M is a *minimal model* [25] for P if there is no model N for P such that $N \subset M$. Given a

3. In this paper, we consider only *negation as failure*, we do not consider strong negation. Both symbols \neg and **not** denote negation as failure throughout the paper.

DLP P , we use $\text{MM}(P)$ and $\text{MOD}(P)$ to denote the set of all minimal models of P and the set of all models of P , respectively. M is a *stable model* (or *answer set*) [11] of P if M is a minimal model of P^M where P^M is defined as follows:

$$\begin{aligned} P^M &= \{A_1 \vee \dots \vee A_n \leftarrow B_1 \& \dots \& B_m : \\ & A_1 \vee \dots \vee A_n \leftarrow B_1 \& \dots \& B_m \& \text{not}(D_1) \& \dots \& \\ & \text{not}(D_k) \text{ is a ground instance of} \\ & \text{a rule in } P \text{ and } \{D_1, \dots, D_k\} \cap M = \emptyset\}. \end{aligned}$$

We use $\text{ST}(P)$ to denote the set of all stable models of P .

We now show how to express the cooking and auctions examples in this framework using the stable model semantics.⁴

Example 1. The logic program associated with the cooking example is shown below:

```
dinner(A, E, D) ← appetizer(A) & entree(E) & dessert(D)
appetizer(A) ∨ not_take(A) ← dish(A, appetizer, C, T)
entree(A) ∨ not_take(A) ← dish(A, entree, C, T)
dessert(A) ∨ not_take(A) ← dish(A, dessert, C, T)
not_take(A) ← dish(A, -, -, -), guest(P) & dislikes(P, A)
← appetizer(A) & appetizer(B) & A ≠ B
← entree(A) & entree(B) & A ≠ B
← dessert(A) & dessert(B) & A ≠ B
chosen_dinner ← dinner(A, E, D)
← not(chosen_dinner).
```

Every stable model M of this program contains exactly one appetizer, one dessert, and one entree. Appendix A2 shows the stable models of this program with the EDB defined in Appendix A1.

Example 2. Suppose bids is the set of all bids received in a combinatorial auction. We construct a disjunctive logic program P_{bids} as follows: Two EDB predicates, bids and requires , encode the input data about the bids received by the auctioneer. In particular, for each bid $(b, x, p) \in \text{bids}$, P_{bids} contains the fact $\text{bids}(b, x, p) \leftarrow$, and the facts describing the set of items x required by the bidder b , namely, a fact $\text{requires}(x, i) \leftarrow$, for each item $i \in x$.

Moreover, an IDB predicate wins encodes the potential winner set of bids chosen by the auctioneer. It is defined by the following rule:

$$\text{wins}(B, X, P) \vee \text{loses}(B, X, P) \leftarrow \text{bids}(B, X, P).$$

Finally, we have the constraint:

$$\leftarrow \text{wins}(B_1, X_1, P_1) \& \text{wins}(B_2, X_2, P_2) \& X_1 \neq X_2 \& \text{requires}(X_1, I) \& \text{requires}(X_2, I).$$

4. In order to improve the readability of programs, we will also use rules without a head, which represents constraints to be satisfied in any allowed model. The following rule $r: \leftarrow B_1 \& \dots \& B_m \& \text{not}(D_1) \& \dots \& \text{not}(D_k)$ means that its body should evaluate to false in every intended model of the program. Such constraints can be easily expressed under the stable model semantics [8]. In particular, r is shorthand for the following normal rule: $\text{no_stable} \rightarrow B_1 \& \dots \& B_m \& \text{not}(D_1) \& \dots \& \text{not}(D_k) \& \text{not}(\text{no_stable})$. It is easy to see that in any stable model of every program containing this rule, the body of r should evaluate to false.

TABLE 2
Some Common Aggregation Strategies

Name	Description	Monotone
count(S)	cardinality of S	yes
sum(S)	sum of elements in S	yes
prod(S)	product of elements in S	no/yes ⁵
avg(S)	average of elements in S	no
min(S), max(S)	min (resp. max) of elements in S	no (min), yes (max)

5. Yes, if we require that all members of S are greater than or equal to 1.

Observe that the stable models of P_{bids} correspond exactly to the potential winners of the auction.

4 OPTIMAL MODELS

In this section, we introduce the notion of optimal model for logic programs. First, we formally define weight assignments to atoms. We then introduce aggregation strategies which provide aggregate information on a given model. Finally, we define optimal models with respect to a given semantics, i.e., with respect to a given family of intended models.

Definition 2. An atomic weight assignment, \wp , for a program P , is a map from the Herbrand Base B_P of P to \mathbf{R}^+ , where \mathbf{R}^+ denotes the set of nonnegative real numbers (including zero).

Example 3. Returning to our ‘‘Dinner’’ example, the following maps \wp_1, \wp_2 are atomic weight assignments (see Table 1):

Intuitively, the first function assigns weights to entrees based on the price/cost of those entrees, while the second function assigns weights based on the time taken to prepare those entrees.

Example 4. For the auction example, the map \wp_3 which assigns P to all atoms of the form $\text{loses}(B, X, P)$ and 0 to all other atoms, is an atomic weight assignment. Intuitively, by assigning a weight to *lost* bids and minimizing the total amount of loss, we find the bids that *maximize* the gain.

Given a set X , we use M^X to denote the set of all multisets whose elements are in X . Membership and inclusion between multisets are defined in the standard way.

Definition 3. An aggregation strategy \mathcal{A} is a map from $M^{\mathbf{R}^+}$ to \mathbf{R} .

An aggregation strategy \mathcal{A} is said to be *monotonic* if, for all $S_1, S_2 \in M^{\mathbf{R}^+}$, $S_1 \subseteq S_2$ implies that $\mathcal{A}(S_1) \leq \mathcal{A}(S_2)$. The set of aggregation strategies may be ordered as follows: $\mathcal{A}_1 \leq \mathcal{A}_2$ if for all S , $\mathcal{A}_1(S) \leq \mathcal{A}_2(S)$. Some sample aggregation strategies are given in Table 2.

Definition 4. Suppose \mathcal{A} is an aggregation strategy and \wp is an atomic weight assignment. The Herbrand Objective Function, $\text{HOF}(\wp, \mathcal{A})$ is a map from 2^{B_P} to \mathbf{R} defined as follows: $\text{HOF}(\wp, \mathcal{A})(M) = \mathcal{A}(\{\wp(A) \mid A \in M\})$.

Intuitively, $\text{HOF}(\wp, \mathcal{A})(M)$ looks at each ground atom $A \in M$, computes $\wp(A)$, and puts $\wp(A)$ in a multiset. It then applies the aggregation strategy to the multiset created in this way. This is illustrated by the following example.

Example 5. Suppose we consider the weights \wp_1 and \wp_2 described in Example 3. Consider the stable models M_1 and M_2 of Appendix A2. If we use the aggregation strategy *sum*, then:

$$\begin{aligned} \text{HOF}(\wp_1, \text{sum})(M_1) &= 6.8 & \text{HOF}(\wp_1, \text{sum})(M_2) &= 5.15 \\ \text{HOF}(\wp_2, \text{sum})(M_1) &= 60 & \text{HOF}(\wp_2, \text{sum})(M_2) &= 70 \end{aligned}$$

The following result says that, whenever a monotonic aggregation function \wp is considered, the function $\text{HOF}(\wp, \mathcal{A})$ is also monotonic.

Proposition 1. Suppose \mathcal{A} is any monotonic aggregation function. Then, $\text{HOF}(\wp, \mathcal{A})$ is monotonic, i.e., for all M, M' , if $M \subseteq M'$, then $\text{HOF}(\wp, \mathcal{A})(M) \leq \text{HOF}(\wp, \mathcal{A})(M')$.

We are now ready to define what it means for a model of a (possibly disjunctive) logic program P to be *optimal* with respect to \wp and \mathcal{A} and a selected family (e.g., all, minimal, stable) of models. Note that all semantics for logic programs identify a ‘‘family’’ of models.

Definition 5. Let P be a logic program, \wp an atomic weight assignment, and \mathcal{A} an aggregation strategy. Suppose that \mathcal{F} is a family of models of P . We say that M is an optimal \mathcal{F} -model of P with regard to (\wp, \mathcal{A}) if:

1. $M \in \mathcal{F}$, and
2. there is no model M' of P in \mathcal{F} such that $\text{HOF}(\wp, \mathcal{A})(M') < \text{HOF}(\wp, \mathcal{A})(M)$.

We use the notation $\text{Opt}(P, \mathcal{F}, \wp, \mathcal{A})$ to denote the set of all optimal \mathcal{F} -models of P with regard to (\wp, \mathcal{A}) .

We will use the expressions *optimal model*, *optimal minimal model*, and *optimal stable model* to denote models that are optimal with regard to the families $\text{MOD}(P)$, $\text{MM}(P)$, and $\text{ST}(P)$, respectively.

Example 6. Consider again our ‘‘Dinner’’ example. The different possible choices for dishes lead to 16 stable models for the cooking logic program P , all listed in Appendix A2. However, if we assign weights to dishes according to \wp_1 , i.e., according to the price/cost of those dishes, we get a unique optimal stable model

with regard to (\wp_1, sum) :⁶ $\text{Opt}(P, \text{ST}(P), \wp_1, \text{sum}) = \{ \text{dinner}(\text{caprese}, \text{idli}, \text{rasgulla}), \dots \}$. On the other hand, if we prefer to minimize cooking time, we can resort to the weight function \wp_2 . In this case, we get the following optimal stable models with regard to (sum, \wp_2) .

$\text{Opt}(P, \text{ST}(P), \wp_2, \text{sum}) = \{M_3, M_4, M_5, M_6\}$, where:

$M_3 = \{ \text{dinner}(\text{caprese}, \text{spaghetti_carbonara}, \text{tiramisu}), \dots \}$.

$M_4 = \{ \text{dinner}(\text{caprese}, \text{spaghetti_carbonara}, \text{rasgulla}), \dots \}$.

$M_5 = \{ \text{dinner}(\text{caprese}, \text{matar_paneer}, \text{tiramisu}), \dots \}$.

$M_6 = \{ \text{dinner}(\text{caprese}, \text{matar_paneer}, \text{rasgulla}), \dots \}$.

Example 7. Consider the program P_{bids} in Example 2 and the weight assignment \wp_3 in Example 4. If we use the aggregation strategy sum , then the set of optimal models $\text{Opt}(P_{\text{bids}}, \text{ST}(P_{\text{bids}}), \wp_3, \text{sum})$ encode the potential winners that minimize the sum of the revenues for bids not accepted by the auctioneer, represented by the atoms with predicate *loses*. Therefore, these models encode the potential winners that maximize the revenue for the auctioneer, and thus exactly represent the best sets of bids to be accepted for the auction at hand.

Another interesting example of applications of optimal models, this time with a nonmonotonic aggregation strategy, can be found in the technical report [20].

5 COMPLEXITY RESULTS

In this section, we analyze the complexity of the main decision problems relating to optimal models. In particular, for different families \mathcal{F} of models, P of logic programs, and assumptions on the monotonicity of the aggregation function used, we study the following problems:

Problem 1 (Checking). Given P, \wp, \mathcal{A} , and M as input, decide whether M is an optimal \mathcal{F} -model of P with regard to (\wp, \mathcal{A}) .

Problem 2 (Cautious reasoning). Given P, \wp, \mathcal{A} , and a ground literal q as input, decide whether q is true in every optimal \mathcal{F} -model of P with regard to (\wp, \mathcal{A}) , denoted by $P \models_c^{\mathcal{F}} q$.

Cautious reasoning is useful to single out the *necessary* consequences of the program. For example, the atom $\text{appetizer}(\text{caprese})$ is true in *all* models in $\text{Opt}(P, \text{ST}(P), \wp_2, \text{sum})$ as is easily seen in Example 6. This means that as long as the user wishes to minimize the time she spends on cooking, she is forced to choose *caprese* as the appetizer. On the other hand, $\text{dessert}(\text{tiramisu})$ is *not* true in all models in $\text{Opt}(P, \text{ST}(P), \wp_2, \text{sum})$.

Problem 3 (Brave reasoning). Given P, \wp, \mathcal{A} , and a ground literal q as input, decide whether there exists an optimal \mathcal{F} -model M of P with regard to (\wp, \mathcal{A}) such that q is true with regard to M , denoted by $P \models_b^{\mathcal{F}} q$.

Unlike cautious reasoning, *brave* reasoning finds out whether there exists an optimal model in which a literal is

true. In the Dinner example (cf. Example 6), we see that $\text{dessert}(\text{tiramisu})$ is a brave consequence of P , but not a cautious consequence.

Problem 4 (Approx($P, \wp, \mathcal{A}, n, S$)). Given P, \wp, \mathcal{A} , a set of ground literals S , and a real number n as input, decide whether there exists a model M in \mathcal{F} such that: 1) $\text{HOF}(\wp, \mathcal{A})(M) \leq n$, and 2) every literal in S is true with regard to M .

A successful solution to Problem 4 allows the user to approximate optimal models.

Example 8. Suppose the cook wishes to ask the following queries:

1. “Is there some way for me to fix dinner at a price less than or equal to \$6.50 per head?” This corresponds to an instance of Problem 4 where $S = \emptyset$ and $n = 6.50$.
2. Suppose, for some reason, the cook wants to make *samosa* as the appetizer. Then, she may want to ask the query: “Is there some way for me to fix dinner including a *samosa* appetizer at a price less than or equal to \$6.50 per head?” This corresponds to an instance of Problem 4 where $S = \{\text{appetizer}(\text{samosa})\}$ and $n = 6.50$.

The classes Σ_k^P , Π_k^P , and Δ_k^P of the Polynomial Hierarchy (PH) are defined as follows: $\Delta_0^P = \Sigma_0^P = \Pi_0^P = P$. For all $k \geq 1$, $\Delta_k^P = P^{\Sigma_{k-1}^P}$, $\Sigma_k^P = \text{NP}^{\Sigma_{k-1}^P}$, $\Pi_k^P = \text{co-}\Sigma_k^P$.

In particular, $\text{NP} = \Sigma_1^P$, $\text{co-NP} = \Pi_1^P$, and $\Delta_2^P = P^{\text{NP}}$. Here, P^C and NP^C denote the classes of problems that are solvable in polynomial time on a deterministic (respectively, nondeterministic) Turing machine with an oracle for any problem π in the class C . The reader seeking details of complexity theory is referred to [28], [13].

Let $\text{max_hof}(\wp, \mathcal{A}, P)$ denote the highest value that $\text{HOF}(\wp, \mathcal{A})$ may assume on P . That is, $\text{max_hof}(\wp, \mathcal{A}, P) = \text{max}(\{\text{HOF}(\wp, \mathcal{A})(M) \mid M \subseteq B_P\})$. In the complexity analysis, we assume that the atomic weight assignment is part of the input and weights are (nonnegative) integers. We also assume that given \wp, \mathcal{A}, P , and M , both $\text{HOF}(\wp, \mathcal{A})(M)$ and $\text{max_hof}(\wp, \mathcal{A}, P)$ are polynomial-time computable and integer valued (note that all results may be extended easily to the general case of real-valued polynomial-time functions, by using standard techniques). We analyze the complexity of the propositional case (that is, we assume that programs are ground); the results can be easily extended to *data complexity* [34].

5.1 Overview of Results

Table 3 summarizes all the complexity results of this paper. Its first column specifies the family of models (Stable models, Minimal models, or All models); the second column specifies possible restrictions on the aggregation strategy; the remaining columns (3 to 6) report the complexity results. Each of these columns refers to a specific task (Checking, Brave Reasoning, Cautious Reasoning, and Approx, in the specified order). All results we report are completeness results, i.e., when we say that a problem P has complexity C , then we have proved that P is C -complete under polynomial-time transformations.

6. Note that, in this example, whenever we have to list a stable model of the program, we just list the atoms that characterize the model, namely, the atoms with predicates *dinner*. We do not list EDB atoms, which are true in every stable model, and auxiliary atoms, like *not_take*.

TABLE 3
Computational Complexity

Models	Aggregation Strategy	Checking	Brave Reasoning	Cautious Reasoning	Approx
Stable	Monotonic	Π_2^P	Δ_3^P	Δ_3^P	Σ_2^P
Stable	Arbitrary	Π_2^P	Δ_3^P	Δ_3^P	Σ_2^P
Minimal	Monotonic	co-NP	Σ_2^P	Π_2^P	Σ_2^P
Minimal	Arbitrary	Π_2^P	Δ_3^P	Δ_3^P	Σ_2^P
All	Monotonic	co-NP	Δ_2^P	Δ_2^P	NP
All	Arbitrary	co-NP	Δ_2^P	Δ_2^P	NP

Consider the results displayed in Table 3. Checking is Π_2^P -complete for stable models. Thus, checking if an interpretation I is an optimal stable model is located one level higher, in the polynomial hierarchy, than checking if it is an ordinary stable model (which is “only” co-NP-complete). Intuitively, this increase of complexity is because of two “orthogonal” sources of complexity: 1) the stability check (i.e., proving that the interpretation is a stable model); and 2) the optimality check (i.e., proving that the value of the objective function on I is minimum in the family of the stable models of the program). Imposing monotonicity on the aggregation strategy does not help for stable models (the complexity remains Π_2^P), because there is no relationship between an interpretation I being stable and $\text{HOF}(\varphi, \mathcal{A})(I)$ being optimal. In this respect, the situation is different for Minimal Models. While Checking is Π_2^P -complete (as for stable models) under arbitrary aggregation strategies, its complexity decreases to co-NP (i.e., it becomes the same as the complexity of minimality checking) for monotonic aggregation strategies. Indeed, in this case, the two minimality criteria to be considered for the check (the minimality of the model and the optimality of its value under the objective function) work in parallel: If an interpretation A is smaller than B (i.e., it is preferable to B under the subset inclusion criterion determining minimal models), then $\text{HOF}(\varphi, \mathcal{A})(A) \leq \text{HOF}(\varphi, \mathcal{A})(B)$. Thus, the complexity of the two sources is not summed up in this case. For the family of All models, the complexity of Checking remains co-NP even if the aggregation strategy is arbitrary. The reason is that compared to stable models or minimal models, one source of complexity is eliminated in this case as checking membership in the family of all models is polynomial (while it is co-NP-complete for the families of stable models and minimal models). Thus, the only hard source of complexity is the optimality check which causes the co-NP-completeness of this problem.

Approx is complementary to Checking. To prove an instance of Approx, one has to find a member A of the models’ family (e.g., a stable model or a minimal model), whose value of the objective function is lower than the given bound (and A contains the specified literals). Similarly, to disprove an instance of Checking: find a member A of the

models’ family whose value of the objective function is lower than the value of the objective function on the model to be checked. By this intuition, the complexity of Approx is complementary (Π_2^P/Σ_2^P , co-NP/NP) to the complexity of Checking, nearly always. The only exception is the case of Monotonic aggregation strategy for the family of Minimal models, where Approx cannot enjoy the “parallelism” among the two sources of complexity which mitigated the complexity of Checking for this case (see the comment above).

Reasoning (Brave and Cautious) brings an additional source of complexity, compared to the previous problems: the (possibly) exponential number of optimal models which should be analyzed (in the worst case) to decide whether the given literal is true or not. Nevertheless, in most cases, the applicability of smart techniques mitigates the complexity of these reasoning problems, allowing us to solve them deterministically by a polynomial number of calls to oracles solving Checking or Approx (an oracle for a problem P is like a subroutine taking an instance of P in input and returning its yes/no solution). Thus, compared to Checking, the complexity of Brave Reasoning and Cautious Reasoning increases only “mildly,” it is located only half a level higher in the polynomial hierarchy, stepping from co-NP and Π_2^P to Δ_2^P and Δ_3^P , respectively. An interesting exception is reasoning with the family of Minimal models under monotonic aggregation strategies. In this case, the problem remains of the same complexity as standard minimal model reasoning (Σ_2^P and Π_2^P for brave and cautious reasoning, respectively).

If we consider positive programs, we cannot find any difference as far as both Minimal model semantics and All model semantics are concerned, because these semantics are syntax independent (unlike Stable model semantics, $a \leftarrow \text{not } b$ is precisely the same as $a \vee b$ for Minimal models and All models semantics). Disallowing negation, however, makes a difference for Stable model semantics and lowers the complexity over monotonic aggregation strategies, which becomes the same as for the case of minimal models (indeed, Stable models degenerate to Minimal models when negation is disallowed).

5.2 General Reduction among Problems

In this section, we show some general relationships among reasoning problems in the optimal models framework. We next prove that, for any of *all*, *stable*, *minimal* family of models, brave and cautious reasoning problems have the same computational complexity, if we consider general aggregation strategies.

Theorem 1 (Brave Reasoning versus Cautious Reasoning Under Arbitrary Aggregation). *Let P be a program, \mathcal{F} a family of models in $\{\text{all}, \text{stable}, \text{minimal}\}$, \wp a weight function, and \mathcal{A} an aggregation strategy. Then, there exists a program P' , a weight function \wp' , and an arbitrary (possibly non monotonic) aggregation strategy \mathcal{A}' , all polynomially constructible, s.t. Brave reasoning for P with regard to (\wp, \mathcal{A}) reduces to Cautious reasoning for P' with regard to (\wp', \mathcal{A}') , and viceversa.*

Proof. Let P be a logic program, q an atom in B_P , \wp an atomic weight assignment, \mathcal{A} an aggregation strategy, and q' a fresh atom. Define a program $P' = P \cup \{q \leftarrow q'; q' \leftarrow q\}$. Moreover, consider the following atomic weight assignment \wp' and aggregation strategies \mathcal{A}' and \mathcal{A}'' :

$$\begin{cases} \wp'(p) = \wp(p) & \forall p \neq q' \\ \wp'(q') = c_{q'} & \text{where } c_{q'} \neq \wp(p) \forall p \neq q' \end{cases}$$

$$\mathcal{A}'(X) = \begin{cases} \mathcal{A}(X) & \text{if } c_{q'} \notin X \\ \mathcal{A}(X - \{c_{q'}\}) + 1 & \text{if } c_{q'} \in X \end{cases}$$

$$\mathcal{A}''(X) = \begin{cases} \mathcal{A}(X - \{c_{q'}\}) & \text{if } c_{q'} \in X \\ \mathcal{A}(X) + 1 & \text{if } c_{q'} \notin X \end{cases}$$

Then, the theorem follows from the following facts.

FACT a. $P \models_c^{\mathcal{F}} q$ with regard to (\wp, \mathcal{A}) iff $P' \models_b^{\mathcal{F}} q'$ with regard to (\wp', \mathcal{A}') .

FACT b. $P \models_c^{\mathcal{F}} \neg q$ with regard to (\wp, \mathcal{A}) iff $P' \models_b^{\mathcal{F}} \neg q'$ with regard to (\wp', \mathcal{A}') .

FACT c. $P \models_b^{\mathcal{F}} q$ with regard to (\wp, \mathcal{A}) iff $P' \models_c^{\mathcal{F}} q'$ with regard to (\wp', \mathcal{A}'') .

FACT d. $P \models_b^{\mathcal{F}} \neg q$ with regard to (\wp, \mathcal{A}) iff $P' \models_c^{\mathcal{F}} \neg q'$ with regard to (\wp', \mathcal{A}'') .

To see that Fact a holds, assume that q is a cautious consequence of P with regard to (\wp, \mathcal{A}) and let $\mathcal{M} = \text{Opt}(P, \mathcal{F}, \wp, \mathcal{A})$ be the set of optimal \mathcal{F} -models of P . Therefore, $\forall M \in \mathcal{M}$, $q \in M$, and $\text{HOF}(\wp, \mathcal{A})(M) = b$, where b is the minimum (and, hence, optimal) value of the Herbrand Objective Function over the \mathcal{F} -models of P . Now, consider program P' . For each model M' of P' , q is true if and only if q' is true. Moreover, M' is a model of P' if and only if $M' - \{q'\}$ is a model of P . Observe that the aggregation function \mathcal{A}' assigns an extra penalty of 1 to all and only those models of P' containing q' and, hence, q . Thus, for each \mathcal{F} -model M' of P' containing q' , $\text{HOF}(\wp', \mathcal{A}')(M') = b + 1$. However, since all optimal \mathcal{F} -models of P contain q , for each \mathcal{F} -model M'' of P without q , $\text{HOF}(\wp, \mathcal{A})(M'') \geq b + 1$. By construction, M'' is also a model of P' and $\text{HOF}(\wp', \mathcal{A}'')(M'') = \text{HOF}(\wp, \mathcal{A})(M'') \geq b + 1$. It follows that such models cannot beat all the models containing q and q' and, thus, q' is a brave consequence of P' with regard to (\wp', \mathcal{A}') .

We can see that the converse holds in a similar way. Assume that q' is a brave consequence of P' with regard to (\wp', \mathcal{A}') and note that \mathcal{A}' assigns an extra penalization to all and only the models containing q' . Thus, for each optimal \mathcal{F} -model M' of P' with regard to (\wp', \mathcal{A}') that contains q and q' , $M = M' - \{q'\}$ is an optimal \mathcal{F} -model of P with regard to (\wp, \mathcal{A}) , and there are no other optimal \mathcal{F} -models of P with regard to (\wp, \mathcal{A}) . Thus, q is a cautious consequence of P with regard to (\wp, \mathcal{A}) .

To see that the remaining facts above hold, one can adopt the same reasoning as for Fact a. Indeed, from case to case, we only change in a suitable way how to assign extra penalties to models containing (or not containing) the special atom q' . \square

Note that the reduction above does not preserve the monotonicity of the aggregation strategy, i.e., even if \mathcal{A} is a monotonic aggregation strategy, \mathcal{A}' may, in general, not be monotonic. However, for stable model semantics and all model semantics, brave and cautious reasoning have the same complexity even with regard to monotonic aggregation strategies, as stated by the following theorem. (For space reasons, the proof is omitted here, but it is available in [20].)

Theorem 2 (Brave Reasoning versus Cautious Reasoning under Monotonic Aggregation). *Let P be a general program, \mathcal{F} a family of models in $\{\text{all}, \text{stable}\}$, \wp a weight function, and \mathcal{A} a monotonic aggregation strategy. Then, there exist a program P' , a weight function \wp' , and a monotonic aggregation strategy \mathcal{A}' , all of them polynomially constructible, s.t. Brave reasoning for P with regard to (\wp, \mathcal{A}) reduces to Cautious reasoning for P' with regard to (\wp', \mathcal{A}') , and viceversa.*

There exists a very natural relation between traditional—i.e., nonoptimal—brave reasoning and the approximation problem. This is stated by the following proposition, whose proof is straightforward.

Proposition 2 (Brave Reasoning versus Approx). *Let P be a program, L a literal, and \mathcal{F} a family of models. There exists an \mathcal{F} -model M for P s.t. L is true with regard to M if and only if $(P, \wp_0, \text{sum}, 0, \{L\})$ is a yes-instance of Approx, where \wp_0 denotes the weight function which assigns 0 to each atom in B_P .*

Thus, for any family of models, traditional brave reasoning reduces to Approx. Note that the aggregation strategy used in the above reduction is monotonic.

Moreover, traditional brave reasoning (without objective functions) is clearly a special case of optimal brave reasoning. To see this, consider any trivial aggregation strategy which assigns some fixed value to every given set. Then, the Herbrand objective function is a constant function and all the models in the given family play the same role in the reasoning task, as in the case of traditional reasoning. Clearly, the same argument applies to the relationship between traditional cautious reasoning and optimal cautious reasoning.

5.3 Selected Proofs

For space reasons, we can only include proofs of some results from Table 3. *Proofs of all results are Web-accessible [20]*. The proofs presented here focus on the complexity of checking, cautious, brave, and approximate reasoning for optimal stable model semantics.

5.3.1 Checking

Theorem 3 (General/Positive Program, Arbitrary Aggregation). *Given a disjunctive program P , \wp , \mathcal{A} , and M as input, deciding whether M is an optimal stable model of P with regard to (\wp, \mathcal{A}) is Π_2^P -complete. Hardness holds even if P is a positive program and M is a stable model of P .*

Proof Π_2^P -Membership. We can verify that M is not an optimal stable model as follows. Guess $M_1 \subseteq B_P$, and check that: either 1) $[M_1$ is a stable model of $P]$ and $[(\wp, \mathcal{A})(M_1) < \text{HOF}(\wp, \mathcal{A})(M)]$, or 2) M is not a stable model of P . Since both (1) and (2) can be done by a single call to an NP oracle [7], this problem is in Σ_2^P and, as a consequence, Checking is in Π_2^P .

Π_2^P -Hardness. Given a disjunctive positive program P and a literal $\neg q$, deciding whether $\neg q$ is a cautious consequence of P under stable semantics (i.e., deciding whether q does not belong to any stable model) is Π_2^P -hard [7]. We reduce this problem to optimal stable model checking.

Let q' be a fresh atom and P' be the (positive) program obtained from P by: 1) inserting the atom q' in the head of every rule of P . It is easy to see that $\text{MM}(P') = \{q'\} \cup \text{MM}(P)$.

Since both P and P' are positive programs, $\text{ST}(P) = \text{MM}(P)$ and $\text{ST}(P') = \text{MM}(P')$. Therefore, $\text{ST}(P') = \{q'\} \cup \text{ST}(P)$.

Let \wp be a one-to-one function from B_P into $[1 \dots |B_P|]$ (\wp assigns an identifier to each atom). Define \mathcal{A} as follows: 1) $\mathcal{A}(K) = 0$ if K is the image (under \wp) of a set N of atoms containing q , 2) $\mathcal{A}(K) = 1$ otherwise (i.e., N does not contain q). (Note that \mathcal{A} is not monotonic.) Now, we have that: 1) $\text{HOF}(\wp, \mathcal{A})(\{q'\}) = 1$, and 2) for each $M \in (\text{ST}(P') - \{q'\})$ s.t. $q \in M$, $\text{HOF}(\wp, \mathcal{A})(M) = 0$. Therefore, $\{q'\}$ is an optimal stable model of P' if and only if [no stable model of P contains q]. That is, $\{q'\}$ is an optimal stable model of P' iff $\neg q$ is a cautious consequence of P . Hence, Checking is Π_2^P -hard; moreover, hardness holds even for positive programs, as the used program P' is positive. Note that $\{q'\}$ is a stable model of P' . \square

Hence, optimal stable model checking for disjunctive programs is Π_2^P -complete and the complexity of the problem does not decrease on positive programs. The aggregation strategy used in the proof of Π_2^P -hardness is arbitrary (not monotonic). One may thus wonder whether assuming the monotonicity of this strategy has an impact on the complexity of optimal stable model checking. The following result says that the answer to this question is “no.”

Theorem 4 (General Program, Monotonic Aggregation).

Given a disjunctive program P , \wp , \mathcal{A} , and M as input, deciding whether M is an optimal stable model of P with regard to (\wp, \mathcal{A}) is Π_2^P -complete even if \mathcal{A} is a monotonic aggregation strategy. Hardness holds even if M is a stable model of P .

Proof. From Theorem 3, it remains to prove only hardness.

Given a (general) disjunctive program P and an atom q , deciding whether q is a cautious consequence of P under stable model semantics (i.e., deciding whether q belongs to all stable model) is Π_2^P -hard [7]. We reduce this problem to optimal stable model checking with monotonic aggregation functions.

Let P' be the program obtained from P by: 1) inserting the literal $\text{not}(q)$ in the body of every rule of P , and 2) adding the rule $q \vee q' \leftarrow$, where q' is a fresh atom. It is easy to see that $\text{ST}(P') = \{q\} \cup \{M \cup \{q'\} \mid M \in \text{ST}(P) \wedge q \notin M\}$.

Let $\mathcal{A} = \text{sum}$ and \wp be the following atomic weight assignment: 1) $\wp(q) = 1$, and 2) $\wp(x) = 0$, for each $x \in B_{P'} - \{q\}$. Now, we have that: 1) $\text{HOF}(\wp, \text{sum})(\{q\}) = 1$, and 2) for each $M \in (\text{ST}(P') - \{q\})$, $\text{HOF}(\wp, \mathcal{A})(M) = 0$. Therefore, $\{q\}$ is an optimal stable model of P' if and only if [it is the only stable model of P'] if and only if [every stable model of P contains q]. That is, $\{q\}$ is an optimal stable model of P' iff q is a cautious consequence of P (we are done). Hence, Checking is Π_2^P -hard even if the monotonicity of aggregation strategy is imposed (as we have used the monotonic strategy sum). \square

In order to decrease the complexity of optimal stable model checking, we have to assume both that the programs involved are positive and that the aggregation strategy involved is monotonic.

Theorem 5 (Positive Program, Monotonic Aggregation).

Given a positive disjunctive program P , \wp , M , and a monotonic aggregation strategy \mathcal{A} as input, deciding whether M is an optimal stable model of P with regard to (\wp, \mathcal{A}) is co-NP-complete. Hardness holds even if M is known to be a stable model.

Proof co-NP-Membership. On positive programs, minimal and stable models coincide. As a consequence, optimal stable models and optimal minimal models also coincide. We can thus verify that M is not an optimal stable model by checking that it is not an optimal minimal model. That is, guess $M_1 \subseteq B_P$, and check that: either (1) $[M_1$ is a model of $P]$ and $[\text{HOF}(\wp, \mathcal{A})(M_1) < \text{HOF}(\wp, \mathcal{A})(M)]$, or $[M_1$ is a model of $P]$ and $[M_1 \subseteq M]$, or 3) M is not a model of P . Since (1), (2), and (3) are polynomial time tasks, this problem is in NP and, as a consequence, Checking optimal stable models is in co-NP.

co-NP-Hardness. Given a disjunctive positive program P and an atom q , it is well-known and easy to see that deciding whether $P \models q$ is co-NP-hard. We reduce this problem to optimal stable model checking.

Let q' be a fresh atom and P' be the (positive) program obtained from P by: 1) inserting the atom q' in the head of every rule of P . It is easy to see that $\text{MM}(P') = \{q'\} \cup \text{MM}(P)$. Since both P and P' are positive programs, $\text{ST}(P) = \text{MM}(P)$ and $\text{ST}(P') = \text{MM}(P')$. Therefore, $\text{ST}(P') = \{q'\} \cup \text{ST}(P)$.

Let \wp be the following weight function: $\wp(p) = 1$ for any atom $p \in B_P$ s.t. $p \neq q$; $\wp(q) = \wp(q') = |B_P|$. The aggregation function \mathcal{A} is sum . Note that sum is monotonic.

Let M be a model of P s.t. $q \notin M$. By definition of \wp and \mathcal{A} , we have that $\mathcal{A}(M) < |B_P|$. Now, consider the set $\{q'\}$. $\{q'\}$ is a stable model for P' and $\mathcal{A}(\{q'\}) = |B_P|$. Hence, $\{q'\}$ is an optimal stable model of P' with regard to (\wp, \mathcal{A}) iff [every model of P contains q], i.e., iff $[P \models q]$. As a consequence, Checking is co-NP-hard. \square

5.3.2 Approx

Theorem 6 (General/Positive Program, Arbitrary Aggregation). *Under stable model semantics, given a disjunctive general program P , \wp , \mathcal{A} , a set of ground literals S , and a real number n as input, the problem $\text{Approx}(P, \wp, \mathcal{A}, n, S)$ is Σ_2^P -complete. Hardness holds even if P is a positive program.*

Proof. We can decide the problem as follows: Guess $M \subseteq B_P$, and check that: 1) M is a stable model of P , 2) $\text{HOF}(\wp, \mathcal{A})(M) \leq n$, and 3) every literal in S is true with regard to M . Clearly, property (2) and (3) can be checked in polynomial time, while (1) can be decided by a single call to a NP oracle. The problem is therefore in Σ_2^P .

Hardness follows from Proposition 2 and from the Σ_2^P -hardness of (traditional) brave reasoning for disjunctive positive programs under stable model semantics [7]. \square

5.3.3 Reasoning

In this section, we will study the complexity of brave and cautious reasoning under the optimal stable model semantics. Some hardness results will be proved by reductions from quantified Boolean Formulas (QBFs) into problems related to optimal models; the disjunctive programs used will be appropriate adaptations and extensions of the disjunctive program reported below (which was first described in [7]).

Let Φ be a formula of the form $\forall Y E$, where E is a Boolean expression over propositional variables from $X \cup Y$, where $X = \{x_1, \dots, x_n\}$ and $Y = \{y_1, \dots, y_m\}$. We assume that E is in 3DNF, i.e., $E = D_1 \vee \dots \vee D_r$ and each $D_i = L_{i,1} \wedge L_{i,2} \wedge L_{i,3}$ is a conjunction of literals $L_{i,j}$. Define the following DLP $LP(\Phi)$:

$$\begin{aligned} x_i \vee x'_i &\leftarrow \text{ for each } i = 1, \dots, n \\ y_j \vee y'_j &\leftarrow y_j \leftarrow w \quad y'_j \leftarrow w \leftarrow y_j \wedge y'_j \\ &\text{ for each } j = 1, \dots, m \\ w &\leftarrow \sigma(L_{k,1}) \wedge \sigma(L_{k,2}) \wedge \sigma(L_{k,3}) \\ &\text{ for each } k = 1, \dots, r, \end{aligned}$$

where σ maps a literal L to the atom x'_i , if $L = \neg x_i$ for some $i = 1, \dots, n$; to the atom y'_j , if $L = \neg y_j$ for some $j = 1, \dots, m$; and to L itself, otherwise.

Intuitively, x'_i corresponds to $\neg x_i$ and y'_j corresponds to $\neg y_j$. It is very important to note that $LP(\Phi)$ is always positive.

Given a truth assignment $\phi(X)$ to $X = \{x_1, \dots, x_n\}$, we denote by $M_\phi \subseteq B_{LP(\Phi)}$ the interpretation

$$M_\phi = \{x_i | \phi(x_i) = \text{true}\} \cup \{x'_i | \phi(x_i) = \text{false}\} \cup \{w\} \cup \{y_1, \dots, y_m\} \cup \{y'_1, \dots, y'_m\}.$$

Moreover, given an interpretation M of $LP(\Phi)$, we denote by ϕ_M the truth assignment to $X = \{x_1, \dots, x_n\}$ such that $\phi_M(x_i) = \text{true}$ iff $x_i \in M$.

Let E be a Boolean expression and ϕ be a truth assignment for the variables in the set X . $E_{\phi(X)}$ denotes the Boolean expression E where each variable $x \in X$ is replaced by its truth value $\phi(x)$.

Then, the following lemma follows immediately from Theorems 3.3 and 3.5 of [7], which precisely state the correspondence between such a formula Φ and its associated program $LP(\Phi)$.

Lemma 1. *Let $\Phi = \forall Y E$ and $LP(\Phi)$ be the formula and the disjunctive logic program defined above. Consider the set A of the truth assignments $\phi(X)$ to $X = \{x_1, \dots, x_n\}$ such that $\Phi_\phi = \forall Y E_{\phi(X)}$ is a valid formula, and let B be the set of the stable models of $LP(\Phi)$ which contain w . Then, there is a one-to-one correspondence between A and B . In particular:*

1. if $\Phi_\phi = \forall Y E_{\phi(X)}$ is valid, then $M_\phi \in \text{ST}(LP(\Phi))$, and
2. if $M \in \text{ST}(LP(\Phi))$ contains w , then Φ_{ϕ_M} is valid.

Note that $LP(\Phi)$ is constructible from Φ in polynomial time and that $\text{ST}(LP(\Phi)) = \text{MM}(LP(\Phi))$, as $LP(\Phi)$ is a positive program. We are now ready to derive the complexity of Brave reasoning with optimal models.

Theorem 7 (General/Positive Program, Arbitrary Aggregation). *Given a disjunctive program P , \wp , \mathcal{A} , and a ground literal q as input, deciding whether q is true in some optimal stable model of P with regard to (\wp, \mathcal{A}) is Δ_3^P -complete. Hardness holds even if P is a positive program.*

Proof Δ_3^P -Membership. We first call a Σ_2^P oracle to verify that P admits stable models (otherwise, q cannot be a brave consequence). We then compute $k = \text{max_hof}(\wp, \mathcal{A}, P)$ (this is done in polynomial time by assumption). After that, by binary search on $[0..k]$, we determine the cost Σ of the optimal stable models, by a polynomial number of calls to a (Σ_2^P) oracle deciding $\text{approx}(P, \wp, \mathcal{A}, n, \emptyset)$ ($n = k/2$ on the first call; then, if the oracle answers yes, " $n = k/4$; otherwise, n is set to $k/2 + k/4$, etc., according with standard binary search). (Observe that the number of calls to the oracle is logarithmic in k , and, as a consequence, is polynomial in the size of the input.) Finally, a call to the oracle $\text{approx}(P, \wp, \mathcal{A}, \Sigma, \{q\})$ verifies that q is true in some optimal stable model of P .

Δ_3^P -Hardness. Let $\langle x_1, \dots, x_n \rangle$ be in lexicographical order. Then, Δ_3^P -hardness is shown by a reduction from the following Δ_3^P -complete problem (see, e.g., [13]): Decide whether the lexicographically minimum truth assignment $\phi(X)$, $X = \{x_1, \dots, x_n\}$, such that $\Phi_\phi = \forall Y E_{\phi(X)}$ is valid, satisfies $\phi(x_n) = \text{true}$ (where such a ϕ is known to exist). W.l.o.g. we assume that E is in 3DNF of the form defined above.

Consider the (positive) program $LP(\Phi)$ defined above. Let $\text{ST}_w(LP(\Phi))$ denote the set of the stable models of $LP(\Phi)$ which contain w .

From Lemma 1, we know that there is a one-to-one correspondence between $\text{ST}_w(LP(\Phi))$ and the set of truth assignments ϕ which make Φ_ϕ valid.

Now, let \wp be the atomic weight assignment such that: 1) $\wp(x_i) = 2^{n-i}$ ($1 \leq i \leq n$), 2) $\wp(w) = 2^n$, and 3) $\wp(y) = 0$, if $y \notin \{w\} \cup \{x_1, \dots, x_n\}$. Moreover, let \mathcal{A} be the aggregation

strategy defined as follows: 1) $\mathcal{A}(X) = 2^n$ if $2^n \notin X$, 2) $\mathcal{A}(X) = \text{sum}(X) - 2^n$, if $2^n \in X$.

Then, the Herbrand objective function $\text{HOF}(\wp, \mathcal{A})$ assigns 2^n to the stable models which do not contain w , while it assigns the sum of the weights of the x_i s in M to each stable model M containing w (the models in $\text{ST}_w(LP(\Phi))$ are thus the candidates for optimal models, as their values is less than 2^n). $\text{HOF}(\wp, \mathcal{A})$ induces a total order on $\text{ST}_w(LP(\Phi))$. In particular, given two stable models M and M' in $\text{ST}_w(LP(\Phi))$, $\text{HOF}(\wp, \mathcal{A})(M) > \text{HOF}(\wp, \mathcal{A})(M')$ iff the truth assignment ϕ_M is greater than $\phi_{M'}$ in the lexicographically order. Therefore, $LP(\Phi)$ has a unique optimal stable model M (actually, M is in $\text{ST}_w(LP(\Phi))$), corresponding to the lexicographically minimum truth assignment ϕ_{\min} such that $\Phi_{\phi_{\min}} = \forall Y E_{\phi_{\min}(X)}$ is valid. Hence, the lexicographically minimum truth assignment $\phi_{\min}(X)$ making $\Phi_{\phi_{\min}}$ valid fulfills $\phi_{\min}(x_n) = \text{true}$ if and only if x_n is true in the optimal stable model of $LP(\Phi)$ with regard to (\wp, \mathcal{A}) (that is, iff x_n is a brave consequence of $LP(\Phi)$ with regard to (\wp, \mathcal{A})). Therefore, brave reasoning is Δ_3^P -hard. Moreover, hardness holds even if the logic program is positive, as the utilized program $LP(\Phi)$ positive. \square

The following result says that even if we require the aggregation strategy to be monotonic, the hardness result presented above continues to persist. Thus, the complexity does not decrease in the case of monotonic aggregation strategies.

Theorem 8 (General Program, Arbitrary/Monotonic Aggregation). *Given a disjunctive program P , \wp , \mathcal{A} , and a ground literal q as input, deciding whether q is true in some optimal stable model of P with regard to (\wp, \mathcal{A}) is Δ_3^P -complete even if \mathcal{A} is a monotonic aggregation strategy.*

Proof. From Theorem 7, it remains to prove only hardness. We provide a reduction from the same problem used in the proof of that theorem, and show that we can impose monotonicity on the aggregation strategy if we allow negation in the logic program.

Consider the program $LP'(\Phi) = LP(\Phi) \cup \{w \leftarrow \text{not}(w)\}$. We have now that every stable model of $LP'(\Phi)$ must contain w , that is, $\text{ST}(LP'(\Phi)) = \text{ST}_w(LP(\Phi))$. Therefore, there is a one-to-one correspondence between $\text{ST}(LP'(\Phi))$ and the set of truth assignments ϕ which make Φ_ϕ valid.

Now, let \wp be the atomic weight assignment \wp such that: 1) $\wp(x_i) = 2^{n-i}$ ($1 \leq i \leq n$), and 2) $\wp(y) = 0$ if $y \notin \{x_1, \dots, x_n\}$. Take the monotonic aggregation strategy sum .

Then, the Herbrand objective function $\text{HOF}(\wp, \text{sum})$ induces a total order on $\text{ST}(LP'(\Phi))$. In particular, given two stable models M and M' in $\text{ST}(LP'(\Phi))$, $\text{HOF}(\wp, \mathcal{A})(M) > \text{HOF}(\wp, \mathcal{A})(M')$ if and only if the truth assignment ϕ_M is greater than $\phi_{M'}$ in the lexicographically order. Therefore, $LP'(\Phi)$ has a unique optimal stable model M , corresponding to the lexicographically minimum truth assignment ϕ_{\min} such that $\Phi_{\phi_{\min}} = \forall Y E_{\phi_{\min}(X)}$ is valid. Hence, the lexicographically minimum truth assignment $\phi_{\min}(X)$ making $\Phi_{\phi_{\min}}$ valid fulfills $\phi_{\min}(x_n) = \text{true}$ if and only if x_n is true in the optimal stable model of $LP'(\Phi)$ with regard to (\wp, \mathcal{A}) (that is, iff x_n is a brave

consequence of $LP(\Phi)$ with regard to (\wp, \mathcal{A})). Therefore, brave reasoning is Δ_3^P -hard, even if we require the monotonicity of the aggregation strategy (as the used strategy sum is monotonic). \square

6 ALGORITHMS FOR COMPUTING OPTIMAL MODELS

In contrast to algorithms that compute stable models of DLPs that use a backtracking strategy, the need for optimization necessitates that we develop branch and bound algorithms to compute optimal stable models. Our key task in developing such algorithms is to efficiently prune the search space. Our approach to computing optimal stable models exploits 1) two suitable operators for deriving both positive and negative necessary consequences of a program from a given interpretation; and 2) the notion of possibly-true literals, that allows us to select candidates to put in an optimal stable model. This way, partial interpretations are extended only by literals that can lead to some stable model, or, in the case of the immediate-consequences operators, that must belong to every stable model (that include the interpretation at hand). Other tricks, e.g., pruning by exploiting the minimality property for positive programs and the monotonicity of aggregation strategies, are described and exemplified in detail when we describe the algorithms below.

Some of these pruning strategies cannot be applied to the all models case (e.g., the notion of possibly-true literals is meaningless in this case). In principle, all literals are candidates for extending an interpretation to a “plain” model. Optimal minimal-models can be computed by modifying the optimal stable model computation algorithm (of positive programs) in a straightforward way. Thus, for space reasons, we refer the reader interested in these algorithms to [20], where we present algorithms for all families of models.

6.1 Definitions and Notations

We start by introducing some notation. Interpretation I' for a program P extends interpretation I for P if I is consistent and $I \subseteq I'$. A model M for P extends I (or I is extended to M) if there exists an interpretation I' such that I' extends I and $M = I'^+$.

We now introduce a skeptical version, $T_P : 2^{B_P \cup \neg B_P} \rightarrow 2^{B_P}$, of the classical immediate consequence operator [22].

$$T_P(I) = \{a \in B_P \mid \exists r \in \text{ground}(P) \text{ s.t. } a \in H(r), \\ H(r) - \{a\} \subseteq \neg I, \text{ and } B(r) \subseteq I\}.$$

$\Phi_P : 2^{B_P \cup \neg B_P} \rightarrow 2^{B_P}$ denotes an extension of Fitting's operator [9] to the disjunctive case, used for computing false atoms of P .

$$\Phi_P(I) = \{a \in B_P \mid \forall r \in \text{ground}(P) \text{ with } a \in H(r) : B(r) \cap \neg I \\ \neq \emptyset \text{ or } (H(r) - \{a\}) \cap I \neq \emptyset\}.$$

Definition 6. Let \mathcal{P} be a program and I a set of literals. A positive possibly-true literal of \mathcal{P} with regard to I is an atom a such that there exists a rule $r \in \text{ground}(\mathcal{P})$ for which all the following conditions hold:

1. $a \in H(r)$;
2. $H(r) \cap I = \emptyset$ (that is, the head is not true with regard to I);

<p>Algorithm Optimal_Stable_Model</p> <p>Input: A disjunctive positive program P, a weight function \wp, a monotonic aggregation strategy \mathcal{A}.</p> <p>Output: An optimal stable model of P w.r.t. (\wp, \mathcal{A}).</p> <p>Procedure <i>Compute_Optimal</i>($I : \text{SetOfLiterals}$; $\text{var } \text{best_mod} : \text{SetOfLiterals}$);</p> <p>var I': <i>SetOfLiterals</i>; L: Atom;</p> <p>repeat $I' := I$; $I := I' \cup T_P(I') \cup \neg.\Phi_P(I')$;</p> <p>until $I = I'$;</p> <p>if $I \cap \neg.I \neq \emptyset$ or $LB_P(I, \wp, \mathcal{A}) > \text{HOF}(\wp, \mathcal{A})(\text{best_mod})$</p> <p> then return</p> <p>end_if</p> <p>if $PT_P(I) = \emptyset$ (* I^+ is a model *)</p>	<p>then if ($I^+ \subset \text{best_mod}$) or $\text{HOF}(\wp, \mathcal{A})(I^+) < \text{HOF}(\wp, \mathcal{A})(\text{best_mod})$</p> <p> then $\text{best_mod} := I^+$;</p> <p> end_if</p> <p> return</p> <p>end_if (* I^+ is a model *)</p> <p>$L := \text{choose}(PT_P(I), \wp, \mathcal{A}, P, I)$;</p> <p>$\text{Compute_Optimal}(I \cup \{L\}, \text{best_mod})$;</p> <p>$\text{Compute_Optimal}(I \cup \{\neg L\}, \text{best_mod})$;</p> <p>end_procedure</p> <p>var best_model: <i>SetOfLiterals</i>;</p> <p>begin (* Main *)</p> <p> $\text{best_model} := B_P$;</p> <p> $\text{Compute_Optimal}(\emptyset, \text{best_model})$;</p> <p> output best_model;</p> <p>end.</p>
---	---

Fig. 1. The algorithm Optimal_Stable_Model.

3. $B(r) \subseteq I$ (that is, the body is true with regard to I). A negative possibly-true literal of \mathcal{P} with regard to I is a literal $\neg a$ s.t. $a \notin (I^+ \cup I^-)$, and there exists a rule $r \in \text{ground}(\mathcal{P})$ for which all the following conditions hold:

1. $\neg a \in B(r)$;
2. $H(r) \cap I = \emptyset$ (that is, the head is not true with regard to I);
3. $B^+(r) \subseteq I$ (that is, the positive atoms in the body are true with regard to I);
4. $B^-(r) \cap I = \emptyset$ (that is, no negative literal in the body is false with regard to I).

The set of all possibly-true literals of P with regard to I is denoted by $PT_P(I)$.

Example 9. Consider the program $P = \{a \vee b \leftarrow c \& d; e \vee d \leftarrow; g \vee h \leftarrow c \& \neg f\}$ and let $I = \{c, d\}$ be an interpretation for P . Then, we have two positive possibly-true literals of P with regard to I , namely, a , and b ; and the negative possibly-true literal $\neg f$.

The following useful property of the set of possibly-true literals can be easily verified. The same property holds for the related notion of possibly-true conjunctions [21].

Proposition 3. Let P be a program and I a consistent interpretation for P . Then, $PT_P(I) = \emptyset$ if and only if I^+ is a model for P .

6.2 Computing an Optimal Stable Model of a Positive Program with a Monotonic Aggregation Strategy

Algorithm **Optimal_Stable_Model** (see Fig. 1) shows how to compute an optimal stable model of a disjunctive positive program P with a monotonic aggregation strategy \mathcal{A} .

The function $LB_P(I, \wp, \mathcal{A})$ is a polynomial-time function which returns a lower bound for the set of real numbers

$\{\text{HOF}(\wp, \mathcal{A})(M) \mid M \text{ is a model of } P, I^+ \subseteq M, \text{ and } I^- \cap M = \emptyset\}$. Thus, no model of P which extends I can get an Herbrand objective function value better than $LB_P(I, \wp, \mathcal{A})$.

For any consistent interpretation I , $\text{HOF}(\wp, \mathcal{A})(I^+)$ is a trivial lower bound for P with regard to (I, \wp, \mathcal{A}) , because \mathcal{A} is a monotonic strategy, and every model of P including I , will clearly be a superset of I^+ . Lower bounds are used to cut the search space. Consider a call to the procedure *Compute_Optimal*($I : \text{SetOfLiterals}$; **var** $\text{best_mod} : \text{SetOfLiterals}$), where I is a set of literals, and best_mod is a model for the program, representing the model with the lowest Herbrand objective function we have computed so far. Intuitively, we are looking for some optimal stable model M “extending” the set of literals I . If P has a lower bound of n with regard to (I, \wp, \mathcal{A}) , and $n > \text{HOF}(\wp, \mathcal{A})(\text{best_mod})$, it clearly does not make sense to continue this way, because we cannot compute a model better than best_mod , by extending I . Thus, we can terminate the current call to the procedure, and go back to explore other possibilities.

The actual algorithm to be used for the computation of a lower bound for some set of literals I with regard to some (\wp, \mathcal{A}) should be chosen based on the aggregation strategy \mathcal{A} , on the weight function \wp and, possibly, on the particular class of programs P belongs to.

The function $\text{choose}(PT_P(I), \wp, \mathcal{A}, P, I)$ selects a possibly-true literal from $PT_P(I)$. We can use different strategies for this selection, which best fit different combinations of weight functions and aggregation strategies. However, we only consider only polynomial-time choices. If \mathcal{A} is monotonic, the simplest strategy is a greedy choice: Possibly true atoms which are assigned the lowest weights will be chosen first. However, more sophisticated methods can be easily designed. For instance, by looking at the

program P , we can choose atoms whose immediate logical consequences give the least increment, and so on.

The following example shows, in detail, how the algorithm works.

Example 10. Let P be the following disjunctive positive program:

$$a \vee b \leftarrow \quad c \vee d \leftarrow a \quad e \vee f \leftarrow b \quad d \leftarrow .$$

We apply the algorithm *Optimal_Stable_Model* to compute a model in $\text{Opt}(P, \text{ST}(P), \wp, \text{sum})$, i.e., an optimal stable model of \overline{P} with the *sum* aggregation strategy, and a weight function \wp such that $\wp(a) = 2$, $\wp(b) = 3$, $\wp(c) = 1$, $\wp(d) = 3$, $\wp(e) = 2$, and $\wp(f) = 2$. We use the lower-bound function LB_P that, applied to an interpretation I , just returns the sum of the weights associated with the atoms in I^+ .

The procedure *Compute_Optimal* is first called with the empty interpretation, and *best_mod* is initialized with the Herbrand base of P . The value of *best_mod* is $\text{HOF}(\wp, \text{sum})(\text{best_mod}) = 13$. After the execution of the **repeat** loop, we get the interpretation $I_1 = \{d, \neg c\}$, because $T_P(\emptyset) = \{d\}$, $\Phi_P(\{d\}) = \{c\}$, and no further atoms can be obtained by using these deterministic operators. The evaluation of the lower-bound function gives $LB_P(I_1, \wp, \text{sum}) = \text{HOF}(\wp, \text{sum})(I_1^+) = 3$. This value is less than $\text{HOF}(\wp, \text{sum})(\text{best_mod})$ and, hence, the algorithm continues and computes the set of possibly-true atoms with regard to I_1 . We get $PT_P(I_1) = \{a, b\}$. Since this set is not empty, we select from it a possibly-true atom, by using the function *choose*. Assume we choose the atom a , and let $I_2 = I_1 \cup \{a\} = \{a, d, \neg c\}$.

Then, the procedure *Compute_Optimal* is recursively called with the parameters I_2 and *best_mod*, to compute possible optimal stable models starting from the interpretation I_2 . The **repeat** loop ends with the new interpretation $I_3 = \{a, d, \neg b, \neg c, \neg e, \neg f\}$. This interpretation is consistent and the lower bound $LB_P(I_3, \wp, \text{sum}) = 5$ is less than the value 13 of *best_mod*. Moreover, $PT_P(I_3) = \emptyset$. Indeed, the set $I_3^+ = \{a, d\}$ is a model of P . Since I_3^+ is a subset of $\text{best_mod} = B_P$, it becomes the new best model, that is, we set $\text{best_mod} = I_3^+$.

At this point, we come back to the previous execution of *Compute_Optimal*, and try to compute optimal models assuming the previously selected atom is false. Therefore, we call *Compute_Optimal*($I_4, \text{best_mod}$), where $I_4 = I_1 \cup \{\neg a\} = \{d, \neg a, \neg c\}$. The **repeat** loop gives the interpretation $I_5 = \{b, d, \neg a, \neg c\}$, because b is now deterministically derivable from the first rule of P . We compute the lower bound for this interpretation: $LB_P(I_5, \wp, \text{sum}) = 6$. This value is compared with the Herbrand objective function evaluated on the best model. Since $\text{HOF}(\wp, \text{sum})(\text{best_mod}) = 5$, this execution of *Compute_Optimal* is stopped because no stable model better than *best_mod* can be found starting from I_5 .

Now, the first call of *Compute_Optimal* has been completed and, thus, the algorithm stops and returns the optimal stable model $\text{best_mod} = \{a, d\}$.

For completeness, note that I_5^+ is not a model for P , and in fact there were some possibly-true atoms with regard to I_5 , as $PT_P(I_5) = \{e, f\}$. Indeed, P has other two

stable models, namely, $\{b, d, e\}$ and $\{b, d, f\}$. However, these models are not optimal because their Herbrand objective function is equal to 8, while the optimal value is 5. Therefore, thanks to the lower-bound function, the algorithm is able to cut the search space, avoiding the generation of useless (i.e., not optimal) stable models.

Theorem 9. Given a disjunctive positive program P , a weight function \wp , and a monotonic aggregation strategy \mathcal{A} , the algorithm *Optimal_Stable_Model* outputs an optimal stable model of P with regard to (\wp, \mathcal{A}) .

Proof. At each time of the computation of *Optimal_Stable_Model*, the variable *best_model* contains a model for P . Indeed, it is initialized with the Herbrand base B_P , which is clearly a model for P , and during the computation of the algorithm it can be replaced only by another model for P having a better Herbrand objective function (short: HOF) value. It is easy to verify that the algorithm terminates after a finite number of steps, because at each recursive call of *Compute_Optimal* we add to the (partial) interpretation I a new literal, and the Herbrand base of P is finite, as no function symbol occurs in P . Then, the algorithm returns a set of atoms, say M . We claim that M is a model for P . This clearly holds if $M = B_P$. Otherwise, by construction of the algorithm, M is the set of positive literals I^+ of an interpretation I such that $PT_P(I) = \emptyset$. In this case, the claim follows from Proposition 3.

Now, assume by contradiction that the algorithm *Optimal_Stable_Model* does not output an optimal stable model for P , and let M' be an optimal stable model for P . There are two possibilities:

1. the HOF value of M is strictly greater than M' , i.e., $\text{HOF}(\wp, \mathcal{A})(M) > \text{HOF}(\wp, \mathcal{A})(M')$; or
2. the models M and M' have the same HOF value, but M is not stable and hence not minimal, as P is positive.

If the second condition holds, we may assume without loss of generality that $M' \subset M$, because in this case every minimal model included in M is an optimal stable model for P . Then, it is easy to see that, for each call to *Compute_Optimal*($I, \text{best_mod}$) during the computation of *Optimal_Stable_Model*, either $\text{HOF}(\wp, \mathcal{A})(\text{best_mod}) > \text{HOF}(\wp, \mathcal{A})(M')$ or $M' \subset \text{best_mod}$ holds.

Let I be the largest interpretation for P that can be extended to M' and that has been used as a parameter for a call to the procedure *Compute_Optimal*, during the execution of *Optimal_Stable_Model*. Note that such an interpretation must exist, because the algorithm starts with the empty interpretation, which can be extended to any model. Now, consider the evaluation of *Compute_Optimal*($I, \text{best_mod}$). We first evaluate the **repeat** loop, which extends I to a new interpretation, say \bar{I} . It is easy to verify that, for every model \bar{M} for P that extends I , \bar{M} extends \bar{I} , as well. In fact, we just add to I literals that are deterministic consequences of P given the interpretation I . It follows that \bar{M}' extends \bar{I} . This in turn entails that \bar{I} is a consistent interpretation. Moreover, $LB_P(\bar{I}, \wp, \mathcal{A})$ cannot be greater than

<p>Algorithm Optimal_Stable_Model_General</p> <p>Input: A disjunctive program P, a weight function \wp, an aggregation strategy \mathcal{A}.</p> <p>Output: An optimal stable model of P w.r.t. (\wp, \mathcal{A}).</p> <p>Procedure <i>Compute_Optimal_G</i>($I : \text{SetOfLiterals}$; var $best_mod : \text{SetOfLiterals}$; var $best_val : \text{Real}$);</p> <p>var $I' : \text{SetOfLiterals}$; L: Literal;</p> <p>repeat $I' := I$; $I := I' \cup TP(I') \cup \neg \Phi_P(I')$; until $I = I'$;</p> <p>if $I \cap \neg I \neq \emptyset$ or $LB_P(I, \wp, \mathcal{A}) > best_val$ then return</p> <p>end if</p> <p>if $PT_P(I) = \emptyset$ (* I^+ is a model *) then if $\text{HOF}(\wp, \mathcal{A})(I^+) < best_val$ and $is_Stable(I^+)$ then $best_mod := I^+$; $best_val := \text{HOF}(\wp, \mathcal{A})(I^+)$;</p>	<p>end if</p> <p>return</p> <p>end if (* I^+ is a model *)</p> <p>$L := choose(PT_P(I), \wp, \mathcal{A}, P, I)$; $Compute_Optimal_G(I \cup \{L\}, best_mod, best_val)$; $Compute_Optimal_G(I \cup \{\neg L\}, best_mod, best_val)$;</p> <p>end procedure</p> <p>var $best_model : \text{SetOfLiterals}$; $best_value : \text{Real}$;</p> <p>begin (* Main *) $best_value := \infty$; $Compute_Optimal_G(\emptyset, best_model, best_value)$; if $best_value < \infty$ then output $best_model$; else output “P does not have stable models”</p> <p>end if</p> <p>end.</p>
--	--

Fig. 2. The algorithm Optimal_Stable_Model_General.

$\text{HOF}(\wp, \mathcal{A})(best_mod)$ because we know that M' is an optimal stable model and extends \bar{I} .

Thus, the computation continues and we evaluate the set of possibly-true literals $PT_P(\bar{I})$. Assume that $PT_P(\bar{I}) = \emptyset$. From Proposition 3, \bar{I}^+ is a model for P . Since \bar{I} can be extended to the model M' , then $PT_P(\bar{I}) = \emptyset$ entails $\bar{I}^+ = M'$. Thus, from the above relationships between M' and $best_mod$, it follows that M' (i.e., \bar{I}^+) should replace $best_mod$. However, if this happens, there is no way to replace M' by M during the algorithm. This is a contradiction, as we assumed M is the output of the computation of *Optimal_Stable_Model* on P .

Thus, assume $PT_P(\bar{I}) \neq \emptyset$ holds. Let L be the possibly-true literal selected by the function *choose* ($PT_P(\bar{I}), \wp, \mathcal{A}, P, \bar{I}$). We then call both

$$Compute_Optimal(\bar{I} \cup \{L\}, best_mod)$$

and

$$Compute_Optimal(\bar{I} \cup \{\neg L\}, best_mod).$$

However, if $L \in M'$, then $\bar{I} \cup \{L\}$ can be extended to M' ; otherwise, $\bar{I} \cup \{\neg L\}$ can be extended to M' . In either case, our assumption on I is contradicted. \square

Since even computing just a minimal model of a disjunctive positive program is an NP-hard task (actually, $P^{\text{NP}[O(\log n)]}$ -hard) [2], unless the polynomial hierarchy collapses, there exists no polynomial time algorithm which computes an optimal stable model for a disjunctive positive program. However, an optimal stable model can be computed by using polynomial space and single exponential time. Moreover, the algorithm runs efficiently on the

tractable class of normal positive programs. (See [20] for formal statements and proofs.)

Remark. Given a positive disjunctive program P , the algorithm *Optimal_Stable_Model* outputs a model which is also optimal with regard to minimal model semantics. To see this, observe that for positive programs, stable and minimal models coincide. Moreover, consider the family of all models. For any optimal model M , there exists a minimal model $M' \subseteq M$ which is also an optimal model. Indeed, for any monotonic aggregation strategy \mathcal{A} and any weight function \wp , $\text{HOF}(\wp, \mathcal{A})(M') = \text{HOF}(\wp, \mathcal{A})(M)$ must hold because M' is a subset of M . Thus, every optimal minimal model is also an optimal model with regard to the family of all models, hence algorithm *Optimal_Stable_Model* outputs an optimal model with regard to to the family of all models, as well.

6.3 Computing an Optimal Stable Model: General Case

Algorithm *Optimal_Stable_Model_General* (see Fig. 2) computes an optimal stable model of a general disjunctive program with any aggregation strategy. Even in the general case, an optimal stable model can be computed by using polynomial space and single exponential time. Moreover, the algorithm runs efficiently on the tractable class of normal positive (or weakly stratified) programs. (See [20] for formal statements and proofs.)

The main difference with the previous case, where the program was restricted to be positive and the aggregation strategy to be monotonic, is that we now have to explicitly check that the model at hand is stable. This is accomplished

by the function *is_Stable*. Stable model checking can be effectively done by looking for some unfounded set possibly included in the model, as suggested in [21], and recently improved in [16]. We assume *is_Stable* is implemented as described in [16].

The implementation of $LB_P(I, \varnothing, A)$ is an important issue. While the search space can be effectively reduced in the case of monotonic aggregations, there are general aggregation strategies where no “good” lower bounds can be computed in reasonable time. In this case, one can just return the trivial lower bound $-\infty$.

Moreover, for positive programs and monotonic aggregation strategies, we can always start with a model for the given program, whose HOF value represents an upper bound to the optimal HOF value. Indeed, the Herbrand base B_P is a model for any program P , and there always exists a model $M \subseteq B_P$ which is a stable model for P and has a HOF value not greater than B_P . In the general case—with negation and general aggregation strategies—this initialization would be meaningless, because logic programs with negation may have no stable models at all, and because larger models may have better HOF values than smaller models. Thus, we have added the new parameter *best_value* to the recursive procedure. *best_value* is initialized with the special value ∞ . By inspecting the value of this variable at the end of the algorithm, we can determine whether P has stable models or not. In the former case, P has also optimal stable models, and the variable *best_model* holds an optimal stable model for P .

7 RELATED WORK

To our knowledge, the idea that users (not necessarily logic programmers!) should select models that best represent their interests (as opposed to what the logic programmer thinks they should use) is novel and has never been done before. In addition, the idea of evaluating the “value” of a model drawn from a selected family of models is new. The concept of an optimal model (from an appropriate model family) is also new.

The idea of mixing logic and optimization has been studied extensively via Constraint logic programs [12]. In CLP, rules have the form $A \leftarrow \Xi \& L_1 \& \dots \& L_n$, where A is an atom, L_1, \dots, L_n are literals, and Ξ is a *constraint* over a prespecified *constraint domain*. For example, if our constraint domain is the integers, then Ξ could be a linear constraint over the integers. If our constraint domain is the reals, then Ξ could be a quadratic constraint over the reals. The semantics of CLP is a natural generalization of that of ordinary logic programs [24], [3], [29]. Constraint logic programs have been used extensively to model optimization problems [4], [35], [5]. In all of these cases, a constraint logic programmer explicitly writes constraint rules to solve the problem. The end user does not have the flexibility to specify his own objective function. Our approach is different from the constraint logic programming approach in many ways. In CLP, the objective function and the way how it should be optimized is *built-in* in the CLP program

(encoded by means of the soft constraints). Our approach, instead, clearly separates the program, the weighting function, and the optimization strategy to be adopted, providing a higher degree of freedom and more clarity from the viewpoint of knowledge modeling. Indeed, in our framework, the logic programmer writes rules in much the same way he does today. The *end-user* and not necessarily the logic programmer decides what the objective function is as well as what space of models he is interested in (the latter can also be set by the logic programmer if he wishes). Our algorithms *automatically* find an optimal model with regard to the semantics chosen by the user and the objective function chosen by the user. The logic programmer does not need to write the optimization code *in* the logic program! The way how models should be weighted, and the function to be optimized is “orthogonal” to the logic program, and it is specified at a different stage. Furthermore, existing CLPs are all written with regard to a specific domain. Many such domains have been investigated by different researchers (e.g., the real-valued domain by Jaffar and Lassez [12] as well as by many others [4], [35], [5], the 0-1 domain [1], and others). In all these frameworks, the CLP program has a set of models associated with it, but no way for the user to choose the one that he likes the most. Thus, optimal model semantics take an ordinary logic program and allow an end-user to determine what models are appropriate for the user to use, while CLP allows an expert programmer to determine how to encode combinatorial optimization problems.

There are also other logic programming frameworks where a (suitably extended) logic program has models that assign numeric values to various atoms. Quantitative logic programming and weighted logic programming methods are two such paradigms. As in the case of CLPs, both these frameworks require the programmer to specify how to assign values to atoms. We consider each of these paradigms below.

Shapiro [32] and Van Emden [36] introduced quantitative rules of the form $v : A \leftarrow B_1 \& \dots \& B_n$, where A, B_1, \dots, B_n are (positive) atoms and v is a real value in the closed interval $[0, 1]$. Intuitively, this rule says that if the truth value of $B_1 \& \dots \& B_n$ is v' , then the truth value of A is at least $v \times v'$. The truth of a conjunction is the minimum of the truth values of its individual atoms. The least Herbrand model semantics of such rules assigns a truth value to each ground atom—however, the user has no ability to compare different models from his point of view (objective function) and choose the one that makes most sense to him. Later works like that of [17], [19], [33], [15], [26], do similar things with somewhat different syntaxes.

Weighted logic programs—in two very different forms—have been studied by Marek and Truszczyński and by Niemela’s group. The first framework, proposed in [23], has the same syntax as the quantitative rules of Shapiro [32] and Van Emden [36], but allow the value associated with a rule to be any real number. The least model of such a program associates costs with each atom. Very recently, Niemela’s group has also introduced a notion of a weighted logic

TABLE 4
Database Relations for the Cooking Program in Example 1

Name	Type	Cost	Time	Person	Name of Dish
caprese	appetizer	0.55	5	nicola	malai_kofta
samosa	appetizer	0.75	25	vs	masala_dosa
spaghetti_carbonara	entree	2.75	15	francesco	masala_dosa
lasagna	entree	3.25	25	gb	caprese
malai_kofta	entree	2.25	40	simona	tiramisu
matar_paneer	entree	2.40	15	simona	idli
masala_dosa	entree	3.15	30	peppe	rasgulla
idli	entree	0.75	30	peppe	lasagna
tiramisu	dessert	3.00	30	tina	matar_paneer
rasgulla	dessert	2.00	30	tina	samosa

Guest
nicola
vs
francesco

program [27]. They have rules of the form $C_0 \leftarrow C_1, \dots, C_n$, where each C_i has the form $L \leq \{\ell_1 = w_1, \dots, \ell_m = w_m\} \leq U$, and where the ℓ_i s are literals. A weighted logic program may have several stable models. The user may specify her preferred stable models by means of statements of the form $\text{minimize}\{a_1 = w_1, \dots, a_n = w_n\}$, where a_i s are atoms and w_i are integers. Intuitively, the above statement selects the stable models M minimizing the expression $\sum_{a_i \in M(1 \leq i \leq n)} w_i$. Thus, this approach can be seen as a very particular case of our framework, where the semantics is “stable” and the aggregation function is “sum.”

8 CONCLUSIONS

Logic programs and disjunctive logic programs form a powerful reasoning paradigm that can be used to elegantly and declaratively encode a variety of problems. Given a (possibly disjunctive) logic program P , there has been an immense amount of work on characterizing the declarative semantics of P . Many of these semantics (such as the stable model semantics [10], the minimal model semantics [25], [6], and the “all-models” semantics) identify a class of models of the program as being epistemically acceptable.

In this paper, we take the point of view that a user of a logic program (which is presumably written by a logic programming professional) may wish to add criteria that she cares about, in the selection of a model from the family of models identified by the semantics. We argue that once the logic programmer encodes an application using some semantics, the user should be able to use objective functions to specify which of the models conforming to the selected semantics should be picked. We have provided a formal definition of optimal models and illustrated their utility through a simple “cooking” example, as well as a combinatorial auction example. Subsequently, we have conducted a complexity-theoretic investigation of various important problems relevant to this optimal models notion. Specifically, we have developed results on the complexity of checking whether a model is optimal (with regard to

stable model semantics, minimal model semantics, and all-models semantics), on determining whether a ground atom is true in all or one of the optimal models with regard to one of these aforementioned semantics, and on checking whether there is a model with regard to one of these semantics such that it contains certain ground atoms and is guaranteed to have a cost below a certain amount. We have also developed an exhaustive set of algorithms to compute optimal models with regard to stable, minimal, and all model semantics. Our results apply to disjunctive logic programs with negation, not just to normal logic programs.

The following questions still need to be studied. First, we need to develop efficient implementations of these algorithms, and conduct experiments in order to determine how much they scale to large programs and data sets. Second, we need to develop efficient methods to answer “brave” and “cautious” queries which ask whether a query is true in some/all optimal models. Third, we need to develop methods to optimize such queries. Preliminary starting points for these algorithms are already contained in the proofs of the complexity results, but extending them to algorithms that scale remains a challenge.

APPENDIX A1

In Table 4, we show three possible extended database relations for the predicates *dish*, *dislikes*, and *guest*, respectively, in the cooking program of Example 1.

APPENDIX A2

The logic program in Example 1 has 16 stable models altogether, depending on what is chosen as an appetizer (two choices), what is chosen as an entree (four choices), and what is chosen for dessert (two choices) (see Table 4). Note that the choices are restricted according to the given group of guests; in our example, Francesco, Nicola, and VS. We list below, for all these stable models, just the predicate *dinner*, which characterizes each menu:

$M_1 = \{dinner(caprese, lasagna, tiramisu), \dots\},$
 $M_2 = \{dinner(samosa, matar-paneer, rasgulla), \dots\},$
 $M_3 = \{dinner(caprese, spaghetti-carbonara, tiramisu), \dots\},$
 $M_4 = \{dinner(caprese, spaghetti-carbonara, rasgulla), \dots\},$
 $M_5 = \{dinner(caprese, matar-paneer, tiramisu), \dots\},$
 $M_6 = \{dinner(caprese, matar-paneer, rasgulla), \dots\},$
 $M_7 = \{dinner(caprese, lasagna, rasgulla), \dots\},$
 $M_8 = \{dinner(samosa, lasagna, rasgulla), \dots\},$
 $M_9 = \{dinner(samosa, matar-paneer, tiramisu), \dots\},$
 $M_{10} = \{dinner(samosa, lasagna, tiramisu), \dots\},$
 $M_{11} = \{dinner(samosa, spaghetti-carbonara, rasgulla), \dots\},$
 $M_{12} = \{dinner(samosa, spaghetti-carbonara, tiramisu), \dots\},$
 $M_{13} = \{dinner(caprese, idli, rasgulla), \dots\},$
 $M_{14} = \{dinner(samosa, idli, rasgulla), \dots\},$
 $M_{15} = \{dinner(caprese, idli, tiramisu), \dots\},$
 $M_{16} = \{dinner(samosa, idli, tiramisu), \dots\}.$

ACKNOWLEDGMENTS

This research was partly funded by ARO grant DAAD190010484, ARL grant DAAL0197K0135, and ARL's CTA on advanced decision architectures. It was also partly supported by the European Commission under project INFOMIX, project number IST-2001-33570, and under project ICONS, project number IST-2001-32429.

REFERENCES

- [1] P. Barth and A. Bockmayr, "Modelling Discrete Optimization Problems in Constraint Logic Programming," *Annals of Operations Research*, vol. 81, pp. 467-496, 1998.
- [2] M. Cadoli, "On the Complexity of Model Finding for Nonmonotonic Propositional Logics," *Proc. Fourth Italian Conf. Theoretical Computer Science*, M. Venturini Zilli, A. Marchetti Spaccamela, and P. Mentrasti, eds. Singapore: World Scientific, pp. 125-139, 1992.
- [3] J. Cohen, "Constraint Logic Programming," *Comm. ACM*, vol. 33, no. 7, 1990.
- [4] P. Dasgupta, P.P. Chakrabarti, A. Dey, S. Ghose, and W. Bibel, "Solving Constraint Optimization Problems from CLP-Style Specifications Using Heuristic Search Techniques," *IEEE Trans. Knowledge and Data Eng.*, vol. 14, no. 2, pp. 353-368, 2002.
- [5] M. Dincbas, H. Simonis, and P. Van Hentenryck, "Solving Large Combinatorial Optimization Problems in Logic Programming," *J. Logic Programming*, vol. 8, nos. 1-2, pp. 75-93, 1990.
- [6] J. Dix and F. Stolzenburg, "Computation of Non-Ground Disjunctive Well-Founded Semantics with Constraint Logic Programming," *Proc. Workshop Nonmonotonic Extensions of Logic Programming*, pp. 202-224, 1996.
- [7] T. Eiter and G. Gottlob, "On the Computational Cost of Disjunctive Logic Programming: Propositional Case," *Annals of Math. and Artificial Intelligence*, vol. 15, nos. 3-4, pp. 289-323, 1995.
- [8] T. Eiter, G. Gottlob, and H. Mannila, "Disjunctive Datalog," *ACM Trans. Database Systems*, vol. 22, no. 3, pp. 364-418, 1997.
- [9] M.C. Fitting, "Logic Programming on a Topological Bilattice," *Fundamenta Informatica*, vol. 11, pp. 209-218, 1988.
- [10] M. Gelfond and V. Lifschitz, "The Stable Model Semantics for Logic Programming," *Proc. Fifth Int'l Conf. Symp.*, pp. 1070-1080, 1988.
- [11] M. Gelfond and V. Lifschitz, "Classical Negation in Logic Programs and Disjunctive Databases," *New Generation Computing*, vol. 9, pp. 365-385, 1991.
- [12] J. Jaffar and J.-L. Lassez, "Constraint Logic Programming," *Proc. 14th ACM Symp. Principles of Programming Languages*, 1987.
- [13] D.S. Johnson, "A Catalog of Complexity Classes," *Handbook of Theoretical Computer Science*, J. van Leeuwen, ed., vol. A, chapter 2, B.V. (North-Holland): Elsevier Science, pp. 67-161, 1990.
- [14] J. Kadin, " $P^{(NP^{(O(\log n))})}$ and Sparse Turing-Complete Sets for NP," *J. Computer and System Sciences*, vol. 39, no. 3, pp. 282-298, 1989.
- [15] M. Kifer and V.S. Subrahmanian, "Theory of Generalized Annotated Logic Programming and Its Applications," *J. Logic Programming*, vol. 12, no. 4, pp. 335-368, 1992.
- [16] C. Koch and N. Leone, "Stable Model Checking Made Easy," *Proc. 16th Int'l Joint Conf. Artificial Intelligence*, Thomas Dean, ed. Morgan Kaufmann, pp. 70-75, Aug. 1999.
- [17] V.S. Lakshmanan and F. Sadri, "Modeling Uncertainty in Deductive Databases," *Proc. Int'l Conf. Database Expert Systems and Applications*, pp. 724-733, Sept. 1994.
- [18] V.S. Lakshmanan and F. Sadri, "Probabilistic Deductive Databases," *Proc. Int'l Logic Programming Symp.*, Nov. 1994.
- [19] V.S. Lakshmanan and N. Shiri, "A Parametric Approach with Deductive Databases with Uncertainty," *IEEE Trans. Knowledge and Data Eng.*, 1997.
- [20] N. Leone, F. Scarcello, and V.S. Subrahmanian, "Optimal Models of Disjunctive Logic Programs: Semantics, Complexity, and Computation," Univ. of Maryland Technical Report CS-TR-4298, www.cs.umd.edu/Library/TRs/CS-TR-4298.ps.Z, 2001.
- [21] N. Leone, P. Rullo, and F. Scarcello, "Disjunctive Stable Models: Unfounded Sets, Fixpoint Semantics and Computation," *Information and Computation*, vol. 135, no. 2, pp. 69-112, June 1997.
- [22] J.W. Lloyd, *Foundations of Logic Programming*. Springer-Verlag, 1987.
- [23] V.W. Marek and M. Truszczynski, "Logic Programming with Cost," unpublished manuscript, 1999.
- [24] K. Marriot and P.J. Stuckey, *Programming with Constraints: An Introduction*. MIT Press, 1998.
- [25] J. Minker, "On Indefinite Data Bases and the Closed World Assumption," *Proc. Sixth Conf. Automated Deduction*, D.W. Loveland, ed., pp. 292-308, 1982.
- [26] R. Ng and V.S. Subrahmanian, "Probabilistic Logic Programming, Information and Computation," vol. 101, no. 2, pp. 150-201, 1993.
- [27] I. Niemela, P. Simons, and T. Soinen, "Extending and Implementing the Stable Model Semantics," *Artificial Intelligence*, vol. 138, nos. 1-2, pp. 181-234, 2002.
- [28] C.H. Papadimitriou, *Computational Complexity*. Addison-Wesley, 1994.
- [29] F. Rossi, "Constraint (Logic) Programming: A Survey on Research and Applications," *New Trends in Constraints*, 1999.
- [30] D. Saccà and C. Zaniolo, "Stable Models and Non-Determinism in Logic Programs with Negation," *Proc. Ninth Symp. Principles of Database Systems*, pp. 205-217, 1990.
- [31] T. Sandholm, "Algorithm for Optimal Winner Determination in Combinatorial Auctions," *Artificial Intelligence*, vol. 135, nos. 1-2, 2002.
- [32] E. Shapiro, "Logic Programs with Uncertainties: A Tool for Implementing Expert Systems," *Proc. Int'l Joint Conf. Artificial Intelligence*, pp. 529-532, 1983.
- [33] N. Shiri, "On a Generalized Theory of Deductive Databases," PhD Dissertation, Concordia Univ., Montreal, Canada, Aug. 1997.
- [34] M.Y. Vardi, "The Complexity of Relational Query Languages (extended abstract)," *Proc. 14th ACM Symp. Theory of Computing*, pp. 137-146, 1982.
- [35] P. Van Hentenryck, "A Logic Language for Combinatorial Optimization," *Annals of Operations Research*, vol. 21, pp. 247-274, 1990.
- [36] M.H. van Emden, "Quantitative Deduction and Its Fixpoint Theory," *J. Logic Programming*, vol. 4, no. 1, pp. 37-53, 1986.



Nicola Leone received the degree in mathematics from the University of Calabria in Italy. In June 1986, he joined CRAI (an industrial consortium for computer science research and application at Rende, Italy), where he worked until December 1991. From January 1992 to September 1995, he was with the ISI Institute of the Italian National Research Council (CNR). From October 1995 to September 2000, he was a professor of database systems in the Information Systems Department at the Vienna University of Technology. Since October 2000, he has been a full professor of computer science in the Department of Mathematics at the University of Calabria, where he chairs the computer science curriculum. He has participated in several international projects, mainly on the development of advanced data and knowledge bases, including the ESPRIT projects KIWI and EDS. He is currently involved in the EU projects ICONS and OpenHeritage, and he is the project leader of EU project INFOMIX. He is the leader of the team that has built the knowledge base system DLV, the state-of-the-art implementation of disjunctive logic programming, widely used around the world. His main research interests are in the areas of knowledge representation, database theory, nonmonotonic reasoning, algorithms, and computational complexity, and deductive databases. He is the author of more than 150 papers published in conference proceedings, books, and outstanding scientific journals, including the *Journal of the ACM*, *Information and Computation*, *JCSS*, *AI Journal*, *ACM Transactions*, *IEEE Transactions*, *APAL*, *TCS*, *JLP*, and *TPLP*. He served on the program committees of several international conferences, has also been the program chair of LPNMR'99, JELIA'02, and APPIA-GULP-PRODE'03, and was guest editor of the *Artificial Intelligence Journal*. He is the President of the Steering Committee of LPNMR.



V.S. Subrahmanian received the PhD degree in computer science from Syracuse University in 1989. Since then, he has been on the faculty of the Computer Science Department at the University of Maryland, College Park, where he currently holds the rank of professor. He received the US National Science Foundation Young Investigator Award in 1993, and the Distinguished Young Scientist Award from the Maryland Academy of Science in 1997. He has

worked extensively on logic programming, nonmonotonic reasoning, and logics of uncertainty. Over the last several years, he has worked extensively on techniques to integrate multiple heterogeneous databases, data sources, and legacy applications. He also developed a theory to agentize legacy code, allowing them to collaborate with other such agentized code bases to solve complex integration and decision problems. Professor Subrahmanian has more than 120 published/accepted papers in leading journals and conferences. He has edited two books, one on nonmonotonic reasoning (MIT Press) and one on multimedia databases (Springer). He has coauthored an advanced database textbook (Morgan Kaufman, 1997), a multimedia database textbook (Morgan Kaufman, January 1998), and a monograph on heterogeneous software agents (MIT Press, June 2000). He has given invited talks at numerous national and international conferences—in addition, he has served on numerous conference and funding panels, as well as on the program committees of numerous conferences. He has also chaired several conferences. Professor Subrahmanian is or has been on the editorial board of *IEEE Transactions on Knowledge and Data Engineering*, *Artificial Intelligence Communications*, *Multimedia Tools and Applications*, *Journal of Logic Programming*, *Annals of Mathematics and Artificial Intelligence*, *Distributed and Parallel Database Journal*, and *Theory and Practice of Logic Programming*. He served on DARPA's (Defense Advanced Research Projects Agency) Executive Advisory Council on Advanced Logistics and as an ad hoc member of the Air Force Science Advisory Board (2001).



Francesco Scarcello received the PhD degree in Computer Science from the University of Calabria in 1997. He is a professor of computer science at the University of Calabria. His research interests are computational complexity, constraint satisfaction, logic programming, knowledge representation, nonmonotonic reasoning, and database theory. He has extensively published in all these areas in leading conferences and journals, including the *Journal of the ACM*, *Journal of Computer and System Sciences*, *Information and Computation*, and *Artificial Intelligence*. On the more applied side, he participated in a number of national and international projects dealing with database and knowledge representation systems. In particular, he is part of the team that built DLV, a widely used knowledge-base management system based on disjunctive logic programming. Professor Scarcello holds his current position since 2001. Before that, he was recipient of two grants from the Italian National Research Council (CNR) until 1999, when he became an assistant professor at the University of Calabria. Also, during these years, he has been visiting the Department of Information Systems at the Vienna University of Technology. Professor Scarcello serves on program committees and as a reviewer for many international conferences and journals.

► For more information on this or any other computing topic, please visit our Digital Library at <http://computer.org/publications/dlib>.