Nested Weight Constraints in ASP

Stefania Costantini†
Università di L’Aquila
Piazza V. Rivera 1, 67100 L’Aquila, Italy
Stefania.Costantini@univaq.it

Andrea Formisano
Università di Perugia
Piazza Università 1, 06123 Perugia, Italy
formis@dmi.unipg.it

Abstract. Weight constraints are a powerful programming construct that has proved very useful within the Answer Set Programming paradigm. In this paper, we argue that practical Answer Set Programming might take profit from introducing some forms of nested weight constraints. We define such empowered constraints (that we call “Nested Weight Constraints”) and discuss their semantics and their complexity.

Key words: Weight constraints in ASP, Non-monotonic reasoning, Knowledge representation.

1. Introduction

Answer Set Programming (ASP for short) [18], has evolved over more than two decades as a paradigm that allows for very elegant solutions to many combinatorial problems (see, e.g., [2, 4, 13, 14, 21, 31]). It has been successfully applied to many forms of knowledge representation and commonsense reasoning (cf., among others, [3, 17] and the references therein). The paradigm is based upon describing a problem by a logic program in such a way that its answer sets correspond to the solutions of the problem.

This paradigm has become even more powerful by extending the ASP language by means of weight constraints [24, 27]. Intuitively, weight constraints allow one to associate weights to the literals occurring

†Address for correspondence: Università di L’Aquila, Piazza V. Rivera 1, 67100 L’Aquila, Italy
in specific subsets of a (candidate) model. Then, bounds can be imposed on the overall weight of each subset. A model is accepted if these bounds are satisfied. Cardinality constraints are a special case where all weights are equal to 1. Weight constraints have proved to be a very useful programming tool in many applications such as planning and configuration. For instance, in the product configuration domain we need to express cardinality, cost, and resource constraints, which may be very difficult to capture using logic programs without specific constructs to model resources and amounts [6, 28].

Weight constraints are nowadays adopted (in some form) by most of the ASP inference engines (usually called “ASP solvers”). Related to weight constraints are aggregates, as introduced in the DLV system [20] and in [12] for the FO system. Aggregates have proved to be very useful in a number of practical applications. Like plain ASP, all common algorithmic tasks related to programs with weight constraints, such as checking the consistency of a program (i.e., whether a program admits answer sets, also called “stable models”), are intractable [11]. However, as shown in [26], tractability can be achieved by imposing some restrictions on program structure (this is the case also for aggregates).

We propose an improved form of constraints that admits nesting of weight constraints. The use of nesting allows one to specify a set of weight constraints within a “container” weight constraint. In turn, such “contained” constraints may include other constraints, and so on, for an arbitrary number of levels. Semantics is provided by requiring that the satisfaction of the internal constraints has to be evaluated with respect to the context defined by the outer constraints. Hence, the new construct introduces a form of locality in program rules: two identical weight constraints might be differently evaluated depending on the context in which they occur. We will see that nesting can be introduced without affecting complexity. Though for the sake of simplicity and conciseness we do not treat aggregates in this paper (as they are a more involved construct than weight constraints), the introduction of similar forms of nesting could be useful also for aggregates.

We argue that practical ASP programming might take profit from the introduction of nested weight constraints. In particular, our proposal is aimed at improving elaboration tolerance where, [23]:

A formalism is elaboration tolerant to the extent that it is convenient to modify a set of facts expressed in the formalism to take into account new phenomena or changed circumstances. Representations of information in natural language have good elaboration tolerance when used with human background knowledge. Human-level AI will require representations with much more elaboration tolerance than those used by present AI programs, because human-level AI needs to be able to take new phenomena into account. The simplest kind of elaboration is the addition of new formulas. We’ll call these additive elaborations. Next comes changing the values of parameters. Adding new arguments to functions and predicates represents more of a change. However, elaborations not expressible as additions to the object language representation may be treatable as additions at a meta-level expression of the facts . . .

One can say that elaboration tolerance implies the ability to cope with minor changes to input problems without major revisions. In general, the introduction of constructs involving forms of locality, as well as modularity, goes in this direction. By considering two representative sample problems, we will show that the formalization in ASP may benefit from the use of nested weight constraints.

The paper is structured as follows. After recalling the notions of weight (and cardinality) constraints, we introduce nested weight constraints for the case of ground programs. Then we further extend the formalism by introducing conditional literals [24] and the use of variables to denote collections of literals. We illustrate nested weight constraints by means of two case-studies, taken from the realm of scheduling and allocation, which is an important field of application of ASP. The complexity issue is addressed and then we discuss our approach and the complexity results with respect to possible future perspectives.
2. Weight and Cardinality Constraints in ASP

Weight and cardinality constraints were introduced in [24, 27], where their semantics is also presented, as well as their implementation in the context of the ASP solver smodels. Deciding whether a program involving ground weight constraints has an answer set is an NP-complete problem and computing an answer set is FNP-complete. Though the computational complexity remains the same with respect to plain ASP, the modeling power of the extended language is higher, as proved by the wide application of this construct. In what follows we recall (following [26]) the syntax and semantics of (ground) programs with weight constraints by abstracting away from any particular concrete syntax. We assume as known the usual notions of constant, predicate, term, atom, literal, etc.

Let us consider as fixed an underlying language and consequently let \( B \) denote the corresponding Herbrand base, namely the set of all ground atoms of the given language. Atoms have the form \( p(t_1, \ldots, t_k) \) where \( p \) is a predicate symbol and each \( t_i \) is a term. For a literal \( \ell \), let \( \pi(\ell) \) denote the predicate symbol of \( \ell \) (e.g., \( \pi(p(t_1, \ldots, t_k)) = p \)). For a set of literals \( S \), let \( \pi(S) = \{ \pi(\ell) | \ell \in S \} \).

A weight literal is a pair \((a, j)\) or \((-a, j)\), \(a \in B\), where \( j \in \mathbb{N} \) is the weight of the literal and \(-\) denotes default negation. A weight constraint is a triple \((S, l, u)\) where \( S \) is a set of weight literals and \( l \leq u \) are non-negative integers, the lower and upper bound. We will often use the symbol \( \infty \) to denote an arbitrarily large upper bound. (This will be useful in situations in which the upper bound is not specified.) Moreover, as a shorthand notation, we denote by \( a \) the weight constraint \((\{(a, 1)\}, 1, 1)\).

A rule \( r \) is a pair \((h, b)\) where \( h \) (the head) is a weight constraint and \( b \) (the body) is a set of weight constraints. We indicate \( h \) with \( H(r) \) and \( b \) with \( B(r) \). A (ground) program with weight constraints (for short, PWC) is a set of rules. Given a weight constraint \( c \) and a set of atoms \( I \), we define the weight of \( c \) in \( I \) as follows:

\[
W(c, I) = \sum_{(a,j) \in Cl(c) \land a \in I} j + \sum_{(-a,j) \in Cl(c) \land a \notin I} j.
\]

A set of atoms \( I \) is a model of \( c \) (denoted by \( I \models c \)) iff \( l(c) \leq W(c, I) \leq u(c) \). (Notice that the second inequality always holds if \( u(c) = \infty \).) For a set of weight constraints \( C \), \( I \models C \) iff \( I \models c \) for all \( c \in C \). Moreover, \( I \) is a model of a rule \( r \) (denoted by \( I \models r \)) iff \( I \models H(r) \) whenever \( I \models c \) holds for each \( c \in B(r) \). For a set of rules \( R \), \( I \models R \) iff \( I \models r \) for all \( r \in R \).

Following [27], the stable models (i.e., synonymously, the answer sets) of a PWC are obtained by means of an extension to the GL-reduct [18] that, instead of removing rules where some negative literals in the body are not modeled in a given set of atoms (candidate stable model) \( I \), it removes rules where the upper bounds of some weight constraints in the body are not satisfied. The upper bounds of constraints are removed and the lower bounds are rearranged in order to eliminate negative literals. Each rule \( r \) is then replaced by a set of rules each of them having as head one of the positive literals in \( H(r) \) which belongs to \( I \). In this manner, a positive PWC is obtained where the heads of rules are atoms (recall that an atom \( a \) is a shorthand notation for the weight constraint \((\{(a, 1)\}, 1, 1)\)). Finally, \( I \) is a stable model if it is the unique minimal model of this resulting program. Formally we have:

\[\text{For the sake of simplicity, in this paper we will deal with non-negative integer weights only. Generalizations involving negative values, as well as real numbers, are possible [27].}\]
3. Nested Weight Constraints

In this section we introduce an extension of ASP where weight constraints can be arbitrarily nested. In such an extension one can specify a collection of “internal” weight constraints, within an “external” weight constraint. The inner constraints represent conditions on the satisfiability of the outer constraint. Conversely, the external constraint affects the interpretation of internal weight literals and defines the local context where these literals have to be evaluated.

Definition 3.1. Let $S$ be a finite set of weight literals and let $l \leq u$ be non-negative integers ($u$ can be $\infty$). Then, nested weight constraints (NWCs) of depth $i$ are defined as follows:

- if $(S, l, u)$ is a weight constraint then $c = (S, \emptyset, l, u)$ is an NWC of depth 1 (i.e., $\text{depth}(c) = 1$).
- given a finite collection $N$ of NWCs, then $(S, N, l, u)$ is an NWC of depth $1 + \max_{c' \in N} \text{depth}(c')$.

A nested weight constraint is an NWC of any finite depth.

The definitions of rule and program are given in analogy with the case of PWCs. (Note in particular that, as for weight constraint, NWCs might occur in rule heads.) We extend to NWCs the notation introduced earlier and, moreover, for any given NWC $c = (S, N, l, u)$ we denote $N(c)$.

To the purposes of this paper, it is not restrictive to assume the finiteness of the Herbrand universe of the underlying language. Also, we will consider only programs with finite depth.

Before introducing a notion of satisfaction for NWCs, let us make some preliminary comments. Recall that the weight of a constraint $c$ w.r.t. a candidate model $I$ (cf., (1)) is determined by considering which positive (resp., negative) literals are satisfied (resp., not satisfied) by the set of atoms $I$. In presence of nesting the satisfaction of an NWC $(S, N, l, u)$ depends on the satisfaction of the NWCs in $N$. In turn, the satisfaction of each $c' \in N$ has to be evaluated within the context determined by $S$. Namely, $S$ acts as a filter on the literals of the candidate model and the weight of $c'$ is evaluated by ignoring the weights of all filtered out literals. Note that both satisfied and falsified atoms might be filtered out because of positive, resp. negative, literals occurring in $S$. Consequently, the context for an inner constraint can be described as a pair of disjoint sets of atoms, whose union is the set of all atoms that have not been filtered out by any of the outer contexts/constraints: the first set includes atoms that are true in the candidate model; the second set includes instead atoms that are false in the candidate model. Notice, moreover, that the union of these two sets is, in general, a proper subset of $S$. Hence, differently from the case of weight constraints, we need to explicitly distinguish between true and false atoms in the candidate model. Def. 3.2 generalizes (1) in defining the weight of an NWC w.r.t. a pair of sets of atoms.
Let $X, Y \subseteq B$ be two disjoint sets of atoms and $c = (S, N, l, u)$ an NWC. Then, we define the weight of the constraint $c$ (w.r.t. $X, Y$) as follows:

$$W(c, X, Y) = \sum_{(a,j) \in S \cap a \in X} j + \sum_{(-a,j) \in S \cap a \in Y} j$$

(2)

A last element is needed to specify the notion of satisfaction of an NWC $(S, N, l, u)$. As mentioned, the satisfaction of each $c' \in N$ is affected by literals occurring in $S$. We need to describe which literals are filtered out in consequence of the occurrence of a literal $l$ in $S$. I.e., we have to explain how we obtain sets $X$ and $Y$. In general, one might impose any equivalence relation $\approx$ on the set of all literals to induce a partition on the set of all literals. Then, the context of $c'$ is obtained by filtering out each literal $l'$ such that $l' \not\in S$ and $l' = l$ for any $l$ occurring in $S$. In what follows, we set a specific partition by considering two literals to be equivalent if and only if they share the same predicate symbol. If we take for instance a predicate $p$ and we assume that only weight literals $(p(t_1, \ldots, t_n), j_1)$ and $(p(q_1, \ldots, q_n), j_2)$ belong to $S$, then any atom $p(r_1, \ldots, r_n) \in B$, with $(r_1, \ldots, r_n) \neq (t_1, \ldots, t_n)$ and $(r_1, \ldots, r_n) \neq (q_1, \ldots, q_n)$, will be filtered away (i.e., $p(r_1, \ldots, r_n) \not\in X \cup Y$). Thus, to the extent of the evaluation of inner constraints in $N$, the domain of predicates (in the example, of predicate $p$) is restricted to the instances occurring in $S$, so that $S$ constitutes the context of the evaluation. The domain of predicates non occurring in $S$ remains unaffected.

Note that however any partition might in principle be used. Such a partition might be specific to the program and, moreover, it might be computed by part of the rules of the program itself. Considering this kind of generalizations goes beyond the purposes of this paper and might be subject of future work. Def. 3.3 introduces the notion of satisfaction for NWCs.

**Definition 3.3.** A pair of sets of atoms $X, Y$ satisfies the NWC $c = (S, N, l, u)$, iff the following two conditions hold. In such a case we write $(X, Y) \models c$.

- $l \leq W(c, X, Y) \leq u$;
- for all $c' \in N$ it holds that $(U, V) \models c'$, where

$$U = \{ a \in X \mid \pi(a) \notin \pi(S) \} \cup \{ a \in X \mid (a, j) \in S \}$$

$$V = \{ a \in Y \mid \pi(a) \notin \pi(S) \} \cup \{ a \in Y \mid (-a, j) \in S \}$$

(3)

Given a set of atoms $I$, we say that $I$ models an NWC $c$ and write $I \models c$, iff $(I, B \setminus I) \models c$. For a set $Q$ of NWCs we write $I \models Q$ if $I \models c$ for each $c \in Q$.

Note that, in absence of nesting, namely, for an NWC $c = (S, N, l, u)$ with $N = \emptyset$, we obtain the notion introduced earlier for weight constraints. As mentioned, if $N \neq \emptyset$, the satisfaction of each $c' \in N$ has to be evaluated within the context determined by $S$. According to the above definition, $c'$ is evaluated by considering only those atoms belonging to the subsets $U \subseteq X$ and $V \subseteq Y$ (recall that for the overall constraint $c$ we have $X = I$ and $Y = B \setminus X$ for a given set of atoms $I$). In this way, all weight literals in $c'$ having an atom in $B \setminus (U \cup V)$ are assumed to have null weight. In other words, in evaluating the weight of $c'$ we ignore the weights of all literals $\ell$ with $\pi(\ell) \in \pi(S)$ but not occurring in $S$. The same procedure is recursively applied for evaluating the weights of the constraints $c'' \in N(c')$, and so on.

As before, a set of atoms $I$ is a model of a rule $r$ (denoted by $I \models r$) if $I \models H(r)$ whenever $I \models B(r)$ holds. Given a program $P$, $I \models P$ iff $I \models r$ for all $r \in P$.

Below we adapt the notion of reduct (cf., Def.2.1) to deal with the nested constraints in $N$. 

---

*S. Costantini and A. Formisano/Nested Weight Constraints in ASP*
Definition 3.4. Given an NWC $c = (S, N, l, u)$ and a pair of disjoint sets of atoms $X, Y$, the **reduct** of $c$ w.r.t. $X, Y$ is defined as follows ($a$ denotes an atom):

$$c^{(X,Y)} = \{(a, j) | (a, j) \in S \} \cup \{(d(U,V)) | d \in N \}, \max(0, l - \sum(\neg a, j)_{a \in S \land a \in Y, j, \neg } \infty)$$

where $U$ and $V$ are obtained from $X$, $Y$ and $S$ as stated in Def. 3.3.

For a set $Q$ of NWCs we denote by $Q^{(X,Y)}$ the set \{ $c^{(X,Y)} | c \in Q \}$.

In analogy with the case of PWCs [24, 26], we define the reduct of a program as follows:

**Definition 3.5.** The **reduct** $P^I$, w.r.t. a set of atoms $I$, of a program $P$ with NWCs is defined as follows:

$$P^I = \{(a, b(r), I, B(r)) | r \in P, (a, j) \in Cl(\Omega(r)), a \in I, W(c, I, B \setminus I) \leq u(c) \text{ for all } c \in B(r) \}$$

Notice that each rule $r$ in $P^I$ has a head of the form $\{(a, 1), (1, 1)\}$, for some atom $a$. Moreover, no negative literals occur in the body of $r$. Similarly to the case of ordinary weight constraints, we introduce an operator $T_{P^I}$ defined as follows:

$$T_{P^I}(J) = \{ a | \exists r \in P^I, a = \Omega(r), J \models B(r) \}$$

To prove monotonicity of $T_{P^I}$ we first introduce a useful lemma. The main proposition will follow.

**Lemma 3.6.** Let $c = (S, N, l, u, \infty)$ be an NWC such that $S$ does not contain any negative literal. For each $i \in \{1, 2\}$ let $I_i, J_i$ be a pair of disjoint sets of atoms such that $I_1 \subseteq I_2$ and $J_1 \subseteq J_2$. Moreover, for each $i \in \{1, 2\}$, let $X_i, Y_i$ be a pair of disjoint sets of atoms such that $X_1 \subseteq X_2$ and $Y_2 \subseteq Y_1$, and $c_i = c^{(X_i, Y_i)}$. We have that if $(I_1, J_1) \models c_1$ then $(I_2, J_2) \models c_2$.

**Proof.** Observe that, for $i \in \{1, 2\}$, $c_i = \{(a, j) | (a, j) \in S \} \cup \{(d(U_i, V_i)) | d \in N \}$ where $U_i = \{ a \in X_i | \pi(a) \notin \pi(S) \}$ and $V_i = \{ a \in Y_i | \pi(a) \notin \pi(S) \}$. Thus $U_1 \subseteq U_2$ and $V_1 \subseteq V_2$.

**Proposition 3.7.** Let $P^I$ be the reduct of a program and let $J_1, J_2$ be two sets of atoms. If $J_1 \subseteq J_2$ then $T_{P^I}(J_1) \subseteq T_{P^I}(J_2)$.

**Proof.** Let $a \in T_{P^I}(J_1)$. There exists a rule $r \in P^I$ of the form $(a, b(r))$ such that $J_1 \models B(r)$. Hence, by Def. 3.3, for each NWC $c = (S, N, l, u, \infty) \in B(r)$, we have that $l \leq W(c, J_1, B \setminus J_1)$ and for each $c' \in N$, $(U_1, V_1) \models c'$, with $U_1 = \{ a | a \in J_1 \land \pi(a) \notin \pi(S) \}$ and $V_1 = \{ a | a \in J_1 \land \pi(a) \notin \pi(S) \}$.
If \( J_1 \subseteq J_2 \), then \( l \leq W(c, J_2, B \setminus J_2) \) (also because no negative literals are in \( S \)). Observe now that \( U_2 = \{ a \mid a \in J_2 \wedge \pi(a) \notin \pi(S) \} \cup \{ a \mid a \in J_2 \wedge (a, j) \in S \} \supseteq U_1 \) and \( V_2 = \{ a \mid a \in (B \setminus J_2) \wedge \pi(a) \notin \pi(S) \} \subseteq V_1 \). The fact that \( (U_2, V_2) \models c' \) holds follows, by Lemma 3.6, from the fact that \( (U_1, V_1) \models c' \) (for each \( c' \in N \)). This allows us to conclude that \( J_2 \models b(r) \), hence \( a \in T_{P^l}(J_2) \).

Given a program \( P \) and a set of atoms \( I \), by the previous result the operator \( T_{P^l} \) is monotonic and has a unique least fix-point which is obtainable by iterated applications of \( T_{P^l} \) starting from the empty set. Let us denote such a fix-point by \( T_{P^l}\uparrow \).

We are now able to introduce the following notion of stable model for programs with NWCs.

**Definition 3.8.** A set \( I \) of atoms is a stable model for a program with NWCs \( P \) iff \( I \models P \) and \( I = T_{P^l}\uparrow \).

## 4. Conditional Literals and the Use of Variables

Similarly to the approach of [24], in this section we adapt the treatment described so far to deal with **conditional literals**. A conditional literal has the form \( \ell \cdot s \) where \( \ell \) is a weight literal and \( s \) is a (possibly empty) set of atoms. The intended meaning is that the conjunction of the atoms in \( s \) constitutes a precondition for the satisfiability of \( \ell \). (Empty conditions, i.e., \( s = \emptyset \), are trivially satisfied. Conditional literals of the form \( \ell \cdot \emptyset \) correspond to weight literals as introduced earlier. We will often write \( \ell \) in place of \( \ell \cdot \emptyset \).

All the notions introduced in the previous section can be easily adapted to deal with conditional literals. In what follows we outline the main steps of such an adaptation. For the sake of readability, in doing this we will maintain the same notational conventions. The next definition is the counterpart of Def. 3.1:

**Definition 4.1.** Let \( S \) be a finite set of conditional literals and let \( l \leq u \) be non-negative integers (\( u \) can be \( \infty \)). Then, **nested weight constraints** (NWCs) of depth \( i \) are defined as follows:

- \( c = (S, \emptyset, l, u) \) an NWC of depth 1 (i.e., \( \text{depth}(c) = 1 \)).
- given a finite collection \( N \) of NWCs, then \( (S, N, l, u) \) is an NWC of depth \( 1 + \max_{c' \in N} \text{depth}(c') \).

A nested weight constraint is any NWC of any finite depth.

Rules and programs are defined in analogy with the case of PWCs. The notion of satisfaction for NWCs is slightly more complicated w.r.t. the one introduced in the previous section. This is so because the initial set of atoms (i.e., the candidate model, \( Z \) in the following) has to be considered in evaluating the preconditions of all conditional literals. Let \( Z, X, Y \subseteq B \) be sets of atoms such that \( X \subseteq Z \) and \( Y \subseteq (B \setminus Z) \). The weight of an NWC \( c = (S, N, l, u) \) w.r.t. \( Z, X, Y \) is defined as

\[
W(c, Z, X, Y) = \sum_{(a, j); s \in S \wedge a \in X \wedge s \subseteq Z} j + \sum_{(a, j); s \in S \wedge a \in Y \wedge s \subseteq Z} j
\]

We say that the sets of atoms \( Z, X, Y \) satisfy the NWC \( c = (S, N, l, u) \), and write \( (Z, X, Y) \models c \), if the following two conditions hold (with abuse of notation, we denote by \( \pi(S) \) the set \{\( \pi(\ell) \mid \ell \cdot s \in S \)\}):

- \( l \leq W(c, Z, X, Y) \leq u \);
- for all \( c' \in N \) it holds that \( (Z, U, V) \models c' \), where

\[
U = \{ a \in X \mid \pi(a) \notin \pi(S) \} \cup \{ a \in X \mid (a, j) : s \in S \wedge s \subseteq Z \}
\]

\[
V = \{ a \in Y \mid \pi(a) \notin \pi(S) \} \cup \{ a \in Y \mid (a, j) : s \in S \wedge s \subseteq Z \}
\]

Given a set \( I \) of atoms, we say that \( I \) models an NWC \( c \) and write \( I \models c \), iff \( (I, I, B \setminus I) \models c \). For a set \( Q \) of NWCs we write \( I \models Q \) iff \( I \models c \) for each \( c \in Q \).
The satisfaction of each \( c' \in N \) has to be evaluated within the context determined by \( S \). The same recursive scheme outlined in the previous section applies here in evaluating the satisfiability of NWCs. As before, a set \( I \) of atoms is a model of a rule \( r \) (denoted by \( I \models r \)) if \( r \) whenever \( I \models B(r) \) holds. Given a program \( P \), \( I \models P \) iff \( I \models r \) for all \( r \in P \).

In defining the notion of reduct we take into account all preconditions of conditional literals. Let \( Z, X, Y \) be sets of atoms. The reduct of an NWC \( c = (S, N, l, u) \), w.r.t. \( Z, X, Y \), is defined as follows:

\[
e_c^{(Z,X,Y)} = \left( \{ (a,j): s \mid (a,j): s \in S \}, \{ d^{(Z,U,V)} \mid d \in N \}, \max(0, l - \sum_{(\neg a,j): a \in S \land a \in Y \land s \subseteq Z} \), \infty \right)
\]

where \( U \) and \( V \) are obtained from \( X, Y, Z \), and \( S \) as explained earlier.

In analogy with the cases of plain [24, Def. 2] and nested (previous section) weight constraints, given a program \( P \) and a set of atoms \( I \), the reduct \( P^I \) is defined as follows:

\[
P^I = \left\{ (a, b(I, r, s)) \mid r \in P, \{ a,j : s \in Cl(H(r)) \}, \{ a \cup s \subseteq I \}, W(c, I, I, B \setminus I) \leq u(c) \text{ for all } c \in B(r) \right\}
\]

where \( b(I, r, s) \) denotes the set

\[
b(I, r, s) = \{ e^{(l,B,I)} \mid e \in B(r) \} \cup \{ (\emptyset, 0, \infty) \mid b \in s \} \cup \bigcup_{e \in B(r)} \{ (\emptyset, 0, \infty), (b, j) : r \in Cl(c), a \in r \text{ s.t. } b \notin I, r \subseteq I \}
\]

The definition of the reduct \( P^I \) of a program is slightly more involved than the analogous definition of previous section. This is so because each negative literal \( (\neg b,j): r \), with \( b \notin I \), occurring in an NWC of the body of a rule \( r \), will give its contribution to the weight of the NWC only if the precondition \( r \) holds in \( I \). This requirement must be reflected in the program \( P^I \) by adding the set shown in previous formula.

In this manner, the body of the resulting rule will be falsified whenever any of such preconditions is false.

Now, we can define an operator \( T_{pr} \) exactly as done in (5). Such an operator is monotonic (an analogous to Proposition 3.7 can be stated) and has a unique least fix-point. Def. 3.8 can be properly generalized to the case of NWCs with conditional literals:

**Definition 4.2.** Given a program \( P \) with NWCs involving conditional literals, a set of atoms \( I \) is a stable model for \( P \) iff \( I \models P \) and \( I = T_{pr} I \).

**Programs with variables.** Variables can be used to denote collections of weight literals. This is done by admitting non-ground conditional weight literals \( \ell : \varphi \) where \( \ell \) has, in general, the form \((p(X_1, \ldots, X_n), j)\) (or the form \((\neg p(X_1, \ldots, X_n), j)\)) and each \( X_i \) is a variable (for \( n \geq 0 \)). Similarly, \( \varphi \) is a set of not necessarily ground atoms. Let \( \text{var}(\varphi) = \{ X_1, \ldots, X_n, Y_1, \ldots, Y_m \} \) be the set of all variables occurring in \( \varphi \) (for \( n, m \geq 0 \)). The variables \( X_1, \ldots, X_n \) are said to be local to the literal and \( Y_1, \ldots, Y_m \) are said to be global. Non-ground NWCs, rules, and programs, are then defined as usual.

Given a program, it is not restrictive to impose that each local variable occurs in a single conditional literal. We will make this assumption in what follows.

Considering a rule with non-ground NWC, all its global variables should be intended as being universally quantified. The instantiation of a rule is defined as the set of the ground rules each of them obtainable, first, by grounding all global variables (i.e., by uniformly substituting them by ground terms from the Herbrand universe of the underlying language) and then by replacing each non-ground conditional weight literal with the collection of all its ground instances that are obtainable by grounding.

\footnote{Note that, in concrete encodings, constants are admissible in place of (some of) the \( X_i \)s. For simplicity, and without loss of generality, we assume that each \( X_i \) is a variable.}
the local variables. Notice that in a literal such as \((a, j)\) (or \((-a, j)\)), we admit \(j\) to be a (global) variable. In this manner, each instantiation of this literal may have a different weight, determined through the grounding process. Analogously, the lower and upper bounds of an NWC can be expressed using variables, provided that the grounding process suitably instantiates them to non-negative integers. (In what follows we will adopt this option.) The instantiation of a program \(P\) is defined as the set of all instantiations of rules in \(P\). Stable models of programs involving variables are easily defined as follows:

**Definition 4.3.** Given a (non-ground) program \(P\), a set of ground atoms \(I\) is a stable model for \(P\) iff it is a stable model for the instantiation of \(P\).

5. Case Studies

To motivate the introduction of NWC in ASP, we resort to two examples where the new construct can be useful in order to obtain a more clear and elaboration-tolerant knowledge representation.

In describing a concrete encoding of our examples, we resort to the lparse-like notation for programs, rules, and literals. As before, we denote by \(p\) an NWC of the form \(\{(p), \emptyset, 1, 1\}\). Also, we denote a weight constraint \(\{(a_1, w_{a_1}), \ldots, (a_n, w_{a_n}), (\neg b_1, w_{b_1}), \ldots, (\neg b_m, w_{b_m})\}, l, u\) as \(l[a_1=w_{a_1}, \ldots, a_n=w_{a_n}, \neg b_1=w_{b_1}, \ldots, \neg b_m=w_{b_m}] u\) and, similarly, an NWC \(\{(a_1, w_{a_1}), \ldots, (a_n, w_{a_n}), (\neg b_1, w_{b_1}), \ldots, (\neg b_m, w_{b_m})\}, \{W_1, \ldots, W_k\}, l, u\) as \(l[a_1=w_{a_1}, \ldots, a_n=w_{a_n}, \neg b_1=w_{b_1}, \ldots, \neg b_m=w_{b_m} | W_1, \ldots, W_k] u\) in both cases, we omit \(u\) whenever \(u = \infty\). For the special case of cardinality constraints, i.e., when \(w_i = 1\) for all \(i\), we adopt the shorthand notation \(l\{a_1, \ldots, a_n, \neg b_1, \ldots, \neg b_m\} u\).

Conditional literals of the form \((a, j)\):\{\(b_1, \ldots, b_k\}\} will be denoted as \(a:b_1, \ldots, b_k=j\). Moreover, in expressing weights and bounds of constraints we may use variables (intended to be suitably instantiated by the grounding phase). Finally, we denote a program rule as \(W_0 := W_1, \ldots, W_n\), where the \(W_i\)s are (nested) weight constraints (for \(n \geq 0\)).

The first example is freely inspired by the Italian computer science undergraduate program. In order to get a bachelor degree in computer science, a student is required to obtain 180 credits. Most of them must be obtained by attending courses and passing the corresponding exams. The remaining ones can be obtained by means of internships and a thesis. There is a certain flexibility, so usually the number of credits that should be obtained from courses is allowed to vary within a range (say between 153 and 171, in the following encoding; actual ranges vary among different universities and tracks). There are different possible choices for the courses to attend, so students are required to present what is called a “plan of studies”, that must be approved by a committee. Some courses must be taken at a certain year, for others there is some flexibility. For simplicity, we assume that the latter can be taken at any year and we neglect constraints related to the order in which certain courses should be taken. Basically, the above (as described up to now) might be summarized by the following rule that characterizes possible plans of studies. (The atom \(in_ps(c, j)\) means that the course \(c\) is inserted into the plan of studies, at year \(j\).)

\[
\begin{align*}
\text{Min} & \left[ \text{in}_p\text{s}(X,Y) : \text{course}(X,W), \text{course}_\text{year}(X,Y) = W \right] \text{Max} := \text{credits}_\text{bounds}(\text{Min}, \text{Max}).
\end{align*}
\]

This knowledge base describes a possible problem instance:

\[
\begin{align*}
\text{year}(1..3). & \quad \text{credits}_\text{bounds}(153,171). \\
\text{course}_\text{desc}(\text{programming}, \text{comp}_\text{sci}, 12). & \quad \text{course}_\text{desc}(\text{databases}, \text{comp}_\text{sci}, 12). \\
\text{course}_\text{desc}(\text{algorithms}, \text{comp}_\text{sci}, 12). & \quad \text{course}_\text{desc}(\text{computer}_\text{arch}, \text{comp}_\text{sci}, 6). \\
\text{course}_\text{desc}(\text{theoretical}_\text{cs}, \text{comp}_\text{sci}, 6). & \quad \text{course}_\text{desc}(\text{sw}_\text{engineering}, \text{comp}_\text{sci}, 6). \\
& \ldots
\end{align*}
\]
where each fact course_desc(c,a,n) specifies that the course c belongs to the area a (see below) and corresponds to an amount of n credits. Moreover, we assume the presence of facts of the form course_year(c,y) specifying, for each year y, the admissible courses c for that year.

Clearly, this simple encoding does not model all aspects of the problem at hand. For instance, an aspect which is not represented is that a plan of study cannot include the same course several times. This can be imposed by adding this NWC (actually a cardinality constraint):

0 \{ in_ps(X,Y): in_ps(X,Y1), neq(Y,Y1) \} 0.

In our case-study, it is always the case that some mandatory courses must be situated at a certain course year. To model this requirement we add this NWC to the initial rule:

0 \{ in_ps(X,Y): mandatory(X,Y1), neq(Y,Y1) \} 0.

For courses that must be included in the solution, but can be situated at any year, we add this extra rule:

1\{ in_ps(C,Y): year(Y) \}1 :- mandatory_course(C).

The specific instance might specify, for example, this piece of knowledge:

mandatory(programming, 1). mandatory(computer_arch, 1). mandatory(algorithms, 2).
mandatory(theoretical_cs, 3). mandatory_course(databases).

Finally, to avoid a student giving too many exams, there is a statement that enforces at least a minimum number of courses of the first two years to weigh 12 credits each. Moreover, courses are allowed to belong to certain scientific areas, namely computer science, mathematics, physics, etc. However, there are directions stating that every subject should contribute to the plan of studies for a quota ranging between a minimum and a maximum number of credits. The next NWCs specify both the number of the 12 credit courses in the first years and ranges of credits for the different areas. Here, the constants comp_sci, maths, ..., identify (through the facts course_desc listed earlier), the area of each course:

Min12 \{ in_ps(X,Y): course(X,W), leq(Y,2), eq(W,12) \} 0
L1 \{ in_ps(X,Y): course_desc(X,comp_sci,W), course_year(X,Y)=W \} 1
...Ln \{ in_ps(X,Y): course_desc(X,maths,W), course_year(X,Y)=W \} 1

where the variables Min12, L1, U1, ..., Ln, and Un, (to be instantiated through atoms in the rule body) express the bounds on the minimum number of courses worth 12 credits in the first two years, and the minimum/maximum amounts of credits in the different areas.

The full encoding of our sample problem by means of NWCs is as follows (to be joined with a specific instance specifying, together with the pieces of knowledge seen earlier, the predicate area_bounds):

Min \{ in_ps(X,Y): course(X,W), course_year(X,Y)=W | 0 \{ in_ps(X,Y): in_ps(X,Y1), neq(Y,Y1) \} 0, 0 \{ in_ps(X,Y): mandatory(X,Y1), neq(Y,Y1) \} 0, Min12 \{ in_ps(X,Y): course(X,W), leq(Y,2), eq(W,12) \}, L1 \{ in_ps(X,Y): course_desc(X,comp_sci,W), course_year(X,Y)=W \} 1, ...Ln \{ in_ps(X,Y): course_desc(X,maths,W), course_year(X,Y)=W \} 1\} Max :- credits_bounds(Min,Max), min_12(Min12), area_bounds(comp_sci,L1,U1), ... area_bounds(maths,Ln,Un).

1\{ in_ps(C,Y): year(Y) \}1 :- mandatory_course(C).
The second case-study involves a doubly-nested weight constraint. It specifies a (simplified) diet planner for a child with a form of diabetes. As a first step, the list of available food is provided, e.g.:

food(fish). food(low_fat_cheese). ...

Then, food is sorted according to its nutritional contents:

% high protein food, carbohydrates, and fats:
protein(yogurt). protein(milk). ...
carbo(boiled_beans). carbo(bread). carbo(pasta). ... fat(olive_oil). ...
% food providing calcium or other minerals and vitamins:
in_calcium(yogurt). ... in_mi_vits(raw_tomato). ...

Then, food is described in terms of how many calories a basic portion (expressed in grams) provides.

% Food, Grams and Calories
descr(bread,40,120). descr(pasta,60,150). ...

Finally, it is stated which food should occur in each meal either necessarily (mandatory food) or optionally (optional food), where some kinds of food are particularly recommended. Here the meaning of optional food is not that it can be skipped, but that one can take any of the available choices, according to requirements on nutritional contents (see below).

mandatory(pasta). mandatory(bread). optional(low_fat_meat). recommended(milk). ...

The following ASP weight and cardinality constraints define a simple diet planner stating that: (i) each meal must include exactly one portion of mandatory food; (ii) from 300 to 600 calories may come from optional food; (iii) at least 100 calories should be provided by food providing calcium, and another 100 calories by food providing minerals and vitamins; (iv) one or two portions of optional carbohydrates are to be included in a meal, as well as (v) one among meat or ham.

1 { in_meal(X,Y) : food(X) : descr(X,Y,W) } 1 :- mandatory(X).
300 [ in_meal(X,Y) : food(X) : descr(X,Y,W) : optional(X) = W | 100 { in_meal(X,Y) : descr(X,Y,W), in_calcium(X) = W | 1 { in_meal(X,Y) : recommended(X)}}, 100 { in_meal(X,Y) : in_mi_vits(X) = Y | 1 { in_meal(X,Y) : carbo(X) } 2 }, 1 { in_meal(low_fat_meat,90), in_meal(low_fat_ham,70) } 1 ] 600.

Assume we want to fulfill the requirement that food providing specified amounts of calcium, minerals, and vitamins have to be chosen among those rated as “optional”. Additionally, we want to take recommended food into account. The following formulation exploits nested weight constraints. It states (as in plain ASP) that, in addition to mandatory food, an amount between 300 and 600 calories may come from optional food; (iii) at least 100 calories should be provided by food providing calcium, and another 100 calories by food providing minerals and vitamins. Also, among those one or two portions of optional carbohydrates are to be included in a meal, as well as (v) one among meat or ham.

1 { in_meal(X,Y) : food(X), descr(X,Y,W) } 1 :- mandatory(X).
300 [ in_meal(X,Y) : food(X), descr(X,Y,W), optional(X)=W | 100 { in_meal(X,Y) : descr(X,Y,W), in_calcium(X)=W | 1 { in_meal(X,Y) : recommended(X)}}, 100 { in_meal(X,Y) : in_mi_vits(X)=Y | 1 { in_meal(X,Y) : carbo(X) } 2 }, 1 { in_meal(low_fat_meat,90), in_meal(low_fat_ham,70) } 1 ] 600.
This formulation provides a representation of knowledge that models the problem requirements directly instead of indirectly. In general, we believe that NWCs may allow a programmer to cope more easily with aspects that can be relevant in a number of applications. Our case-studies, in fact, are simple examples of scheduling/allocation problems: this kind of problems are an important realm of application of ASP (cf., e.g., [3, 17, 19, 29] and the references therein). We believe therefore that many kinds of ASP applications might profit from programming constructs that allow for some degree of nesting. Thus, we deemed it appropriate to introduce some kind of contextual constructs.

6. Complexity of Nested Weight Constraints

In [27] it is proved that the introduction of weight constraints does not affect the complexity of ASP. That is, for instance, the complexity of the problem of checking whether a program has a stable model does not depend on the presence of weight constraints. Here, we are more generally concerned with checking whether a program with NWCs admits stable models. In particular, we can formulate the following result, stating that the introduction of NWCs still does not affect the complexity of the resulting extended ASP formalism. This can be seen on one hand as an advantage of NWCs, as they provide a programmer with a useful extra-feature for knowledge representation without an actual computational burden. On the other hand however, it might be seen as a limitation as one might question the usefulness of introducing a construct not adding extra-expressivity. In the next section we will discuss this issue and contextualize our approach with respect to related work. Below we state the formal result.

Proposition 6.1. Deciding whether a ground NWC program admits stable models is NP-complete.

PROOF. To decide whether a ground NWC program admits a stable model is NP-hard. This follows from the NP-completeness of ASP with weight constraints in absence of nesting [27]. As regards inclusion in NP, this can be verified by showing that, given a set $M$ of atoms, it can be checked in polynomial time whether $M$ is a stable model of $P$. To do this we have to show that: (a) given a rule $r \in P$, checking if $M \models r$ takes polynomial time w.r.t. the size of $P$; (b) the reduct $P_M$ of the program $P$ can be computed in polynomial time (and has polynomial size) w.r.t. the size of $P$; (c) $T_{PM}$ can be computed in polynomial time w.r.t. the size of $P$. Given a set $R$ of weight literals, let us denote its size by $z(R) = |R|$. Moreover, given a weight constraint $c = (S, N, l, u)$, let $z(c) = |S| + \sum_{c' \in N} z(c')$. Similarly, the size of a rule is defined to be the sum of the sizes of the weight constraints occurring in it; the size of a program is the sum of the sizes of its rules.

As regards (a), observe that checking whether $M \models c$ for an NWC $c$ involves the evaluation of the weight of $c$ (cf., (2)) and the computation of the sets $U, V$ (cf., (3)). Both these tasks can be completed in linear time. In general, evaluating if $(X, Y) \models c$ takes linear time w.r.t. $z(c)$. This fact can be shown by induction on the depth of $c = (S, N, l, u)$. Indeed, if $N = \emptyset$ the result immediately follows. The inductive step can be accomplished by observing that both the evaluation of $W(c, X, Y)$ and the computation of the sets $U, V$ take linear time w.r.t. $|S|$. Moreover, by inductive hypothesis, for all $c_i = (S_i, N_i, l_i, u_i) \in N$, evaluating whether $(U, V) \models c_i$ holds, takes linear time w.r.t. $z(S_i)$. Hence, checking if $(X, Y) \models c$ can be completed in time proportional to $z(c) = z(S) + \sum_{i \in N} z(S_i)$. Concerning (b), for each rule $r$ in $P$ a linear number of rules is introduced in $P_M$ (cf., (4)). Moreover, for each NWC $c$ occurring in $r$ the computation of the reduct of $c$ takes linear time w.r.t. $z(c)$. This claim can be verified by proceeding
by induction on the depth of $c$, similarly to what done earlier in the proof of point (a). Finally, (c) can be proved by observing that the computation of the set $I_2 = T_{PM}(I_1)$, for a given $I_1$, can proceed by processing, one-by-one, each unsatisfied rule whose head is not in $I_2$, and checking the satisfaction of its body. By (a) and (b) we can conclude that $T_{PM}^{\uparrow}$ can be computed in polynomial time.

From this result, it follows that NWCs might be rephrased in plain ASP. As shown in [27, 16], for weight constraints this can be done at the expense of introducing a (polynomial, but not negligible) number of new atoms and rules. Moreover, except for cardinality constraints, the translation is quite involved. Therefore, it turns out that weight constraints are a quite substantial programming construct, rather than simple syntactic sugar. This is of course true also for NWCs. Notice that a represent of NWCs in plain ASP is far from easy to understand. Outlining a translation into plain ASP and evaluating the necessary number of additional atoms and rules is a subject of future work. We give here just some hints about how to compile an NWC into weight constraints. Consider, for instance, the following simplified case of an NWC devoid of negative literals and occurring in the body of a rule (using the concrete syntax):

```
head :- ..., l [p(1)=10, p(2)=20, p(3)=30 | n [p(1)=1, p(3)=3, p(5)=5, q(4)=4] m ] u, ...
```

the NWC in can be replaced as follows (where $p'$ is a fresh predicate symbol):

```
head :- ..., l [p(1)=10, p(2)=20, p(3)=30] u, n [p'(1)=1, p'(3)=3, q(4)=4] m, ...
```

while these new rules define the new predicate:

```
p'(1) :- p(1). p'(3) :- p(3).
```

In general, the translation is applied to all NWCs of a rule, by proceeding inwards and introducing, whenever needed, fresh symbols not occurring elsewhere in the program. After completing the translation of all rules, the obtained PWC can be translated into pure ASP by applying standard techniques [27, 16].

7. Concluding Remarks

The weight constraint construct is widely used in ASP practical programming. In this paper, we introduced an extension involving arbitrary nesting of weight constraints and provided a semantics for the enhanced framework. The extension does not affect the complexity of ASP.

Weight Constraints and aggregates (as introduced in the DLV system [20] and in the FO system [12]) are important enhancements to Logic Programming and Answer Set Programming. As a matter of fact, they have been usefully exploited in a number of applications. Their semantics and complexity have been extensively studied, e.g., in [1, 15, 25]. It turns out that reasoning with aggregates without restrictions may easily increase the complexity of the computation. However, many interesting classes of programs exist where one is able to represent knowledge via aggregates while retaining the same complexity of plain ASP. In this perspective, our result about complexity of NWCs is in line with related research and should be considered as an advantage. It is important also to notice that we have chosen to introduce nesting only into weight constraints for the sake of simplicity. The introduction of forms of nesting into aggregates is certainly possible and represents a theme for future investigation.

The reason why weight constraints and aggregates are not to be considered simply as syntactic sugar is that, from the perspective of Knowledge Representation, they enable more concise, expressive (from the “practical” point of view), readable, modifiable and, in sum, elaboration-tolerant encoding of commonsense knowledge. We believe that NWCs provide a further element in this direction, as we illustrated by means of non-trivial examples.
It may be observed that all the above-mentioned constructs can be considered to be special cases of modules: in fact, they are units which are clearly recognizable within a program and can be reused elsewhere. Their input/output interface is somewhat implicit, but values for variables internal to these constructs can be provided by other parts of the program. Therefore, in perspective, an integration with explicit forms of modularization should be envisaged.

Various aspects related to NWCs are still to be investigated. First of all, the proposed construct has not been implemented yet and no full translation into plain ASP has been devised (though, by Proposition 6.1, such a translation necessarily exists). When an implementation will be available, practical use will help us exploring the feasibility of further extensions and generalizations. For instance, the enrichment of weight constraints by means of complex/conditional preferences might be considered. In particular, the present work could be integrated with the approach to preference handling devised in [5, 10] and extended to weight constraints in [7, 8]. (In the resulting setting, referring to the first of our examples, one might enrich the formulation with student’s preferences, by stating which kind of courses are preferred and in which conditions.)

We will have to explore both the usefulness in practical applications of nested-constraints, as well as their feasibility in cases where both negative weights and circular definitions are admitted.

From the formal point of view, an interesting research theme consists in the extension of the concept of strong equivalence to NWC programs. Strong equivalence [22, 30, 9] in fact, as widely recognized, provides an important conceptual tool for program simplification, transformation, and optimization. In the case of NWC programs, the form of locality implicitly present in NWCs might have interesting consequences. A further issue for future research regards the relation between NWC and (nested) aggregates.

Acknowledgements. John McCarthy, whose definition of “Elaboration Tolerance” is mentioned in the Introduction, passed away on October 24, 2011 after a life dedicated to science, where he reached exceptional achievements. This paper is dedicated to him. One of his last famous sentences: “those who don’t speak math are doomed to speak nonsense”.

We would like to thank the anonymous reviewers for their comments.

References


