Logic programming with costs

Victor W. Marek
Mirosław Truszczyński
Department of Computer Science
University of Kentucky
Lexington, KY 40506-0027
{marek,mirek}@cs.engr.uky.edu

Fax: (606)-323-1971
Phone: (606)-257-3961

Abstract

We investigate logic programs whose rules are assigned nonnegative real numbers. These numbers are interpreted as the costs of applying rules. There are several ways in which these costs can be interpreted. In the paper, two such interpretations are discussed in detail. They are referred to as *no-reusability* and *reusability* interpretations. The former requires that an atom be paid for each time it appears in the body of a rule that fires. In the latter, once an atom is derived (and the cost of its derivation is covered), it can be used for free in the future.

We show that under the first interpretation, weighted logic programming has several useful properties. In the finite case, it is computationally tractable and there are polynomial time algorithms for computing lowest costs of deriving atoms. Moreover, several basic concepts of logic programming, including resolution and one-step provability operator, can be generalized to weighted logic programming with the no-reusability interpretation of costs. In contrast, for the second interpretation, the problem of computing lowest costs of deriving atoms from finite weighted programs under the reusability interpretation is NP-complete. In addition, logic characterizations of the lowest costs of deriving atoms and sets of atoms require more complex notions.

In the paper, we also discuss yet another interpretation of costs (as time). We show that under this interpretation weighted logic programming is equivalent to the system proposed by van Emden to incorporate confidence factors into logic programs.

1 Introduction

Logic programming is a paradigm in which declarative aspects of first-order logic are combined with corresponding provability techniques into an effective programming environment. Programs are not necessarily encodings of specific algorithms to solve a problem at hand. Rather, they consist of rules that describe relevant information. The semantics (meaning) of such programs is described by Herbrand interpretations of some first order theories. General reasoning (provability) techniques provide a computational mechanism to check whether a given atom follows from the program or not.

Computing with a logic program means finding atoms entailed by it. In this context, programs consisting of rules without negation in the body, the so called *Horn* or *definite* programs [Llo84], are especially well understood. The main issue is to decide whether an atom is a consequence of a program. For Horn programs, characterizations of collections of atoms that are implied by a program are well-known. They refer to the notion of the least Herbrand model and least fixpoint of the one-step provability operator. Furthermore, algorithmic methods, based on the resolution technique, to process queries to Horn programs are known, too.

In this paper we investigate an extension of (Horn) logic programming in which we not only want to know whether an atom can be derived from a program. We are also interested in the *cost* of derivation.

Assume that you are planning to acquire a bicycle. You have the following information:

1. A bicycle can be purchased for $108.95

2. A bicycle can be assembled by a mechanic from two kits: *kit1* and *kit2*. The mechanic charges $50.00 for labor. The first of the two kits, *kit1*, is available and costs $49.99

3. You can assemble the bicycle yourself. Your time is worth $5.45/hour (minimum wage) and it takes approximately an hour to assemble the bike once you have the following main parts: frame, back wheel and front wheel. Back wheel is put together in 2 hours from wheel frame, a tire and a brake. Front wheel requires the same parts and the same amount of time to assemble. Wheel frame costs $14.95, a tire $8.99, and a brake $6.99. The frame costs $29.99.

If we disregard the costs, we can formalize this information by a logic program as follows:

(1) *bicycle* $\leftarrow$
(2) *bicycle* $\leftarrow$ *kit1*, *kit2*
(3) *kit1* $\leftarrow$
(4) *bicycle* $\leftarrow$ *frame*, *frontWheel*, *backWheel*
(5) *backWheel* $\leftarrow$ *wheelFrame*, *tire*, *brake*
(6) *frontWheel* $\leftarrow$ *wheelFrame*, *tire*, *brake*
(7) *frame* $\leftarrow$

(8) *wheelFrame* $\leftarrow$
(9) *tire* $\leftarrow$
(10) *brake* $\leftarrow$

The program implies that a bicycle can be acquired. In fact, there are two ways to accomplish this. The first of them uses rule (1). The other one uses rules (4) - (10). Rule (2) cannot be used to acquire the bicycle as *kit2* is not available (cannot be proved from the program).

Let us now consider the costs. Clearly, the first way of acquiring the bicycle requires at least $ 108.95. The attempt to use clause (2) fails as there is no way to acquire *kit2*, no matter how much resources we have available. Finally, acquiring the bicycle by means of rules (4) - (10) costs $119.10. Indeed, each of the wheels costs $41.83 (parts - $30.93, labor - $10.90). The frame costs $29.99. The cost of labor to assemble the wheels and the frame into a bike is $5.45.

The reasoning about costs is not possible within the logic programming formalization of the problem that is given above. The costs are simply not represented. In this paper we study *weighted* logic programs in which every rule is assigned a non-negative real number. This number, called the *weight* or *cost* of a rule, is interpreted as the cost of applying this rule. In particular, we will formalize our *bicycle* example as the following weighted logic program:

(1) *bicycle* $\stackrel{108.99}{\leftarrow}$
(2) *bicycle* $\stackrel{50.00}{\leftarrow}$ *kit1*, *kit2*
(3) *kit1* $\stackrel{49.99}{\leftarrow}$
(4) *bicycle* $\stackrel{5.45}{\leftarrow}$ *frame*, *frontWheel*, *backWheel*
(5) *frontWheel* $\stackrel{10.90}{\leftarrow}$ *wheelFrame*, *tire*, *brake*
(6) *backWheel* $\stackrel{16.35}{\leftarrow}$ *wheelFrame*, *tire*, *brake*
(7) *frame* $\stackrel{29.99}{\leftarrow}$
(8) *wheelFrame* $\stackrel{14.95}{\leftarrow}$
(9) *tire* $\stackrel{8.99}{\leftarrow}$
(10) *brake* $\stackrel{6.99}{\leftarrow}$ *r*

In the paper, we will introduce the notion of derivation from a weighted program. It is a generalization of the notion of a derivation in the standard case (no costs). We will study the properties of derivations, in particular, their costs. Of main interest to us will be the problem of computing the *lowest* costs of deriving an atom or a set of atoms. We will consider both the case of finite propositional programs and the case of predicate programs.

It turns out that there are several ways to define the cost of a derivation from a collection of weighted rules and they lead to different results. Consider the following set of rules:

(a) $a \stackrel{1}{\leftarrow} c$
(b) $b \stackrel{1}{\leftarrow} c$
(c) $c \stackrel{3}{\leftarrow} .$

To compute a, one needs to compute c. Hence, the cost of computing a is 4. Assume that in addition to computing a we want to compute b. The key question is: *do we need to compute c again*? In the case when weighted rules model production processes (like in our bicycle example), another copy of c is needed in order to obtain b. Hence we need to compute it from scratch and pay for it. There are, however, other possibilities. For instance, an auxiliary result can be used in a mathematical proof as many times as necessary, without the need to re-derive it. Similarly, when describing a *design* of a complex object, each part needs to be designed once, independently of the number of times it actually appears as a component. If rules (a) - (c) model a structure of a design, there is no need to pay for the derivation of c again. Coming back to our bicycle example, if the goal is to develop a design of a bike, rules (5) - (11) can be used to this end. However, costs have to be interpreted differently. Once a wheel frame is designed for the front wheel (and we paid for the design), the same design can be used in the design of the back wheel with no need to pay for it again.

Similar distinctions can be made when considering the following two rules:

(d) $a \stackrel{1}{\leftarrow} d, d$
(e) $d \stackrel{2}{\leftarrow} .$

Assume that these rules describe how an item can be obtained from other items. Under the "design interpretation", to generate a design for a, the design of d needs only to be described once. In such case, rule (d) is equivalent to

(d′) $a \stackrel{1}{\leftarrow} d.$

The situation is different under the "production interpretation". Rule (d) states that two different copies of an item d are needed to produce item a while rule (d′) states that just one copy of d is needed to yield a. Consequently, rules (d) and (d′) are not equivalent when "production" interpretation is assumed. Let us note that in logic programming, rules like (d) are usually considered as poorly encoded but equivalent versions of (d′).

This discussion points to two approaches to computing costs of deriving elements (atoms). One is based on the *no-reusability* assumption, the other one on the *reusability* assumption. In the first of them, in order to apply a rule, one has to pay for the derivation of all the atoms in the body of the rule as well as for the application of the rule. In the second model, one has to pay for the application of the rule and for the derivation of those atoms in the body of the rule that have not yet been computed. Those that have been established earlier can be used for free. There are evident similarities to the concept of

using lemmas in mathematical arguments and to tabling in logic programming (facts once derived are available to use in future derivations). The difference is that in mathematical reasoning or in deriving facts from logic programs, there is no cost function to minimize. In contrast, in our case, we will be looking for derivations of minimum possible cost. Thus, while tabling improves efficiency of computation in logic programming [RRS$^+$95], here the complexity of optimizing with respect to the reusability cost i increases (see Section 3).

In this paper, we will formally define and study these two measures of the cost of a derivation. We will show that the no-reusability measure has a number of useful properties. First, the problem of computing the minimum cost of deriving a set of atoms can be reduced (in polynomial time) to that of computing the minimum cost of deriving a single atom. Second, there exists an embedding of weighted logic programming into constraint logic programming. Third, in the case of finite propositional weighted programs, the minimum cost of a derivation can be computed in polynomial time by a version of Dijkstra shortest path algorithm. Finally, in the case of predicate programs we have natural counterparts to the notions such as Herbrand base, model, one-step provability operator and the resolution procedure. These notions provide characterizations of minimum costs of deriving atoms.

The reusability measure behaves in a drastically different way. The problem of computing the minimum cost of deriving a set of atoms cannot be reduced to that of computing the minimum cost of deriving a single atom. No constraint logic programming interpretation of weighted logic programs capturing the reusability interpretation is known so far. Further, in the finite propositional case, computing lowest costs is related to the minimum Steiner tree problem rather than to the shortest path problem and, thus, is NP-hard. Finally, while generalizations of the notion of Herbrand base, one-step provability operator and resolution procedure can be formulated in the case of the reusability approach, they are substantially more complex than in the no-reusability case. For instance, a generalization of the Herbrand base to the case of reusability consists of finite *sets* of ground atoms, and not just ground atoms.

There are several other ways of measuring costs of derivations. The weights of rules can also be interpreted as time. Then, under our "production" interpretation, the cost of deriving an atom is equal to the total time needed to "produce" an atom, assuming that no two tasks can be done in parallel. But what if we allow parallelism? Let us assume that we have enough resources (people, machines, processors, etc.) to execute a weighted rule

(f) $d \stackrel{2}{\leftarrow} a, b$

in parallel. That is, our resources allow us to "produce" a and b concurrently. Consider the program consisting of rules (a) - (c) and (f). The time needed to produce d is 6. Indeed, a takes 4 time units and b takes 4 time units. Both can be computed in parallel. Then, 2 time units are needed to assemble a and b into d. Note that under the no-reusability interpretation, the cost of producing d is 10 (the cost of producing a and b

plus the cost of applying rule (f)) and under the reusability interpretation, the cost of producing d is 7. Hence, all three interpretations are different.

Interestingly, the "parallel" measure of costs has all the nice properties of the no-reusability measure. In particular, computing the lowest cost of deriving an atom from a finite propositional program can be done in polynomial time, appropriate resolution procedure can be defined and an embedding into constraint logic programming exists. Moreover, these results can be proved by the same techniques that are used in the proofs of the corresponding results for the no-reusability measure.

Logic programs with weights were first studied by van Emden [vE86]. The syntax of van Emden's programs is almost identical to that of weighted logic programs. The difference is that van Emden requires that the label assigned to a rule be from the interval $(0, 1]$. It is motivated by the intention of van Emden to represent and compute *confidence factors* (his paper was written in the heyday of expert systems). It turns out that weighted logic programming with the parallel measure of costs is isomorphic to the van Emden's approach. Consequently, all the nice computational properties of the no-reusability and parallel measures of costs hold for the van Emden's system, too.

Existence of almost identical results for all three systems — weighted logic programming with no-reusability and parallel cost measures, and the van Emden system — suggest that there is a more general approach to weighted logic programming that would allow us to treat all these three interpretations as special cases. Finding this common generalization remains an open problem.

The paper is organized as follows. In Section 2, we introduce weighted logic programs and two interpretations of rule weights and costs — one based on the no-reusability assumption, the other one based on the reusability assumption. We also state and prove several basic properties of weighted logic programs. In the following section, we study the case of finite propositional weighted programs. We establish the complexity of computing minimum costs of derivations and present polynomial time algorithms to compute these minimum costs (for the no-reusability approach). In Section 4, we deal with predicate programs with, possibly, infinite Herbrand bases. In the case of no-reusability measures, we introduce generalizations to the notions of Herbrand base, model, one-step provability operator and resolution procedure. We use these concepts to characterize minimum costs of deriving atoms and sets of atoms. We briefly discuss a corresponding approach for the reusability interpretation. As mentioned, it is much more complex and does not yield elegant computational methods.

In Section 5, we discuss yet another interpretation of costs. In this approach, rule weights are interpreted as time and we assume that premises of a rule can be computed "in parallel". Finally, in Section 6, we discuss the approach of van Emden [vE86]. We show that van Emden's system of logic programming for confidence factors is isomorphic to our weighted logic programming under the the "parallel" measure of cost. in particular, there is a 1-to-1 transformation between the programs in both systems such that the corresponding intended models are preserved. As a consequence, we get results on complexity of computing confidence factors using the system of van Emden (in the case of finite propositional programs) and a resolution technique (for the general case).

In Section 7 we discuss the connection between the programs with costs and the general formalism for annotated logic programs proposed by Kifer and Subrahmanian in [KS92]. Due to the generality of the approach of [KS92], no overall complexity results are known for annotated logic programs. Our results show that for particular algebras (semilattices with additional operations) complexity results can be obtained. In particular, the representation of three (out of four considered below) classes of programs as annotated programs translates into complexity results for some classes of such programs. To the best of our knowledge these are first specific complexity results in this area.

The last section contains conclusions and open problems.

2 Preliminaries

We will now formally introduce weighted logic programs, the notions of derivation tree and forest, and the two models of measuring the costs of derivations.

Let $\mathcal{L}$ be some first-order language and let At be its set of atoms. A *weighted* logic program rule is an expression of the form

$$C = a \stackrel{s}{\leftarrow} b_1, \ldots, b_n \tag{1}$$

where s is a nonnegative real number and $a, b_1, \ldots, b_n$ are atoms of $\mathcal{L}$. As usual, a is called the *head* of C, the atoms $b_1, \ldots, b_n$ are said to form the *body* of C. The real number s is called the *cost of applying rule* C (or, simply, the *cost* of C). Given a weighted rule C, we will denote its cost by $wt(C)$.

A *weighted logic program* is a set P of weighted rules. Most of our results are concerned with the case of finite predicate and propositional programs. Given a weighted logic program P, by $u(P)$ we will denote the underlying logic program, that is, the collection of rules from P with their weights stripped. By the *Herbrand base* of a weighted program P, denoted by $\mathcal{H}(P)$, we mean the Herbrand base of the program $u(P)$. Similarly, by *ground*(P) we mean the set of ground instances of weighted clauses from P. Note that weights do not contain variables and, thus, do not change during the instantiation.

The basic intuition behind weighted rules and programs is that they are considered in the context of some resources necessary in order for weighted rules to be used. Specifically, the intended meaning of a weighted rule (1) is: *if* $b_1, \ldots, b_n$ *have been established and the amount of available resources equals or exceeds* s*, then we can derive* a *and, to do so, we decrease the amount of available resources by* s. Let us consider a rule

$$a \stackrel{s}{\leftarrow} .$$

It allows us to derive a *but only if we have enough resources* (at least s).

The central question we study in this paper is: what is the cost of deriving a ground atom or a set of ground atoms by means of a weighted logic program P. To study this question, we need the notion of a derivation. Let P be a logic program (possibly weighted; weights of rules are immaterial in this definition). A *derivation tree* from P is a rooted finite tree ε such that

1. each vertex of ε is labeled with a rule of the form $r\Theta$, where $r \in P$ and Θ is a ground substitution

2. if vertex v is labeled with a rule $a \leftarrow b_1, \ldots, b_k$, then v has exactly k children $v_1, \ldots, v_k$ in ε, and for each i, $1 \leq i \leq k$, v_i is labeled with a rule whose head is b_i.

Clearly, the nodes labeled with rules that have empty bodies are leaves of derivation trees. A *derivation forest* is a finite collection of derivation trees. (Note that in logic programming derivations are usually defined as sequences of atoms. Our definition of a derivation tree constitutes an equivalent alternative.)

Let ε be a derivation tree (forest). By $R(\varepsilon)$ we denote the set of all rules used to label the nodes of ε. If ε is a derivation tree, its root will be denoted by $root(\varepsilon)$.

An important question is: what is implied by a derivation tree or forest? Commonly, a derivation (proof) is said to provide a justification for the head of the last rule of a derivation. However, it is clear that, at least in the standard setting, a derivation in the same time implies the heads of all its rules. Interestingly, in the case of weighted logic programming, the concept of a set of atoms *yielded* or *implied* by derivation trees and forests depends on the model of costs. Indeed, let us look again at the two rules (a) and (c). Consider a derivation shown in Figure 1. Under the no-reusability model (it is convenient to resort to our "production" intuition here), this derivation yields a only. It is so because c, available after rule (c) is applied, is then used up while producing a during the application of rule (a).

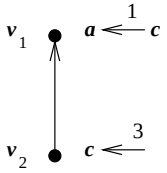

Figure 1: Derivation tree using rules (c) and (a)

Under the reusability model ("design" analogy), once c is obtained through an application of rule (c), it is used to "activate" rule (a), but remains available for other applications in the future. Hence, under the reusability model, both a and c are implied by the derivation tree in Figure 1. To derive $\{a, c\}$ under the no-reusability model, a derivation forest is needed (see Figure 2).

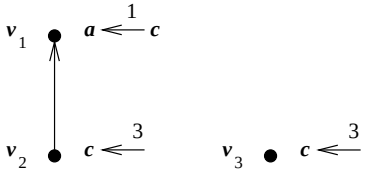

Figure 2: Derivation forest implying $\{a, c\}$ under the no-reusability assumption

To formalize this discussion, we will now introduce two relations $\vdash_{reu}$ and $\vdash_{nr}$. Let ε be a derivation forest (or tree) and let A be a set of atoms. Define

$$\varepsilon \vdash_{reu} A$$

if A is contained in the set of the heads of all rules used as labels by ε. Define

$$\varepsilon \vdash_{nr} A$$

if A is contained in the set of heads of rules used to label the roots of the trees in ε.

Clearly, the relations $\vdash_{reu}$ and $\vdash_{nr}$ capture the concept of a derivation forest (tree) implying a set of atoms for the two cost models considered. Note that if ε is a tree, under the no-reusability model it implies exactly one atom — the head of the rule used as the label of the root.

The following property of derivation forests can be easily proved using standard logic programming techniques (observe that the notion of a weight is immaterial in the context of derivation trees and forests).

Proposition 2.1 *Let P be a weighted logic program and let A be a finite subset of the least model of $u(P)$. Then*

1. *there is a derivation forest ε such that $\varepsilon \vdash_{reu} A$*
2. *there is a derivation forest ε such that $\varepsilon \vdash_{nr} A$.*

We will now introduce the notion of the cost of derivation trees and forests for both cost models. Let ε be a derivation tree. We will first assume the no-reusability interpretation of costs. That is, each time an atom serves as a premise to a rule in a derivation, we have to pay for this atom's derivation. The resulting notion of the cost of a derivation tree will be referred to as the *no-reusability cost*.

We define the no-reusability cost of a vertex v in ε by induction. Assume that v is labeled by a rule r and that the costs of all children $v_1, \ldots, v_k$ of a vertex v have been already computed. Then, the cost of v is defined by

$$cst^{nr}(v) = wt(r) + \sum_{i=1}^{k} cst^{nr}(v_i).$$

The no-reusability cost of the derivation tree ε, $cst^{nr}(\varepsilon)$, is now defined as the cost of the root of ε. For a derivation forest ε, its no-reusability cost $cst^{nr}(\varepsilon)$ is defined as the sum of the costs of all its trees. It is easy to see that the cost of a derivation forest under the no-reusability interpretation can be equivalently defined as the sum of weights of rules used to label the nodes of the derivation forest, counting the cost of each rule as many times as it appears as a label.

Our second definition of cost assumes the *reusability* model. That is, once an atom is derived, we can use it for free whenever it appears as a premise of a rule in a derivation.

The resulting notion of cost will be referred to as *reusability cost*. For a derivation forest ε, the reusability cost $cst^{reu}(\varepsilon)$ is defined by

$$cst^{reu}(\varepsilon) = \sum_{r \in R(\varepsilon)} wt(r).$$

We will show later that, under the no-reusability assumption, there are close connections between weighted logic programming and standard logic programming (and, especially, constraint logic programming). This is in sharp contrast with the reusability approach, for which only much weaker analogies are established.

To illustrate the two concepts of the cost, consider program P consisting of rules (a), (b) and (c) described in the introduction, and of rules

(g) $d \overset{3}{\leftarrow} a, b$
(h) $e \overset{3}{\leftarrow} b$.

Figure 3 shows a derivation forest, say ε, from this program. Under the reusability assumption, ε implies the set of atoms $\{a, b, c, d, e\}$ (and each of its subsets). The cost $cst^{reu}(\varepsilon)$ is 11. On the other hand, under the no-reusability assumption, ε implies $\{d, e\}$ (and its subsets) and the cost $cst^{nr}(\varepsilon)$ is 18.

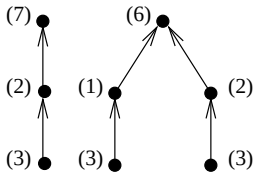

Figure 3: An example of a derivation forest

Derivation forests have a natural representation as graphs. Let ε be a derivation forest. The corresponding derivation graph is obtained from ε by collapsing all vertices labeled by the same rule. Figure 4 shows the derivation graph for the derivation forest shown in Figure 3.

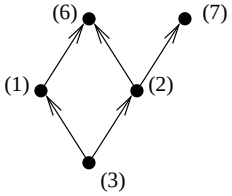

Figure 4: The derivation graph of the derivation forest from Figure 3

It is easy to see that if all rules in a program have weight 1 then the number of nodes in a derivation forest (respectively, derivation graph) coincides with the no-reusability (respectively, reusability) cost of this derivation.

Let A be a set of atoms. Define

$$cst^{nr}(A) = \inf\{cst^{nr}(\varepsilon) : \varepsilon \text{ is a derivation forest such that } \varepsilon \vdash_{nr} A\}$$

and

$$cst^{reu}(A) = \inf\{cst^{reu}(\varepsilon) : \varepsilon \text{ is a derivation forest such that } \varepsilon \vdash_{reu} A\}.$$

Observe that if there is no derivation forest that proves A (for instance, when A is infinite) then the corresponding cost ($cst^{nr}(A)$ or $cst^{reu}(A)$) is equal to infinity.

In this paper, for each of these two cost measures, we will study the following three questions:

single-atom-derivation: Given a weighted logic program P, an atom x, and an integer k, decide whether there is a derivation tree of x from P with cost no more than k?

set-of-atoms-derivation: Given a weighted logic program P, a set of atoms X, and an integer k, decide whether there is a derivation forest of X from P with cost no more than k?

least-model-derivation: Given a weighted logic program P and an integer k, decide whether there is a derivation forest of the least model of $u(P)$ with cost no more than k? (This is a special case of set-of-atoms-derivation problem, when X is the least model of P.)

We conclude this section with several simple properties of the notions introduced earlier.

Proposition 2.2 *Let P be a weighted logic program and let A be a subset of the least model of $u(P)$.*

1. *If A is finite, $cst^{nr}(A)$ and $cst^{reu}(A)$ are finite.*
2. *If both P and A are finite, then the infima $cst^{nr}(A)$ and $cst^{reu}(A)$ are attained.*

Proof: (1) By Proposition 2.1, there is a derivation forest ε such that $\varepsilon \vdash_{nr} A$. This derivation forest has a finite cost $cst^{nr}(\varepsilon)$. Moreover, the cost $cst^{nr}(\alpha)$ of every derivation forest α is non-negative. Hence, it follows that $cst^{nr}(A)$ is finite. The argument in the case of the reusability model is the same.
(2) Let $P = \{r_1, \ldots, r_m\}$. Consider a derivation forest ε such that $\varepsilon \vdash_{nr} A$ (such derivations exist by Proposition 2.1). Since the cost $cst^{nr}(\alpha)$ of any derivation forest α is of the form

$$n_1 s_1 + \ldots + n_m s_m,$$

where n_i are non-negative integers and s_i are the costs of applying rule r_i, there are only finitely many different reals in the interval $[0, cst^{nr}(\varepsilon)]$ that can serve as the cost of a derivation that implies A. Hence, the claim, for the no-reusability case, follows. The reasoning for the reusability case is the same and is omitted. □

The assertion of Proposition 2.2(2) fails if infinite programs are allowed. Consider a program $P = \{r_1, r_2, \ldots, \}$, where $r_k = \quad p \overset{1/k}{\leftarrow}$. Clearly, $cst^{nr}(p) = cst^{reu}(p) = 0$ but there is no derivation forest of p of cost 0.

Our last result in this section deals only with the no-reusability case and shows that function $cst^{nr}(\cdot)$ is *additive*. There is no corresponding result for the reusability measure.

Proposition 2.3 *Let P be a weighted program and let $a_1, \ldots, a_k$ be atoms. Then,*

$$cst^{nr}(\{a_1, \ldots, a_k\}) = \sum_{i=1}^{k} cst^{nr}(a_i).$$

Proof: If at least one atom a_i is not derivable from $u(P)$, then there is no derivation tree of a_i and there is no derivation forest of $\{a_1, \ldots, a_k\}$. Hence, both sides of the identity are equal to infinity.

Therefore, from now on we will assume that all atoms a_i are derivable from $u(P)$. Let ε be a derivation forest of $\{a_1, \ldots, a_k\}$. For each a_i, this derivation forest contains a derivation tree ε_i that implies a_i. Hence,

$$cst^{nr}(\varepsilon) \geq \sum_{i=1}^{k} cst^{nr}(\varepsilon_i) \geq \sum_{i=1}^{k} cst^{nr}(a_i).$$

Consequently,

$$cst^{nr}(\{a_1, \ldots, a_k\}) \geq \sum_{i=1}^{k} cst^{nr}(a_i).$$

Conversely, let ε_i be a derivation of a_i. Let ε be a derivation forest consisting of all trees ε_i. Then, $\varepsilon \vdash_{nr} \{a_1, \ldots, a_k\}$ and

$$\sum_{i=1}^{k} cst^{nr}(\varepsilon_i) = cst^{nr}(\varepsilon) \geq cst^{nr}(\{a_1, \ldots, a_k\}).$$

Consequently,

$$\sum_{i=1}^{k} cst^{nr}(a_i) \geq cst^{nr}(\{a_1, \ldots, a_k\}).$$

Hence, the equality follows. □

Remark 2.1 Proposition 2.3 shows that under the no-reusability interpretation of the cost, problems set-of-atoms-derivation and least-model-derivation can be reduced (in polynomial time) to the single-atom-derivation problem.

3 Finite propositional weighted logic programs

In this section we will investigate computational issues related to *finite propositional weighted logic programs*. We will start our discussion of the three problems listed in Section 2, that is single-atom-derivation, set-of-atoms-derivation and least-model-derivation, in the case of the no-reusability approach.

First, we will study the case of propositional weighted bi-horn programs. A *bi-horn rule* is a definite (Horn) rule with at most one atom in the body. A *weighted bi-horn* program is a weighted logic program that consists only of (weighted) bi-horn rules. We will associate with each propositional weighted bi-horn program P a weighted directed graph $G(P)$. All atoms appearing in P, together with one new element st, are the vertices of the graph $G(P)$. There is a directed edge from x to y if one of the following two cases holds:

1. $x = st$ and $y = a$ for some rule $a \leftarrow \in P$
2. $x = b$ and $y = a$ for some rule $a \leftarrow b \in P$.

The edge of $G(P)$ corresponding to a rule r will be denoted by e_r. Similarly, the rule that gives rise to an edge e of $G(P)$ will be denoted by r_e.

The weight of the rule $r_e \in P$ that defines an edge e in $G(P)$ is assigned to e as the weight of e. We will denote it by $wt(e)$. For a subgraph H of $G(P)$, the sum of the weights of all edges in H will be called the weight of H and will be denoted by $wt(H)$.

We have the following result establishing a correspondence between derivation trees and directed paths in $G(P)$ that start in st. Throughout the paper, by a path we mean a *simple* path, that is, repetitions of vertices are not allowed.

Proposition 3.1 *Let P be a weighted bi-horn program and let a be an atom appearing in P.*

1. *Let ε be a derivation tree of a from P. There is a directed path W in $G(P)$ starting in st and terminating in a. Moreover, the length of W is not greater than the cost of ε (for both measures of cost).*
2. *Conversely, let W be a directed path in $G(P)$ starting in st and terminating in a. The rules corresponding to the edges of W determine a derivation tree ε of a from P in which no rule is used more than once as a label and with the same cost (under both cost measures) as the length of W.*

Proof: (1) Since every rule in a bi-horn program has at most one atom in the body, the underlying tree $T(\varepsilon)$ of ε is a path. For every atom v that appears as the head of a rule in ε, let us denote by $r(v)$ the lowest rule in ε with the head v. Next, define

$$R'(\varepsilon) = \{r(v) \colon v \text{ is the head of a rule in } R(\varepsilon)\}.$$

Let $r_1, \ldots, r_m$ be an enumeration of $R'(\varepsilon)$ according to the order in which the rules appear in ε. Let k be the integer such that $r_k = r(a)$.

Clearly, the body of r_1 is empty. Hence, the corresponding edge, e_{r_1}, starts in st. Moreover, for every i, $1 \leq i \leq k-1$, the head of r_i is the only premise of r_{i+1}. Consequently, the head of the edge e_{r_i} is the tail of $e_{r_{i+1}}$. Thus, the edges corresponding to the rules $\{r_1, \ldots, r_k\}$ form a path starting in st. Since the last of these edges corresponds to the rule with a as the head, the path ends in a.

Due to the direct correspondence between the costs of rules and edges, the second part of the assertion of (1) follows. Part (2) can be argued similarly. □

By Proposition 3.1, it follows that for weighted bi-horn programs, the single-atom-derivation problem, for each of the two cost measures, is equivalent to the shortest-path problem:

> Given a directed weighted graph G, a vertex st of G, a vertex x of G and an integer k, decide whether there is a directed path in G between st and x and such that the sum of its edges is at most k.

Indeed, by Proposition 3.1, there is a derivation tree ε of x from a bi-horn program P such that $cst^{reu}(\varepsilon) \leq k$ (or, equivalently, $cst^{nr}(\varepsilon) \leq k$) if and only if there is a path from st to x in $G(P)$ with length no more than k. Hence, we get the following result.

Theorem 3.2 *For the class of weighted bi-horn logic programs and for each cost measure cst^{nr} and cst^{reu} the single-atom-derivation problem is in P and can be solved by Dijkstra shortest path algorithm.*

We will now show that, under the no-reusability interpretation, this result extends to the case of arbitrary weighted logic programs. Furthermore, as observed in Remark 2.1, problems set-of-atoms-derivation and all-atoms-derivation can be reduced to single-atom-derivation. Hence, all of them are in class P.

Theorem 3.3 *Under the no-reusability interpretation, problems single-atom-derivation, set-of-atoms-derivation and all–atoms-derivation are all in* P*.*

Proof: Let P be a finite propositional weighted logic program. Throughout the algorithm, we will maintain two sets of atoms $A \subseteq C \subseteq At(P)$. After every iteration of the algorithm, A will consist of those atoms for which the lowest cost of deriving them has already been found. For an atom $a \in A$, this cost will be denoted by $d(a)$. Similarly, C will consist of those atoms that are not in A but can be proved from the atoms in A by applying one rule from P. For each atom $c \in C$, $d(c)$ denotes the lowest cost of a derivation tree ε of c such that the heads of the rules from $R(\varepsilon) \setminus root(\varepsilon)$ are in A.

In the remainder of the proof, we will use C and A to denote the corresponding sets before an iteration and C' and A' to denote the updated versions of these sets after the iteration.

We start with $A = \emptyset$, $C = \{a\colon a \stackrel{w}{\leftarrow} \in P\}$, $d(a) = \min\{w\colon a \stackrel{w}{\leftarrow} \in P\}$ and $d(a) = \infty$ for every atom a appearing in P, $a \notin C$. In each iteration, we select an element from $c \in C$

with the lowest value of $d(c)$. We include it in A and update the sets A and C. That is, we remove c from C and add it to A (in this way we obtain sets that we denote C' and A'). Including c to A may increase the set of the rules with bodies included in the set A'. Consequently, two types of updates may become necessary:

1. Some new atoms may have to be added to C. It is the case for every atom a such that $a \notin C$ and there is a rule $r \in P$ such that $body(r) \subseteq A'$. For each such atom a, $d(a)$ is computed by the following formula:

$$d(a) = \sum_{b \in body(r)} d(b) + wt(r).$$

2. For some atoms $a \in C'$, the value $d(a)$ may have to be modified (lowered). It is the case for every atom $a \in C'$ such that

$$d(a) > \sum_{b \in body(r)} d(b) + wt(r).$$

   In such case, $d(a)$ is replaced by $\sum_{b \in body(r)} d(b) + wt(r)$.

In each iteration, these updates are executed by inspecting all rules with c in the body and with all other atoms from the body in A (that is, rules activated by moving c from C to A). For each of these rules, depending on whether its head is in the current version of C or not, either update of type (2) or of type (1) is executed. We continue iterating as long as C is not empty.

We will now prove the correctness of this algorithm. To this end, we will prove, by induction on the number of iterations, that after every iteration the following conditions hold:

1. For every $a \in A$, $d(a)$ is the minimum cost of a derivation tree of a from P
2. C consists of those atoms that are not in A but can be proved in one step from A
3. For every $a \in C$, $d(c)$ is the lowest cost of a derivation tree of c whose all rules but one (labeling the root) have heads in A.

Clearly, all conditions hold before we start the iterations. Let us assume that all these conditions hold after k iterations. We will show that they hold after $k+1$ iterations (assuming $C \neq \emptyset$ after the iteration k).

Let c be the vertex selected in the iteration $k+1$ from C. By the induction hypothesis, there is a derivation tree of c with cost $d(c)$. Assume that $d(c)$ is not the minimum cost of deriving c. Let ε be a minimum cost derivation tree of c. Let δ be a subderivation tree of ε (a derivation tree based on a subtree of $T(\varepsilon)$ and using the same labels for its nodes as ε) such that all its rules but the one used to label the root have heads in A. Since $c \notin A$, such a derivation tree δ exists.

Denote by c' the head of the rule labeling the root of $T(\delta)$. By the induction hypothesis, $c' \in C$. Moreover, $d(c') \leq cst^{nr}(\delta) \leq cst^{nr}(\varepsilon) < d(c)$. Consequently, c' should have been selected in the iteration $k+1$ rather that c — a contradiction. Hence, condition (1) holds after iteration $k+1$. Conditions (2) and (3) are also satisfied due to the update process, described earlier, that is applied in each iteration. □

The algorithm described in the proof of Theorem 3.3 is based on the same ideas as the algorithm by Dijkstra to compute shortest paths in directed graphs. Consequently, it can be implemented in the same way, with a Fibonacci heap used to represent the set C. This yields an implementation of the algorithm to compute a cost of deriving a single atom (under the no-reusability interpretation of costs) that runs in time $O(m + n \log n)$, where m is the size of the program and n is the number of atoms appearing in the program.

Let us consider now the reusability measure cst^{reu}. Again, let us start with the case of bi-horn programs. In this case, the set-of-atoms-derivation problem is equivalent to directed minimum Steiner tree (DMST) problem:

> Given a directed weighted graph G, a vertex st of G, a subset X of the vertex set of G and an integer k, decide whether there is a directed subtree of G rooted in st, covering all vertices in X, and such that the sum of its edges is at most k.

Indeed, using the reasoning of the proof of Proposition 3.1, one can show that there is a derivation forest ε for a set of atoms A and such that $cst^{reu}(\varepsilon) \leq k$, if and only if there is a tree in $G(P)$ rooted in st, covering all vertices in A and with total weight at most k.

In the special case, when A consists of all atoms derivable from P, deciding whether A has a derivation forest of cost at most k (all-atoms-derivation problem) becomes equivalent to the directed minimum cost spanning tree (DMCST) problem:

> Given a directed weighted graph G, a vertex st of G, and an integer k, decide whether there is a directed subtree of G rooted in st, covering all vertices reachable from st, and such that the sum of its edges is at most k.

These observations, together with some well-known results on DMST problem and DMCST problem, imply the following theorem.

Theorem 3.4 *For the class of weighted bi-horn logic programs and the reusability measure cst^{reu} we have:*

1. *Set-of-atoms-derivation problem is NP-complete*
2. *All–atoms-derivations is in P.*

Proof: Part (2) follows from the well-known results for the DMCST problem [Tar77, GGST86]. We will now prove part (1) by showing that DMST problem is NP-complete.

It is clear that the problem is in NP. To prove NP-hardness, we will use the fact that the undirected version of Minimum Steiner Tree problem is known to be NP-complete [GJ79]. Let us recall that the Minimum Steiner Tree (MST) problem is defined as follows:

> Given an undirected weighted graph G, a subset X of the vertex set of G and an integer k, decide whether there is a subtree of G covering all vertices in X and such that the sum of its edges is at most k.

Let G be an undirected weighted graph with the vertex set V. Let G^* be a directed graph with V as its vertex set and obtained from G by replacing each undirected edge $\{x, y\}$ in G by two directed edges (x, y) and (y, x). To each edge (x, y) of G^* we assign the weight of the edge $\{x, y\}$ in G.

Let X be a subset of V and let k be an integer. We have that G, $X \subseteq V$ and an integer k is a YES instance of MST problem if and only if there is a vertex $st \in V$ such that G^*, st, X and k is a YES instance of the DMST problem. Indeed, assume that T is a subtree of G covering all vertices in X and such that the sum of all edges of T is at most k. Select any vertex in T, say st, and form a directed tree T^* by ordering all edges in T "away" from st. Clearly, T^* is a directed subtree of G^*, covering X, rooted in st and with the same total cost as T. Conversely, if T^* is a directed subtree of G^*, covering X, rooted in st and with the sum of its edges no more than k, then the tree T obtained from T^* by dropping the ordering of the edges is a subtree of G. Moreover, it covers X and has the same cost as T^*.

It follows that if there existed a polynomial time algorithm to solve DMST problem then, calling it $|V|$ times, once for each vertex of V as a potential root, would constitute a polynomial time method for solving MST. Since MST is NP-hard, NP-hardness of DMST follows. □

We will next study the case of arbitrary propositional weighted programs and the reusability measure cst^{reu}. It turns out that all three problems are, under these assumptions, NP-complete.

Theorem 3.5 *Under the reusability model problems single-atom-derivation, set-of-atoms-derivation and all-atoms-derivation are NP-complete.*

Proof. All problems are clearly in NP. NP-hardness of the set-of-atoms-derivation problem follows from Theorem 3.4. We will now show NP-hardness of single-atom-derivation problem.

Let P be a propositional weighted logic program, let X be a set of atoms and let k be an integer (that is, P, X and k form an input for the set-of-atoms-derivation problem). Assume that $X = \{a_1, \ldots, a_k\}$. Let P' be a program obtained by adding to P a rule

$$z \stackrel{0}{\leftarrow} a_1, \ldots, a_k,$$

where z is a new atom not appearing in P. Clearly, z has a derivation tree from P' with cost no more than k if and only if X has a derivation forest from P with cost no

more than k. Since the set-of-atoms-derivation problem is NP-hard, NP-hardness of the single-atom-derivation problem follows.

Finally, consider a propositional weighted logic program P, an atom x and and integer k (an input to the single-atom-derivation problem). Define

$$P' = P \cup \{a \stackrel{0}{\leftarrow} x : a \in \mathcal{H}(P),\ a \neq x\}.$$

Clearly, if x is not derivable from P (this can be checked in linear time in the size of P [DG84]) then there is no derivation tree of x from P with cost at most k. Otherwise (x is derivable from P) we have that there is a derivation tree of x with cost at most k if and only if the least model of P' has a derivation from P' with cost at most k. Since the single-atom-derivation is NP-hard, the NP-hardness of the all-atoms-derivation follows. □

The results of this section point to a fundamental difference between the reusability measure cst^{reu} and the no-reusability measure cst^{nr}. In the first case, computing minimum costs of derivations of atoms and sets of atoms is NP-hard, in the second one, it can be accomplished in polynomial time.

4 Computing costs for predicate programs

Several concepts of standard logic programming, such as Herbrand base, model, one-step provability operator and a resolution procedure, have natural extensions to the case of weighted logic programs. The associated notion of a cost of an atom turns out to coincide with the no-reusability cost introduced in the previous section. We will also show that under the no-reusability interpretation, weighted logic programming can be embedded into constraint logic programming. This seems to be the underlying reason for all the elegant properties of no-reusability measure.

We will now introduce these generalizations, and formally establish connections to the no-reusability interpretation of weighted logic programs. We will also describe the connection to constraint logic programming.

At the end of this section, we will briefly discuss possible extensions of the concepts of Herbrand base, model, one-step provability operator and resolution procedure that are appropriate to capture the reusability approach.

We will assume that predicate weighted programs contain *at least one constant symbol*, so the Herbrand universe of the underlying language is non-empty.

By a *generalized Herbrand base* of a weighted logic program P we mean the set

$$\mathcal{H}^g(P) = \mathcal{H}(P) \times R^+,$$

where R^+ is the set of non-negative reals, and $\mathcal{H}(P)$ is the Herbrand base of $u(P)$. We call the elements of $\mathcal{H}^g(P)$ *resourced atoms*, that is, atoms with resources to cover costs of deriving them.

A subset $M \subseteq \mathcal{H}^g(P)$ is called a *resourced model* of a weighted program P if the following conditions are met

1. Whenever $C = a \stackrel{s}{\leftarrow} b_1, \ldots, b_n$ is a rule from P, Θ is a ground substitution of all variables in C and $\langle b_1\Theta, x_1\rangle \in M$, ..., $\langle b_n\Theta, x_n\rangle \in M$, then $\langle a\Theta, x_1 + \ldots + x_n + s\rangle \in M$.

2. Whenever $\langle a, x\rangle \in M$ and $y \geq x$, then $\langle a, y\rangle \in M$.

The following propositions establish basic properties of resourced models and their relationship to the standard notion of a model.

Proposition 4.1 *Let P be a weighted program. Then there exists a least resourced model of P.* □

In the remainder of the paper, the least resourced model of a weighted program P will be denoted by $LM(P)$.

Given a resourced model N of a program P, and an atom $a \in \mathcal{H}(P)$, the weight of a in N, $w_N(a)$, is defined by:

$$w_N(a) = \inf\{x : \langle a, x\rangle \in N\}$$

(in particular, $w_N(a) = \infty$ if for no x, $\langle a, x\rangle \in N$).

Proposition 4.2 *For every resourced model N of P and for every atom $a \in \mathcal{H}(P)$, $w_N(a) \leq w_{LM(P)}(a)$.*

Proof: Clearly, if a resourced atom $\langle p(t), x\rangle \in LM(P)$, then $\langle p(t), x\rangle \in N$. Consequently, $w_N(a) = \inf\{x : \langle a, x\rangle \in N\} \leq \inf\{x : \langle a, x\rangle \in LM(P)\} = w_{LM(P)}(a)$. □

The notion of the least resourced model can be characterized by means of the one-step derivability operator U_P. Namely, for every subset $A \subseteq \mathcal{H}^g(P)$ let us define

$$\begin{aligned} U_P(A) \;=\; & \{\langle a\Theta, x\rangle : a \stackrel{s}{\leftarrow} b_1, \ldots, b_n \in P,\ \Theta \text{ is a ground substitution, and there are reals} \\ & x_1, \ldots, x_n \text{ such that } \langle b_i\Theta, x_i\rangle \in A \text{ and } s + x_1 + \ldots + x_n \leq x\}. \end{aligned}$$

Following the analogy with logic programming, define

$$U_P^0(\emptyset) = \emptyset \;\text{ and }\; U_P^n(\emptyset) = U_P(U_P^{n-1}(\emptyset)), \;\text{ for } n \geq 1.$$

Finally, define

$$U_P^\omega(\emptyset) = \bigcup_{n=0}^{\infty} U_P^n(\emptyset).$$

We have the following characterization of the least resourced model in terms of the operator U_P.

Proposition 4.3 *Let P be a weighted logic program.*

1. *The operator U_P is compact (and, thus, monotone)*

2. *The least resourced model of P is equal to $U_P^\omega(\emptyset)$. That is, $LM(P) = U_P^\omega(\emptyset)$.* □

The next result states that if an atom a (in conjunction with some cost) is in the least resourced model of a weighted program P (that is, at some price it can be derived from P), then a is entailed by the underlying standard logic program $u(P)$. The converse also holds. That is, if an atom a is entailed by $u(P)$, then there is a cost for which it can be derived from P. In the proof and also later in the paper we will refer to the one-step provability operator of van Emden and Kowalski for the program $u(P)$. We will denote it by T_P the one-step provability operator of van Emden and Kowalski for the logic program $u(P)$ underlying the weighted program P. Let us also denote by $T_P^n(\emptyset)$ and $T_P^\omega(\emptyset)$ the n^{th} iteration of T_P over the empty set and the least fixpoint of T_P, respectively.

Proposition 4.4 *Let N be the least Herbrand model of $u(P)$. Then, for every $a \in \mathcal{H}(P)$, there is x such that $\langle a, x\rangle \in LM(P)$ if and only if $a \in N$.*

Proof: It follows by an easy induction that for every $n \geq 1$

$$T_P^n(\emptyset) = \{a\colon \langle a, x\rangle \in U_P^n(\emptyset), \quad \text{for some } x \geq 0\}.$$

Thus,

$$T_P^\omega(\emptyset) = \{a\colon \langle a, x\rangle \in U_P^\omega(\emptyset), \quad \text{for some } x \geq 0\}$$

and the assertion follows. □

We will now establish a connection between the notions introduced in this section and the no-reusability approach to weighted logic programming.

Theorem 4.5 *Let P be a weighted logic program.*

1. *Let $\langle a, x\rangle \in \mathcal{H}^g(P)$. There is k such that $\langle a, x\rangle \in U_P^k(\emptyset)$ if and only if there is a derivation tree ε such that $cst^{nr}(\varepsilon) \leq x$ and $\varepsilon \vdash_{nr} a$.*

2. *For every atom a, $w_{LM(P)}(a) = cst^{nr}(a)$.*

Proof: Assume that $\langle a, x\rangle \in U_P^k(\emptyset)$. We will prove by induction on k that there is a derivation tree ε such that $cst^{nr}(\varepsilon) \leq x$ and $\varepsilon \vdash_{nr} a$. This is, clearly, the case for $k = 0$ ($U_P^0(\emptyset) = \emptyset$).

Assume that the claim holds for an integer $k \geq 0$. We will show that the claim holds for $k + 1$, as well. Let $\langle a, x\rangle \in U_P^{k+1}(\emptyset)$. Then, there is a rule $C = p \stackrel{s}{\leftarrow} q_1, \ldots, q_m \in P$, reals $x_1, \ldots, x_m$ and a ground substitution Θ such that

1. $p\Theta = a$
2. $\langle q_i\Theta, x_i\rangle \in U_P^k(\emptyset)$
3. $x \geq s + x_1 + \ldots + x_m$.

By the induction hypothesis, for every i, $1 \leq i \leq m$, there is a derivation tree ε_i such that $cst^{nr}(\varepsilon_i) \leq x_i$ and $\varepsilon \vdash_{nr} q_i\Theta$. The tree obtained by adding a new vertex v, labeled by $C\Theta$, and making the roots of ε_i the children of v, is a derivation tree of a. Moreover, it is easy to see that

$$cst^{nr}(\varepsilon) = s + \sum_{i=1}^{m} cst^{nr}(\varepsilon_i) \leq s + \sum_{i=1}^{m} x_i \leq x.$$

Thus, the inductive step follows.

Conversely, assume that ε is a derivation tree of a and that $cst^{nr}(\varepsilon) \leq x$. We will show that there is k such that $\langle a, x\rangle \in U_P^k(\emptyset)$. The proof will be by induction on the depth of ε. Let v be the root of ε and let $v_1, \ldots, v_k$ be the children of v. Denote by r_i the rule that labels v_i. Denote by q_i the head of r_i. Finally, denote by ε_i the subtree of ε rooted in v_i.

Clearly, ε_i is a derivation tree for q_i. Since its depth is smaller than the depth of ε, it follows that there is k_i such that $\langle q_i, cst^{nr}(r_i)\rangle \in U_P^{k_i}(\emptyset)$. Hence, there is k such that $\langle q_i, cst^{nr}(r_i)\rangle \in U_P^k(\emptyset)$, for every i, $1 \leq i \leq n$. Consequently, $\langle a, cst^{nr}(\varepsilon)\rangle$ and $\langle a, x\rangle$ are in $U_P^{k+1}(\emptyset)$. This completes the proof of part (1). Part (2) is a direct consequence of part (1). □

Next, we will describe the resolution procedure for weighted programs. A *weighted goal* is a pair $\langle L, w\rangle$ where L is a list of atoms (possibly with variables) and $w \geq 0$. Let $G = \langle (a_1, \ldots, a_k), w\rangle$ be a weighted goal. Consider a clause

$$C = p \stackrel{s}{\leftarrow} b_1, \ldots, b_m$$

from P such that $w \geq s$ and the most general unifier of a_j and p, say Θ. The *resolvent* of G and C is the weighted goal

$$\langle (a_1\Theta, \ldots, a_{j-1}\Theta, b_1\Theta, \ldots, b_m\Theta, a_{j+1}\Theta, \ldots, a_k\Theta), w - s.\rangle$$

If $w < s$, or if p cannot be unified with any of the atoms in G, G and C are not resolvable.

A *resolution derivation* from P of a weighted goal $\langle L, w\rangle$ is a sequence of pairs $\langle G_i, C_i\rangle_{i=0}^n$ where $G_0 = \langle L, w\rangle$, $G_n = \langle \mathbf{nil}, t\rangle$, each G_{j+1} is the resolvent of G_j and C_j, and $t \geq 0$. Notice that C_n is not applied in the derivation (any clause can be used for C_n). Define

$$R_P = \{\langle p, w\rangle \in \mathcal{H}^g(P) \colon \langle p, w\rangle \text{ has a resolution derivation from } P\}.$$

It is easy to see that R_P is *upward closed*, that is, if $\langle p, w\rangle \in R_P$ then $\langle p, w'\rangle \in R_P$ for every $w' \geq w$.

Let us observe that since the weights of rules are constant and do not change when the variables are instantiated, a lifting lemma obviously holds. Consequently, we obtain the following useful result.

Lemma 4.6 *For every ground atom p and nonnegative real w, a resourced atom $\langle p, w\rangle$ possesses a resolution derivation from P if and only if it possesses a resolution derivation from ground(P).*

We will show that the least resourced model of a weighted program P, $LM(P)$, coincides with R_P (soundness and completeness result). To this end we need to define an operation of combination of resolution sequences. Let D_1, D_2 be two resolution derivations:

$$D_1 = \langle\langle\langle L_1^1, x_1^1\rangle, C_1^1\rangle, \ldots, \langle\langle L_{k_1}^1, x_{k_1}^1\rangle, C_{k_1}^1\rangle\rangle$$
$$D_2 = \langle\langle\langle L_1^2, x_1^2\rangle, C_1^2\rangle, \ldots, \langle\langle L_{k_2}^2, x_{k_2}^2\rangle, C_{k_2}^2\rangle\rangle$$

We define the *combination* $\otimes$ of derivations D_1, D_2 as the sequence $D = D_1 \otimes D_2$:

$\langle\langle\langle L_1^1 \circ L_1^2, x_1^1 + x_1^2\rangle, C_1^1\rangle,$
$\langle\langle L_2^1 \circ L_1^2, x_2^1 + x_1^2\rangle, C_2^1\rangle,$
$\ldots$
$\langle\langle L_{k_1}^1 \circ L_1^2, x_{k_1}^1 + x_1^2\rangle, C_1^2\rangle,$
$\langle\langle L_2^2, x_{k_1}^1 + x_2^2\rangle, C_2^2\rangle,$
$\ldots$
$\langle\langle L_{k_2}^2, x_{k_1}^1 + x_{k_2}^2\rangle, C_{k_2}^2\rangle\rangle,$

where, $\circ$ stands for the concatenation of lists. Notice that the clause $C_{k_1}^1$ disappears from this definition. The operation of combination of resolution derivations is extended to sequences collections of derivations by setting

$$D_1 \otimes \ldots \otimes D_m = (\ldots((D_1 \otimes D_2) \otimes D_3)\ldots)$$

Without a proof we state the following lemma.

Lemma 4.7 *Let $D_1, \ldots, D_m$ be resolution derivations of weighted goals $\langle L_1, w_1\rangle, \ldots, \langle L_m, w_m\rangle$. Then $D_1 \otimes \ldots \otimes D_m$ is a resolution derivation of the weighted goal $\langle L_1 \circ \ldots \circ L_m, w_1 + \ldots + w_m\rangle$.*

We will use Lemma 4.7 to characterize the least resourced model of a weighted program P in terms of resolution derivations.

Theorem 4.8 *Let P be a weighted logic program. Then, $LM(P) = R_P$.*

Proof. Let $C = p \overset{s}{\leftarrow} q_1, \ldots, q_n \in \mathit{ground}(P)$ and assume that $\langle q_1, z_1\rangle, \ldots, \langle q_n, z_n\rangle \in R_P$. By the definition of R_P and Lemma 4.6, for every i, $1 \leq i \leq k$, there exists a resolution derivation D_i of $\langle q_i, z_i\rangle$ from $\mathit{ground}(P)$.

Let $D' = D_1 \otimes \ldots \otimes D_n$. Assume

$$D' = \langle\langle\langle L_1, u_1\rangle, C_1\rangle \ldots, \langle\langle L_k, u_k\rangle, C_k\rangle\rangle$$

Define

$$D = \langle\langle\langle p, s + u_1\rangle, C\rangle, \langle\langle L_1, u_1\rangle, C_1\rangle, \ldots, \langle\langle L_k, u_k\rangle, C_k\rangle\rangle$$

It is easy to see that D is a resolution derivation of $\langle p, s + u_1\rangle$ from *ground*(P). By Lemma 4.6, $\langle p, s + u_1\rangle$ has a resolution derivation from P. That is, $\langle p, s + u_1\rangle$ belongs to R_P. Since $u_1 = z_1 + \ldots + z_n$, $\langle p, s + z_1 + \ldots + z_n\rangle \in R_P$. Since R_P is upward closed, it follows that R_P is a model of P. Consequently, $LM(P) \subseteq R_P$.

It remains to be shown that $R_P \subseteq LM(P)$. To this end we will prove the following claim:

(C) If $L = (p_1, \ldots, p_k)$ is a list of ground atoms, w is a non-negative real and $\langle L, w\rangle$ has a resolution derivation from *ground*(P), then there exist non-negative reals $x_1, \ldots, x_k$ such that $\langle p_1, x_1\rangle, \ldots, \langle p_k, x_k\rangle \in LM(P)$ and $x_1 + \ldots + x_k \leq w$.

We will prove the claim by induction on the length of a resolution derivation of $\langle L, w\rangle$. Let

$$D = \langle\langle\langle L_1, w_1\rangle, C_1\rangle, \ldots, \langle\langle L_m, w_m\rangle, C_m\rangle\rangle$$

be a resolution derivation of $\langle L, w\rangle$ (hence $w_1 = w$). Assume the claim holds for all weighted goals with a resolution derivation shorter than m.

The sequence

$$D' = \langle\langle\langle L_2, w_2\rangle, C_2\rangle, \ldots, \langle\langle L_m, w_m\rangle, C_m\rangle\rangle$$

is a resolution derivation of length $m - 1$ of the weighted goal $\langle L_2, w_2\rangle$. Let $L_2 = (q_1, \ldots, q_n)$. There is an atom in the list L_1 (without loss of generality, we will assume that it is the last atom in the list, p_k) so that for the clause

$$C_1 = p_k \overset{s}{\leftarrow} h_1, \ldots, h_l$$

1. $w_2 = w_1 - s$
2. $L_2 = (q_1, \ldots, q_n) = (p_1, \ldots, p_{k-1}) \circ (h_1, \ldots, h_l)$.

By the induction hypothesis, there exist non-negative reals $y_1, \ldots, y_n$ such that

1. $y_1 + \ldots + y_n \leq w_2$, and
2. $\langle q_1, y_1\rangle, \ldots, \langle q_n, y_n\rangle \in LM(P)$.

Define

1. $x_i = y_i$, for $i < k - 1$,
2. $x_k = s + y_k + \ldots + y_{k+l-1}$

Since $\langle p_i, x_i\rangle = \langle q_i, y_i\rangle$, for $i \leq k - 1$, $\langle p_i, x_i\rangle \in LM(P)$ for $i \leq k - 1$ (by the induction hypothesis). Since $\langle q_j, y_j\rangle, \ldots, \langle q_{j+l-1}, y_{j+l-1}\rangle \in LM(P)$ (by the induction hypothesis), and $C \in$ *ground*(P), $\langle p_k, s + y_k + \ldots + y_{k+l-1}\rangle \in LM(P)$. Now, observe that $x_k = s + y_k + \ldots + y_{k+l-1}$. Hence, $\langle p_k, x_k\rangle \in LM(P)$. Clearly, $x_1 + \ldots + x_k = y_1 + \ldots + y_n + s \leq w_2 + s = w_1 = w$. Hence, the claim follows.

Let $\langle p, w\rangle \in R_P$. Then, $\langle p, w\rangle$ has a resolution derivation from *ground*(P). Now, the claim implies that for some $s \leq w$, $\langle p, s\rangle \in LM(P)$. Since $LM(P)$ is closed upwards, $\langle p, w\rangle \in LM(P)$. This completes the argument. □

The results of this section can be explained by an observation that, under no-reusability approach, every weighted logic program can be interpreted as a constraint logic program. This interpretation requires a modification of the language and uses constraints of a special syntactic form.

To modify the language, for every predicate p of the original language $\mathcal{L}$ we introduce a new predicate p'. The predicate p' has one additional variable (which will be the last variable of p'). This variable ranges over *non-negative* reals. Second, we will use an additional predicate $\leq$ for reals, and we will use a symbol $+$ for real addition.

Consider a clause C in the language $\mathcal{L}$:

$$C = p(t) \stackrel{x}{\leftarrow} q_1(t_1), \ldots, q_k(t_k).$$

We define the corresponding constraint logic programming clause over the domain R^+ (non-negative reals) by:

$$clp(C) = p'(t, X) \leftarrow (x + X_1 + \ldots + X_k \leq X), q_1'(t_1, X_1), \ldots, q_k'(t_k, X_k).$$

Here, $x + X_1 + \ldots + X_k \leq X$ is the constraint of the clause $clp(C)$. Then, for a weighted program P, we define $clp(P) = \{clp(C) : C \in P\}$.

Given an arbitrary constraint domain D a *D-interpretation* is a set of atoms in which every variable ranging over D is replaced by a constant (denoting an element from D) and all other variables are replaced by ground terms (those terms can contain constants for elements from D). By a *D-valuation* we mean a function that assigns ground terms to variables not ranging over D and constants for elements of D, otherwise. Each such valuation uniquely extends to a valuation of all terms and formulas (including those of the language of D). The effect of a D-valuation v on a constraint c is denoted by $v(c)$ [1]. Similarly, a formula $v(b)$ is the effect of the valuation v on an atom b. Following [JM94], we can assign to a constraint logic program Q over a domain D an operator T_Q^D mapping D-interpretations to D-interpretations and defined as follows:

$$\begin{aligned} T_Q^D(I) = \{p(d) : \quad & p(X) \leftarrow c, b_1, \ldots, b_n \text{ is a rule of } Q, a_i \in I,\ i = 1, \ldots, n, \\ & v \text{ is a } D\text{-valuation such that} \\ & D \models v(c), v(X) = d, \text{ and } v(b_i) = a_i, i = 1, \ldots, n\} \end{aligned}$$

We have now the following fact.

Theorem 4.9 *Let P be a weighted logic program and let x be a real number. Let $p(t)$ be a ground atom. Then, $\langle p(t), x\rangle$ belongs to the least resourced model of P if and only if $p'(t, x)$ belongs to $T_{clp(P)}^{R^+} \uparrow \omega(\emptyset)$.*

[1] We follow the terminology of [JM94]. The notation $D \models c[v]$ instead of $D \models v(c)$ is often used.

Proof: We show by induction on n that $\langle p(t), x\rangle \in U_P^n(\emptyset)$ if and only if $p'(t,x) \in T_{clp(P)}^{R^+} \uparrow n(\emptyset)$. The case of $n = 0$ is evident. To deal with the induction step, we will assume that $n = 1$. The general case is similar. Assume that for a sequence of ground terms t, $\langle p(t), x\rangle \in U_P^1(\emptyset)$. It follows that there is a rule $p(u) \stackrel{s}{\leftarrow} \in P$ and a substitution Θ such that $u\Theta = t$ and $s \leq x$. Then, $p'(u, X) \leftarrow s \leq X \in clp(P)$. Define a R^+-valuation v by setting $v(X) = x$ and using Θ to define all other variable assignments. Then $v(s \leq X) = s \leq x$ is true in R^+. Thus $p'(t,x)$ belongs to $T_{clp(P)}^{R^+} \uparrow 1(\emptyset)$.

Conversely, if $p'(t,x) \in T_{clp(P)}^{R^+} \uparrow 1(\emptyset)$ then there is a rule $p'(u, X) \leftarrow (s \leq X) \quad \in clp(P)$ and a D-valuation v such that $v(X) = x$ and $s \leq x$. Define Θ by assigning to variables not ranging over R^+ ground terms assigned to those variables by v. It follows that $p(u) \stackrel{s}{\leftarrow}$ is in P. Moreover, since $s \leq x$ and $u\Theta = t$, $\langle p(t), x\rangle \in U_P^1(\emptyset)$. □

Our results imply that there is a class of constraint logic programs for which there are polynomial algorithms that decide whether a ground atom is entailed by the program. Specifically, let $\mathcal{K}_1$ be the class of constraint logic programs built of clauses of the following two types:

1. $p(X) \leftarrow s \leq X$
2. $p(X) \leftarrow s + X_1 + \ldots + X_k \leq X, p_1(X_1), \ldots, p_k(X_k)$

where p and p_i, $1 \leq i \leq k$, are unary predicate symbols over non-negative reals and s is a non-negative real. Our results on the complexity of a single-atom-derivation problem from Section 3 and Theorem 4.9 yield the following result.

Theorem 4.10 *There is an algorithm that decides whether a finite constraint program from class $\mathcal{K}_1$ entails a ground atom and that runs in time $O(m + n \log n)$, where m is the size of the program and n is the number of atoms appearing in the program.*

There are counterparts to the notions of generalized Herbrand base and one-step provability operator that lead to the reusability approach. However, an important modification seems to be necessary. Rather than to assign costs only to single atoms (as was sufficient for the no-reusability case), costs need to be assigned to finite collections of atoms under the reusability interpretation of costs. (This may explain, in particular, differences in the complexity of reasoning under the two models.)

Let At be a set of ground atoms of some first-order language. Define $\mathcal{H}_{fin}^g(P) = \{\langle X, w\rangle : X \subseteq At,\ |X| < \omega,\ w \geq 0\}$. This set will be referred to as *power Herbrand base* for a weighted logic program P. The elements of this set will be called *ground power atoms*. By a *power model* of a weighted logic program P we mean each set $M \subseteq \mathcal{H}_{fin}^g(P)$ such that:

> for every $\langle X, w\rangle \in M$ and for every rule $a \stackrel{s}{\leftarrow} b_1, \ldots, b_k \in ground(P)$, if $\{b_1, \ldots, b_k\} \subseteq X$, then $\langle X \cup \{a\}, x\rangle \in M$, for every $x \geq w + s$.

It is easy to see that $\mathcal{H}_{fin}^g(P)$ is a power model of P and that the intersection of power models is a power model. Hence, the least power model of a weighted logic program exists. We will denote it by $LM_p(P)$.

We say that weighted program P and a ground power atom $\langle X, w\rangle \in \mathcal{H}^g_{fin}(P)$ *imply* a ground power atom $\langle X', w'\rangle$ if there is a rule $a \stackrel{s}{\leftarrow} b_1, \ldots, b_k \in \mathit{ground}(P)$ such that $\{b_1, \ldots, b_k\} \subseteq X$, $X' = X \cup \{a\}$ and $w' \geq w + s$. In such case, we will write $P, \langle X, w\rangle \vdash \langle X', w'\rangle$.

We will define now a generalization of the one-step provability operator. Specifically, for any set $M \subseteq \mathcal{H}^g_{fin}(P)$ define

$$S_P(M) = \{\langle X', w'\rangle : \text{there is } \langle X, w\rangle \in M \text{ such that } P, \langle X, w\rangle \vdash \langle X', w'\rangle\}.$$

Several classic results from logic programming can be extended to the setting of power Herbrand base and related notions. The proofs are not difficult and we omit them. In particular, it is easy to show that $M \subseteq \mathcal{H}^g_{fin}(P)$ is a power model of a weighted logic program P if and only if $S_P(M) \subseteq M$. It is also easy to see that S_P is compact and, thus, monotone. Hence, S_P has a unique least fixpoint over $\emptyset$. This fixpoint coincides with the least power model of P. The connection to weighted logic programming under the reusability interpretation is given by the following result.

Theorem 4.11 *Let P be a weighted program. The least fixpoint of S_P exists and consists of precisely those pairs $\langle X, t\rangle$ for which $cst^{reu}(X) \leq t$. This least fixpoint coincides with the least power model of P.*

We will now introduce an appropriate notion of resolution. By a *power goal* we mean a pair $\langle X, w\rangle$, where X is a finite set of atoms of the language and w is a non-negative real. Consider an atom $q \in X$ and a rule $C = a \stackrel{s}{\leftarrow} b_1, \ldots, b_k$ from P such that $w \geq s$. Assume that Θ is a most general unifier for q and a. Then the power goal $\langle (X\Theta \setminus \{q\Theta\}) \cup \{b_1\Theta, \ldots, b_k\Theta\}, w - s\rangle$ is called a *resolvent* of $\langle X, w\rangle$ and C. Observe that, unlike in the previous case, here the first component of the resolvent is a *set* of atoms — repetitive occurrences are ignored.

A *power resolution derivation* of a power goal $\langle X, w\rangle$ from P is a sequence of pairs $\langle G_i, C_i\rangle_{i=0}^n$ where $G_0 = \langle X, w\rangle$, $G_n = \langle \emptyset, t\rangle$, each G_{j+1} is the resolvent of G_j and C_j, and $t \geq 0$. Define

$$R'_P = \{\langle X, w\rangle \in \mathcal{H}^g_{fin}(P) : \langle X, w\rangle \text{ has a power resolution derivation from } P\}.$$

The following result establishes soundness and completeness of power resolution method.

Theorem 4.12 *Let P be a weighted logic program. Then, $LM_p(P) = R'_P$.*

5 Cost as time under unlimited parallelism

In this section we will discuss yet another interpretation of cost in weighted logic programs. Despite the difference in the way the cost is interpreted, all of the results of

Section 4 (or their appropriate modifications) can be established for this new system, as well. This indicates that even more general treatment of costs in logic programming is possible. We will outline some of the possible directions in the last section of the paper.

The interpretation discussed in this section leads to a system that is closely related to the logic programming for confidence factors that was proposed by van Emden [vE86]. The system by van Emden and the connection will be discussed in the next section.

Let us start again with the "production" intuition behind no-reusability. A rule

$$a \stackrel{s}{\leftarrow} b_1, \ldots, b_n$$

encodes a piece of knowledge that an item a can be obtained from items $b_1, \ldots, b_n$ at cost s. Let us interpret s not as the cost of assembling $b_1, \ldots, b_n$ into a but as the *time* that is required. Assuming that no two tasks can be done in parallel, our no-reusability cost cst^{nr}, defined in Section 2, measures the total time required for the derivation. What we will study in this section is a *parallel* version of this measure. Namely, we will assume that we have enough resources to "produce" $b_1, \ldots, b_n$ in parallel. Moreover, every rule necessary in the "production" of each b_i will also be assumed to be executed in parallel.

Let ε be a derivation tree. We define the *parallel cost* of a vertex v in ε by induction. We will denote this cost by $cst^p(v)$. Assume that v is labeled by a rule r and that the parallel costs of all children $v_1, \ldots, v_k$ of a vertex v have been already computed. Then, the parallel cost of v is defined by

$$cst^p(v) = wt(r) + \max\{cst^p(v_i) \colon i = 1, \ldots, k\}.$$

The parallel cost of the derivation tree ε, $cst^p(\varepsilon)$, is now defined as the cost of the root of ε. This implicitly assumes a no-reusability model. All atoms proved "on the way" to deriving the root of ε are used up in the "production" process and only the head of the rule that labels the root of the derivation remains. It is easy to see that the cost of a derivation forest under the parallel interpretation can be equivalently defined as the largest sum of weights of rules used to label the nodes of a path in the derivation forest (counting the cost of each rule as many times as it appears as a label on the path).

For a derivation forest ε, its parallel cost $cst^p(\varepsilon)$ is defined as the maximum of the costs of all its trees. This implies that the set-of-atoms-derivation problem and the all-atoms-derivation problem can be reduced to the single-atom-derivation problem. From now on we will focus on this last problem only. Given a ground atom a, we define

$$cst^p(a) = \inf\{cst^p(\varepsilon) \colon \varepsilon \text{ is a derivation tree for } a\}.$$

In this section, we will study techniques to compute this measure.

Let us consider an example. Define P to be a weighted logic program consisting of the following rules:

(1) $s \stackrel{3}{\leftarrow} q, r$
(2) $q \stackrel{2}{\leftarrow}$
(3) $r \stackrel{1}{\leftarrow}$.

Clearly, there is a derivation tree for s (consisting of the root and two children, the root being labeled with rule (1) and the children with rules (2) and (3)). Under both reusability and no-reusability interpretations, the cost of this derivation tree is 6. On the other hand, under the parallel interpretation, the cost of this derivation tree is 5. It is also easy to see that this is an optimal derivation of s. Hence, $cst^p(s) = 5$.

The results of Section 4 can be extended to the case of this new interpretation of costs. In particular, there is a corresponding notion of a *p-model*, and a sound and complete resolution procedure. We will now define these notions and state the appropriate results. The proofs are similar to those in Section 4 and are omitted.

A subset $M \subseteq \mathcal{H}^g(P)$ is called a *p-model* of a weighted program P if the following conditions are met

1. Whenever $C = a \stackrel{s}{\leftarrow} b_1, \ldots, b_n$ is a rule from P, Θ a ground substitution of all variables in C, $\langle b_1\Theta, x_1\rangle \in \mathcal{M}$, ..., $\langle b_n\Theta, x_n\rangle \in M$, then $\langle a\Theta, s + \max\{x_1, \ldots, x_n\}\rangle \in M$.

2. Whenever $\langle a, x\rangle \in M$, $y \geq x$, then $\langle a, y\rangle \in M$

For a p-model M of a weighted logic program P, we define

$$w^p_M(a) = \inf\{x\colon \langle a, x\rangle \in M\}.$$

Reasoning as in Section 4, one can prove that a least p-model exists.

Proposition 5.1 *Let P be a weighted logic program. Then P has a least p-model.*

Let us briefly note that in the case of bi-horn programs there is no parallelism within the clauses. Consequently, we have the following result.

Proposition 5.2 *If P is bi-horn, then the least resourced model and the least p-model coincide.*

The least p-model of P can be characterized by means of the appropriate one-step provability operator. Specifically, for every subset $A \subseteq \mathcal{H}^g(P)$ let us define

$$\begin{aligned} U^p_P(A) \;=\; & \{\langle a\Theta, x\rangle\colon a \stackrel{s}{\leftarrow} b_1, \ldots, b_n \in P,\ \Theta \text{ is a ground substitution, and there are reals} \\ & x_1, \ldots, x_n \text{ such that } \langle b_i\Theta, x_i\rangle \in A \text{ and } s + \max\{x_1, \ldots, x_n\} \leq x\}. \end{aligned}$$

Proposition 5.3 *Let P be a weighted logic program.*

1. *The operator U^p_P is compact (and, thus, monotone)*

2. *The least p-model of P is equal to the least fixpoint of U^p_P over the empty set.*

Least resourced models and costs of derivations under the parallel interpretation of the cost are related.

Proposition 5.4 *Let P be a weighted logic program and let M be its least p-model. For every atom $a \in LM(P)$, we have $w^p_M(a) = cst^p(a)$.*

The parallel interpretation of weighted programs has also a complete resolution procedure. This procedure is similar to the resolution procedure described in Section 4. However, there are some important differences. First, instead of weighting entire goal we weight individual atoms. Specifically, a *p-weighted goal* is a list $L = \langle\langle p_1, s_1\rangle, \ldots, \langle p_m, s_m\rangle\rangle$, where for all i, $1 \leq i \leq m$, p_i is a ground atom and s_i is a non-negative real. Now, given such p-weighted goal L and a clause $C = p \stackrel{s}{\leftarrow} q_1, \ldots, q_m$ such that p_i and p are unifiable by a most general unifier Θ, the resolvent of L and C is

$$\langle\langle p_1\Theta, s_1\rangle, \ldots, \langle p_{i-1}\Theta, s_{i-1}\rangle, \langle q_1\Theta, s_i - s\rangle, \ldots, \langle q_n\Theta, s_i - s\rangle, \langle p_{i+1}\Theta, s_{i+1}\rangle, \ldots, \langle p_m\Theta, s_m\rangle\rangle$$

providing $s_i \geq s$ (otherwise resolution cannot be performed). The notion of a p-resolution derivation of a weighted goal is analogous to the one from Section 4.

With this form of the resolution we have the following result characterizing the least p-model of P.

Proposition 5.5 *The least p-model of P consists precisely of those resourced atoms for which the p-resolution derivation exists.*

Next, we show that we can interpret the weighted logic programs with the parallel interpretation of the cost as constraint logic programs (cf. a similar result, Theorem 4.9 in Section 4). However, although the constraint domain remains the same (the set R^+ of non-negative reals), the max operation is needed (together with the + operation and the $\leq$ relation).

The interpretation of a clause $C = p(t) \stackrel{s}{\leftarrow} q_1(t_1), \ldots, q_n(t_n)$ is

$$clp^p(C) = p'(t, X) \leftarrow (X \geq s + \max\{X_1, \ldots, X_n\}), q'_1(t_1, X_1), \ldots, q'_n(t_n, X_n).$$

Given a weighted logic program P, let us denote by $clp^p(P)$ the constraint logic program obtained by replacing each rule C by its constraint logic programming interpretation $clp^p(C)$. We now have the following theorem.

Theorem 5.6 *Let P be a weighted logic program. Let $p(t)$ be a ground atom. Then, $\langle p(t), x\rangle$ belongs to the least p-model of P if and only if $p'(t, x)$ belongs to $T^{R^+}_{clp^p(P)} \uparrow \omega(\emptyset)$.*

It is well known that the shortest-path problem remains in P if the length of a path is computed as the maximum of the weights of its edges rather then the sum. In fact, a minor modification of the Dijkstra algorithm can be used to solve it. The same is true when we switch from the no-reusability interpretation to the parallel interpretation of costs. We have the following result.

Theorem 5.7 *The problems of single atoms derivation, set-of-atoms-derivation and all-atom-derivation for parallel interpretation of weighted programs are all in the class P.*

In fact, as in Section 3, one can show that the all these problems can be solved in time $O(m + n \log n)$, where m is the size of the program and n is the number of atoms appearing in the program.

The results of this section allow us to introduce yet another class of constraint logic programs for which the entailment problem can be decided in polynomial time. Let $\mathcal{K}_2$ be the class of constraint logic programs built of clauses of the following two types:

1. $p(X) \leftarrow s \leq X$
2. $p(X) \leftarrow s + \max\{X_1, \ldots, X_k\} \leq X, p_1(X_1), \ldots, p_k(X_k)$

where p and p_i, $1 \leq i \leq k$, are unary predicate symbols over non-negative reals and s is a non-negative real. Theorems 5.6, together with our comments on the complexity of solving derivation problems under the parallel interpretation of costs, yield the following result.

Theorem 5.8 *There is an algorithm that decides whether a finite constraint program from class $\mathcal{K}_2$ entails a ground atom and that runs in time $O(m + n \log n)$, where m is the size of the program and n is the number of atoms appearing in the program.*

6 Logic programming for confidence factors

Weighted logic programs, with weights restricted to the interval $(0, 1]$, were considered by van Emden [vE86] as a way to incorporate confidence factors into logic programming. In that work, a rule

$$p \stackrel{s}{\leftarrow} q_1, \ldots, q_n$$

is assigned the following intuitive interpretation: if the confidence factors of $q_1, \ldots, q_n$ are at least $x_1, \ldots, x_n$, then the confidence factor of p is at least $s \times \min\{x_1, \ldots, x_n\}$. We will refer to weighted rules with weights from $(0, 1]$ as *cf-rules* and to weighted programs built of cf-rules as *cf-programs*.

The results obtained by van Emden are quite similar to those in Sections 4 and 5. In particular, van Emden defines an appropriate generalization of the Herbrand base of a program P by setting $\mathcal{H}^{cf}(P) = \mathcal{H}(P) \times (0, 1]$. He then defines the notion of a model of a cf-program appropriate for the confidence factor interpretation. A subset $M \subseteq \mathcal{H}^{cf}(P)$ is called a *cf-model* of a weighted program P if the following conditions hold:

1. Whenever $C = a \stackrel{s}{\leftarrow} b_1, \ldots, b_n$ is a rule from P, Θ a ground substitution of all variables in C, $\langle b_1\Theta, x_1\rangle \in \mathcal{M}$, ..., $\langle b_n\Theta, x_n\rangle \in M$, then $\langle a\Theta, s \times \min\{x_1, \ldots, x_n\}\rangle \in M$.
2. Whenever $\langle a, x\rangle \in M$ and $0 < y \leq x$, then $\langle a, y\rangle \in M$.

For a cf-model M of a cf-program P, we define

$$w_M^{cf}(a) = \sup\{x\colon \langle a, x\rangle \in M\}.$$

As defined by van Emden, cf-models are closed downward rather than upward. It is natural as if we believe an atom with confidence x, then we should believe in the atom with any confidence y, where $0 < y \leq x$. van Emden shows that the least cf-model of weighted program exists. He also describes a complete resolution procedure for deciding the membership of an atom from $\mathcal{H}^{cf}(P)$ in the least cf-model of P. This procedure is similar to the one described above for the parallel interpretation, except that one divides by s rather than subtracts it. These similarities are not accidental. It turns out that parallel interpretation and confidence factor interpretation of costs are isomorphic.

Indeed, let

$$C = p \stackrel{s}{\leftarrow} q_1, \ldots, q_n$$

be a cf-rule. Define

$$ve(C) = p \stackrel{-\log s}{\leftarrow} q_1, \ldots, q_n.$$

For a cf-program P, define $ve(P) = \{ve(C) : C \in P\}$. Finally, define a map $ve : \mathcal{H}^{cf}(P) \mapsto \mathcal{H}^{g}(P)$ by

$$ve(\langle p, s\rangle) = \langle p, -\log s\rangle.$$

This mapping is one-to-one and onto. It induces a one-to-one and onto mapping (overloading the notation, we will also refer to it as ve) between the subsets of $\mathcal{H}^{cf}(P)$ and the subsets of $\mathcal{H}^{g}(P)$. This induced mapping transforms downward closed subsets of $\mathcal{H}^{cf}(P)$ into upward closed subsets of $\mathcal{H}^{g}(P)$. We then have the following result establishing a precise relationship between cf-programs and weighted programs with the parallel interpretation of costs.

Theorem 6.1 *Let P be a cf-program and let M be a subset of $\mathcal{H}^{cf}(P)$. Then, M is a cf-model of P if and only if $ve(M)$ is a p-model of the weighted program $ve(P)$. In particular, M is the least cf-model of P if and only if $ve(M)$ is the least p-model of P.*

Using this relationship we can extend the remaining results from Section 5 to the case of cf-programs. In particular, one can define an appropriate measure of the cost of a derivation tree and derive the following characterization of the least cf-model. Specifically, let ε be a derivation tree. We define the *confidence factor cost* (cf-cost) of a vertex v in ε by induction. We will denote this cost by $cst^{cf}(v)$. Assume that v is labeled by a rule r and that the cf-costs of all children $v_1, \ldots, v_k$ of a vertex v have been already computed. Then, the cf-cost of v is defined by

$$cst^{cf}(v) = wt(r) \times \min\{cst^{cf}(v_i)\colon i = 1, \ldots, k\}.$$

The cf-cost of the derivation tree ε, $cst^{cf}(\varepsilon)$, is now defined as the cost of the root of ε. Given a ground atom a, define

$$cst^{cf}(a) = \sup\{cst^{cf}(\varepsilon)\colon \varepsilon \text{ is a derivation tree for } a\}.$$

Proposition 6.2 *Let P be a cf-program and let M be its least cf-model. For every atom a, we have $w^{cf}_M(a) = cst^{cf}(a)$.*

We can also construct an embedding of cf-programs into constraint logic programming over the domain $\langle (0,1], \times, \min, \leq \rangle$. Indeed, consider a cf-rule

$$C = p(t) \stackrel{s}{\leftarrow} q_1(t_1), \ldots, q_m(t_m)$$

and assign to it the rule

$$clp^{cf}(C) = p'(t, X) \leftarrow (X \leq s \times \min\{X_1, \ldots, X_m\}), q'_1(t_1, X_1), \ldots, q'_m(t_m.X_m).$$

Now, define $clp^{cf}(P) = \{clp^{cf}(C) : C \in P\}$. We then have the following result establishing a correspondence between cf-programs and constraint logic programming.

Theorem 6.3 *Let P be a cf-program. Let $M \subseteq \mathcal{H}^g(P)$ be the least cf-model of P. Then for every $\langle p(t), s\rangle \in \mathcal{H}^{cf}(P)$, $\langle p(t), s\rangle \in M$ if and only if $p'(t, s) \in T^{(0,1]}_{clp^{cf}(P)}\uparrow\omega(\emptyset)$.*

Complexity results for computing confidence factors of atoms and sets of atoms can be derived from Theorems 5.7 and 6.1 (the confidence factor of a set of atoms is defined as the minimum of confidence factors of its elements).

7 Weighted programs and generalized annotated logic programs

In [KS92] Kifer and Subrahmanian proposed a general scheme for treatment of annotated logic programs. We will give a short description of their construction, The generalized annotated logic programs (GAPs for short) consist of clauses of the following format:

$$p(\vec{X}) : t \leftarrow q_1(\vec{X}) : r_1, \ldots, q_m(\vec{X}) : r_m$$

A GAP is a collection of such clauses. A program of this type presupposes an upper semi-lattice $\mathcal{L} = \langle L, \sqcup\rangle$ of *annotations*. Terms t, $r_1, \ldots, r_m$ refer to elements of that semi-lattice. Terms $r_1, \ldots, r_m$ are either constants denoting elements of L or variables ranging over L. The term t is an expression of some functional language expanding that of semi-lattice $\mathcal{L}$. We will treat it as an algebra that endowes $\mathcal{L}$ with some additional operations. That is the signature of semi-lattice is expanded by a number of operations, $f_1, \ldots, f_n$. The operation $\sqcup$ generates a partial ordering $\sqsubseteq$ in L. Now, t is a term of that algebra. It depends on some of $r_1, \ldots, r_m$ and is always supposed to be $\sqsubseteq$-monotone in variables among $r_1, \ldots, r_m$. The choice of operations is not restricted as long as this monotonicity condition is satisfied.

Thus, assume that GAP program P and an algebra $\mathcal{L} = \langle L, \sqcup, f_1, \ldots, f_n\rangle$ is given. Kifer and Subrahmanian introduce the annotated universe (that is $\mathcal{H} \times L$) and annotated

base $At \times L$. The notion of the annotated model describes the intuitive closure of subsets of $B \times L$ under the rules treated as templates for grounded rules. The models are supposed to be $\sqsubseteq$-closed with respect to $\sqsubseteq$ on second coordinate. Kifer and Subrahmanian prove that GAPs possess the least annotated model. This model is complete with respect to some form of resolution. This resolution is a variant of "lazy evaluation". Roughly, the resolution is executed on the first coordinate (atoms of logical language) and the terms used as annotations generate constraints on elements of L. These constraints are collected in the process into a conjunction called reductant. When the resolution proper returnes empty clause one needs to check if the resulting constraint has a solution in the algebra $\langle L, \sqcup, f_1, \ldots, f_n \rangle$. If such solution exists, it generates an answer substitution for the query to the program P.

It should be clear that such general procedure, with many unknown parameters, does not allow for general complexity results. When the algebra $\mathcal{L}$ is finite, the constraints become decidable (this fact has been noticed in [KS92]) but not much more can be said.

By contrast, the resolution procedures complete for no-reusability interpretation of weighted programs and also for unlimited parallelism of interpretation of weighted programs are not lazy evaluation style – the constraints are solved in parallel with resolution and they can stop resolution at any stage. An interesting aside is the fact that, sometimes, different SLD resolution procedures are complete for the same class of programs!

Weighted programs under some semantics can be interpreted as GAPs. This happens for three out of four semantics considered above. Semantics of no-reusability cost, unlimited parallelism, and van Emden semantics can be represented by GAPs with appropriately selected algebras [2]. Here is how this can be done. To get no-reusability semantics, we consider the algebra $\langle R^+, \max, + \rangle$ of non-negative reals with the additional operation of addition. With this algebra, the weighted programs with nonreusability interpretation are represented as follows. A clause

$$C = p \stackrel{s}{\longleftarrow} q_1, \ldots, q_m$$

is interpreted by a annotated clause

$$ks_1(C) = p : (X_1 + \ldots X_m + s) \leftarrow q_1 : X_1, \ldots, q_m : X_m$$

A weighted program P is now interpreted as $ks_1(P) = \{ks_1(C) : C \in P\}$. Then, by an argument, essentially, of [KS92], p.356 (it shows the analogical result for what below is called ks_3 interpretation) we find that the least resourced model of P coincides with the least annotated model of the GAP program $ks_1(P)$.

For the unlimited parallelism case a similar, but slightly different algebra, and a different interpretation is needed. Specifically, consider the same algebra as above, but with a different interpretation of weighted clauses, assigning to a clause C as above an annotated clause

$$ks_2(C) = p : \max(X_1 + \ldots X_m) + s \leftarrow q_1 : X_1, \ldots, q_m : X_m$$

[2] We thank anonymous reviewer for bringing this to our attention.

and $ks_2(P) = \{ks_2(C) : C \in P\}$. Again, it is easy to see that the least p-resourced model of P coincides with the least annotated model of GAP program $ks_2(P)$.

In the case of van Emden interpretation of weighted clauses, the algebra is different (but it is clear what it does need to be, bt the isomorphism with the case of unlimited parallelism). It is $\langle (0,1], \min, * \rangle$. The interpretation of the clause C as above (now $0 < s \leq 1$) is

$$ks_3(C) = p : \min(X_1 + \ldots X_m) * s \leftarrow q_1 : X_1, \ldots, q_m : X_m$$

The isomorphism result is proven by Kifer and Subrahmanian in [KS92].

As noted above, the generality of the results of [KS92] makes it impossible to get sharp complexity results for their scheme. The complexity results shown above in Sections 3,5, and 6 as well as the isomorphism results stated above allows us to draw complexity results for *some* classes of annotated programs (with very specific algebras). Specifically, Theorems 3.2, 3.3, 4.10 and 5.7 (as well as analogous results for van Emden programs which follow directly from the Isomorphism Theorem 6.1) allow for reading off a number of complexity results for annotated logic programs (of restricted syntactic form) over some algebras. We will illustrate this contention by means of one example; we will read off a complexity result for some annotated programs over the algebra $\langle [0, \infty), \max + \rangle$. Clearly, we must restrict the syntax seriously since the ordinary (Horn) logic programming embeds trivially into GAP logic programming and so all recursively enumerable sets would be expressible.

To get these complexity results for some classes of annotated programs, we need to make precise what sort of programs we are looking at, as well as algebras. This second task will be clear from the context. Moreover, we need to specify the problems that we are going to solve.

To give the sample of the result about annotated programs using for instance Theorem 3.2, let us look at annotated programs of the following form. The atoms appearing in the program will be grounded. The annotations will not be necessarily grounded, but the following restrictions will be accepted:

1. Annotations in the bodies are either constants or variables
2. Annotations in the head will be of the form $t_1 + \ldots + t_k + r$ where $t_1, \ldots, t_k$ are all annotations appearing in the body, and r will be a non-negative real.

These will be called type-1 GAPs. We will now formulate the problems we are studying. Those are:

single-atom-cost: Given a type-1 GAP finite program P, an atom x, and an integer k, decide whether the pair $\langle x, k \rangle$ belongs to the least model of P.

set-of-atoms-cost: Given a type-1 GAP finite program P, a set of atoms X, and an integer k, decide whether there are reals $z_x, x \in X$ so that for all $x \in X$, $\langle x, z_x \rangle$ belongs to the least model of P and $\sum_{x \in X} z_x \leq k$.

least-model-cost: Given a type-1 finite GAP program P and an integer k, decide whether there is a an assignment z_x of nonnegative real numbers to the elements of the least model M of $u(P)$ so that $\sum_{x \in M} z_x \leq k$. (This is a special case of set-of-atoms-cost problem, when X is the least model M of $u(P)$, that is P stripped of annotations.)

We then have the following result.

Theorem 7.1 *1. If P is a type-1 GAP, then all three above problems are in P*

2. If, in addition, P is bi-Horn, then single-atom-cost can be solved by Dijkstra shortest path algorithm.

Other complexity results mentioned above lead to additional sharp results for other classes of GAP programs.

We conclude this section by mentioning that the type of constraints that can be used in heads of GAP programs and to which algorithms similar to those disccussed above apply include constraints of the form:

$$p * (X_1 + \ldots + X_k) + r.$$

We will discuss these and related isssues in the next section.

We conclude this section by noting that it is an open problem if the reusability interpretation of weighted programs can be expressed as an annotated program.

8 Conclusions and further research

The results of the paper point to several interesting directions for further work on logic programming with costs. There is close similarity in the type of results we were able to obtain for weighted logic programs under the no-reusability, parallel and confidence factor interpretations of costs. In addition, in each case the methods needed for proofs were very similar, too. It is of interest to find a *general* class of interpretations of costs with properties analogous to those we obtained for the no-reusability interpretation. Specifically, it is of interest to find a characterization of a class of interpretations for which there are polynomial algorithms for computing costs in the propositional case, and a sound and complete resolution procedure in the predicate case (without the need to resort to the notion of the power Herbrand base).

In particular, notice that all these interpretations (no-reusability, parallel and confidence factor) can be viewed as special cases of the following scheme. Let φ be an operation acting on finite lists of non-negative reals. Assign to a clause $p \stackrel{s}{\leftarrow} q_1, \ldots, q_n$ this interpretation: "If the atoms $q_1, \ldots, q_n$ can be computed with the costs $s_1, \ldots, s_n$ respectively, then p can be computed with the cost $\varphi(s, s_1, \ldots, s_n)$. Our results show that for some operations φ ($\varphi(s, s_1, \ldots, s_k) = s + s_1 + \ldots + s_k$, or $\varphi(s, s_1, \ldots, s_k) = s + \max(s_1, \ldots, s_k)$) there is a natural notion of a model and a corresponding "resolution" technique. Similarly, van Emden results show the same for $\varphi(s, s_1, \ldots, s_n) = s \times$

$\min(s_1, \ldots, s_n)$. In addition, one can easily see that the same holds for $\varphi(s, s_1, \ldots, s_n) = \max\{s, s_1, \ldots, s_n\}$. The question which other operations φ lead to the same results remains open.

In the paper we found substantial differences between no-reusability and reusability interpretations. Interestingly, for the "parallel" (time) interpretation of weighted logic programs reusability and non-reusability lead to the same lowest costs. The same holds for regeneration interpretation. Is there a characterization of interpretations for which these two approaches coincide?

It is possible to extend the formalism presented in this paper to the situation when the bodies of program rules may also contain negated atoms. Such rules admit several interpretations. We will mention here only one. We will restrict now to the case of propositional programs. Specifically, we interpret a rule C:

$$p \stackrel{s}{\leftarrow} q_1, \ldots, q_m, \neg r_1, \ldots, \neg r_n$$

as follows: " If $q_1, \ldots, q_m$ have been derived, and $r_1, \ldots, r_n$ are not and will not be derived, then derive p and decrease the amount of available resource by s".

It is easy to define a corresponding notion of a model. Namely, $M \subseteq \mathcal{H}^g(P)$ is a model of C if for all reals $s_1, \ldots, s_m$, whenever $\langle q_1, s_1\rangle \in M, \ldots, \langle q_m, s_m\rangle \in M$, and for all non-negative reals $t_1, \ldots, t_n$,

$$\langle r_1, t_1\rangle \notin M, \ldots, \langle r_n, t_n\rangle \in M,$$

then $\langle p, s + s_1 + \ldots + s_m\rangle \in M$.

We can now generalize the concept of a stable model. To illustrate the approach, assume the no-reusability interpretation of costs. Namely, $M \subseteq \mathcal{H}^g(P)$ is a nr-stable model for a weighted logic program P if it coincides with the least (resourced) model of the reduct of P with respect to M (the reduct is computed according to the Gelfond-Lifschitz definition — the costs are disregarded). Similar definitions can be given for other ways to interpret costs of the rules. This yields the notions of reu-stable and p-stable models.

An interesting observation is that the complexity of the existence problem for nr-stable models of cost at most k grows to NP-complete (from polynomial, in the Horn case), while it stays the same as in the Horn case (NP-complete) for reu-stable models.

Next, let us observe that clauses of logic programs can be assigned more than one weight. For instance, consider a clause:

$$p \stackrel{s,t}{\leftarrow} q_1, \ldots, q_n.$$

It might be interpreted as "if $q_1, \ldots, q_n$ can be computed with cost $s_1, \ldots, s_n$ respectively, then p can be obtained with cost $t \times (s + s_1 + \ldots + s_n)$". Such 2-parameter clauses could model situations where not only we apply a rule at some cost, but we also apply a discount (or premium, depending on the magnitude of t) when the rule is applied. Our results on no-reusability and parallel interpretations can be repeated in this case. In

particular, the notion of a model, a derivation, an appropriate resolution algorithm, and an embedding into some version of CLP can easily be constructed.

Finally, let us note that resolution procedures given in this paper together with our results on embeddings of weighted logic programming into CLP indicate that some classes of CLP programs have a particularly simple resolution procedure.

References

[DG84] W.F. Dowling and J.H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, 1(3):267–284, 1984.

[GGST86] H.N. Gabow, Z. Galil, T. Spencer, and R.E. Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6(2):109–122, 1986.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness*. W.H. Freeman and Co., San Francisco, Calif., 1979.

[JM94] J. Jaffar and M.J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19(20):503–581, 1994.

[KS92] M. Kifer and V.S. Subrahmanian. Theory of generalized annotated logic programs and its applications. *Journal of Logic Programming*, 12:335–367, 1992.

[Llo84] J. W. Lloyd. *Foundations of logic programming*. Symbolic Computation. Artificial Intelligence. Springer-Verlag, Berlin-New York, 1984.

[RRS+95] P. Rao, I.V. Ramskrishnan, K. Sagonas, T. Swift, and D. S. Warren. Efficient tabling mechanisms for logic programs. In *Proceedings of the 12th International Conference on Logic Programming*, Cambridge, MA, 1995. MIT Press.

[Tar77] R.E. Tarjan. Finding optimum branchings. *Networks*, 7(1):25–35, 1977.

[vE86] M.H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 3(1):37–53, 1986.