Intelligence Analysis Using Quantitative Preferences

Davy Van Nieuwenborgh[⋆], Stijn Heymans, and Dirk Vermeir[⋆⋆]

Dept. of Computer Science
Vrije Universiteit Brussel, VUB
Pleinlaan 2, B1050 Brussels, Belgium
{dvnieuwe,sheymans,dvermeir}@vub.ac.be

Abstract. The extended answer set semantics for simple logic programs, i.e. programs with only classical negation, allows for the defeat of rules to resolve contradictions. In addition, a partial order relation on the program's rules can be used to deduce a preference relation on its extended answer sets. In this paper, we propose a "quantitative" preference relation that associates a weight with each rule in a program. Intuitively, these weights define the "cost" of defeating a rule. An extended answer set is preferred if it minimizes the sum of the weights of its defeated rules. We characterize the expressiveness of the resulting semantics and show how the semantics can be conveniently extended to sequences of weight preferences, without increasing the expressiveness. We illustrate an application of the approach by showing how it can elegantly express largest common subgraph and subgraph isomorphic approximation problems, a concept often used in intelligence analysis to find similarities or specific regions of interest in large graphs of observed activity.

1 Introduction

Over the last decade a lot of research has been done on declarative programming using the answer set semantics [10, 2, 18], a generalization of the stable model semantics [8]. In answer set programming, one uses a logic program to modularly describe the requirements that must be fulfilled by the solutions to a particular problem, i.e. the answer sets of the program correspond to the intended solutions of the problem. One of the possible problems in answer set programming is the absence of any solutions in case of inconsistent programs. To remedy this, the authors proposed [16] the *extended answer set semantics* which allows for the *defeat* of problematic rules. E.g., the rules $a \leftarrow$, $b \leftarrow$ and $\neg a \leftarrow b$ are clearly inconsistent and have no classical answer sets, while both $\{a, b\}$ and $\{\neg a, b\}$ will be recognized as extended answer sets. Intuitively, $\neg a \leftarrow b$ is defeated by $a \leftarrow$ in $\{a, b\}$, while $\neg a \leftarrow b$ defeats $a \leftarrow$ in $\{\neg a, b\}$.

Within the context of inconsistent programs, it is natural to have some kind of preference relation that is used to prefer certain extended answer sets above others. In [16],

[⋆] Supported by the FWO

[⋆⋆] This work was partially funded by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2001-37004 WASP project

a “qualitative” preference semantics is proposed, using a preference relation on rules, to induce a partial ordering on the extended answer sets of a program.

As an alternative, this paper considers a “quantitative” preference relation for the extended answer set semantics on simple programs, i.e. programs containing only classical negation. We assign each rule in a program a (nonnegative) weight, representing the cost associated with defeating the rule. Solutions for these weighted programs, called *weighted answer sets*, are those extended answer sets that minimize the sum of the weights of defeated rules.

The resulting semantics turns out to be more expressive than classical answer set programming, even in the absence of negation as failure. We demonstrate that e.g. the membership problem is complete for the second level of the deterministic class of the polynomial hierarchy, i.e. Δ_2^P-complete.

In some situations more than one actor is involved in the process of finding a solution to a particular problem. Quite often we have a sequence of decision makers, where each one sorts out the best solutions according to her preferences among the solutions that are preferred by the previous one in the sequence. Intuitively, the solutions that are still preferred by the last decision maker in the sequence are the ones that are acceptable by all parties. E.g., in a job selection procedure, the secretary will only keep the applicants that passed all the tests. Secondly, the head of the department will prefer people that have better marks on their math tests, and among those, the management of the firm will select those with a better psychological profile.

Such hierarchies of individual weight preferences are supported by *weight sequence programs*, where each rule in a program is equipped with a sequence $\langle w_i \rangle_{i=1,\ldots,n}$ of weights corresponding to the cost each decision maker associates with defeating this rule (w_i has a higher priority than w_{i+1}). Semantically, weighted answer sets for such programs will be obtained from first finding the weighted answer sets w.r.t. the weights of the first decision maker, i.e. the weights w_1, and among those finding the ones that are minimal w.r.t. the weights of the second decision maker, i.e. the weights w_2, etc. Regarding the complexity, it turns out that such sequences of weights do not result in any additional expressiveness of the formalism, nevertheless allowing to express certain problems more intuitively.

The proposed semantics has applications in several areas where quantitative preferences are useful. E.g., in the area of subgraph isomorphism algorithms [14] it is useful, in case of absence of an exact match of the pattern graph in the larger graph, to search for *subgraph isomorphic approximations* (SIA for short) of the larger graph that are minimal in some sense, i.e. searching for a “minimal” set of items to add to the larger graph such that the pattern occurs in it. We show how the solutions of such SIA problems correspond with the weighted answer sets of a weighted program that can be constructed out of the given instance graphs. Applications of SIA can be found in the area of intelligence analysis [9, 4], where it is common to search for a pattern of interest in a large attributed relational graph [9] (ARG for short). An ARG is a normal graph where nodes and edges can carry additional attributes e.g. denoting relationships. In intelligence analysis, ARGs are used to model observed activity in the world under consideration. We show how the translation of the SIA problem for graphs into weighted programs can be intuitively adapted to the setting of ARGs, thus providing a useful tool for intelligence analysis. A similar approach can be applied for finding the largest common subgraphs between two ARGs.

The remainder of this paper is organized as follows: Section 2 introduces weighted programs and the corresponding weighted answer set semantics, together with a characterization of the expressiveness. Section 3 formalizes weight sequence programs and we show that these systems do not have additional expressiveness in comparison to normal weighted programs. In Section 4, we introduce the problem of largest common subgraphs, as well as subgraph isomorphic approximations in graph theory, and show how weighted programs can be conveniently used to compute them. Section 5 discusses a generalization of these graphs in the area of attributed relational graphs. Finally, we conclude in Section 6. Due to space restrictions, proofs have been omitted.[1]

2 Weighted Programs

We use the following basic definitions and notation. A *literal* is an *atom* a or a negated atom $\neg a$. For a set of literals X, $\neg X$ denotes $\{\neg a \mid a \in X\}$ where $\neg\neg a = a$. X is *consistent* if $X \cap \neg X = \emptyset$. An *interpretation* I is a consistent set of literals. A *simple rule* r is of the form $a \leftarrow \beta$ with $\{a\} \cup \beta$ a finite set of literals[2]. The rule r is *satisfied* by I, denoted $I \models r$, if $a \in I$ whenever $\beta \subseteq I$, i.e. if r is *applicable* ($\beta \subseteq I$), then it must be *applied* ($a \in I$).

A countable set of simple rules is called a *simple logic program* (SLP). The *Herbrand base* $\mathcal{B}_P$ of a SLP P contains all atoms appearing in P. For a SLP P and an interpretation I we say that a rule $a \leftarrow \beta \in P$ is *defeated* w.r.t. I iff there exists an applied *competing rule* $\neg a \leftarrow \beta' \in P$. Furthermore, we use $P_I \subseteq P$ to denote the *reduct* of P w.r.t. I, i.e. $P_I = \{r \in P \mid I \models r\}$, the set of rules satisfied by I.

An interpretation I is called a model of a SLP P if $P_I = P$, i.e. I satisfies all rules in P. If there is no model J of P such that $J \subset I$, I is a minimal model or *answer set* of P. An *extended answer set* for P is any interpretation I such that I is an answer set of P_I and each unsatisfied rule in $P \setminus P_I$ is defeated.

Example 1. Consider the following SLP P about diabetes.

$$\begin{array}{ccc} \mathit{hypoglycemia} \leftarrow & \mathit{diabetes} \leftarrow & \mathit{sugar} \leftarrow \mathit{hypoglycemia} \\ \neg \mathit{sugar} \leftarrow \mathit{diabetes} & \mathit{cola\_light} \leftarrow \neg \mathit{sugar} & \mathit{cola} \leftarrow \mathit{sugar} \end{array}$$

Clearly, while this program has no traditional answer sets, it has, however, two extended answer sets $I = \{\mathit{diabetes}, \mathit{hypoglycemia}, \mathit{sugar}, \mathit{cola}\}$ and $J = \{\mathit{diabetes}, \mathit{hypoglycemia}, \neg \mathit{sugar}, \mathit{cola\_light}\}$.

The extended answer sets of a program are not always equally preferred. E.g., in the above example, when low on sugar ($\mathit{hypoglycemia}$), one would prefer drinking cola, rather than taking no sugar at all ($\neg \mathit{sugar}$). So, defeating the rule $\mathit{sugar} \leftarrow \mathit{hypoglycemia}$ is "worse" than defeating the rule $\neg \mathit{sugar} \leftarrow \mathit{diabetes}$. Therefore, we equip the rules in simple programs with a weight representing the "penalty" involved

[1] They are available in http://tinf2.vub.ac.be/˜dvnieuwe/graphasptech.ps

[2] As usual, we assume that programs have already been grounded.

when defeating the rule. Naturally, extended answer sets that minimize the total penalty of a program are to be preferred over others.

Definition 1. *A **simple weight rule** is a rule r of the form $a \leftarrow \beta\langle w\rangle$, where $\{a\} \cup \beta$ is a finite set of literals and w is an associated weight value, i.e. a non-negative integer. We use $w(r)$ to denote the weight of r. A countable set of such simple weight rules is a **simple weight program** (SWP). The **extended answer sets** of a SWP P coincide with the extended answer sets of the SLP P' obtained from P by removing the weights from the rules.*

The program from Example 1 can be extended to a SWP containing a larger "penalty" weight for the hypoglycemia rules, i.e. the program:

$$\begin{array}{ccc} \mathit{hypoglycemia} \leftarrow \langle 0\rangle & \mathit{diabetes} \leftarrow \langle 0\rangle & \mathit{sugar} \leftarrow \mathit{hypoglycemia}\langle 1\rangle \\ \neg \mathit{sugar} \leftarrow \mathit{diabetes}\langle 0\rangle & \mathit{cola\_light} \leftarrow \neg \mathit{sugar}\langle 0\rangle & \mathit{cola} \leftarrow \mathit{sugar}\langle 0\rangle \end{array}$$

This program still has I and J as its extended answer sets, but intuitively I is better than J as it satisfies the rule with weight 1 while J does not, which we formalize in the following definition.

Definition 2. *The **penalty** of an extended answer set S w.r.t. a SWP P, is defined by $\Phi_P(S) = \sum_{r \in P \setminus P_S} w(r)$, i.e. the sum of the weights of all defeated rules in P w.r.t. S.*

*For two extended answer sets S_1 and S_2 of P, we define $S_1 \preceq S_2$ iff $\Phi_P(S_1) \leq \Phi_P(S_2)$. A **weighted answer set** of P is an extended answer set of P that is minimal w.r.t. $\prec$ ($a \prec b$ iff $a \preceq b$ and not $b \preceq a$) among the set of all extended answer sets of P. A weighted answer set S of P with $\Phi_P(S) = 0$ is called a **proper weighted answer set**.*

Intuitively, weighted answer sets are those solutions that minimize the penalties incurred by defeating rules. For the weighted version of the program from Example 1 one obtains that $\Phi_P(I) = 0$ and $\Phi_P(J) = 1$ such that $I \prec J$, which corresponds with our intuition.

While the previous example uses only two different weight values, the following example shows that one can use the proposed semantics to represent complex relations between defeated rules.

Example 2. Consider a company that wants to hire an employee. To get hired, you have to do some tests and based on these results the company decides.

$$\begin{array}{ccccc} \mathit{math} \leftarrow \langle 0\rangle & \mathit{lang} \leftarrow \langle 0\rangle & \mathit{psych} \leftarrow \langle 0\rangle & \mathit{prac} \leftarrow \langle 0\rangle & \mathit{phys} \leftarrow \langle 0\rangle \\ \neg\mathit{math} \leftarrow \langle 0\rangle & \neg\mathit{lang} \leftarrow \langle 0\rangle & \neg\mathit{psych} \leftarrow \langle 0\rangle & \neg\mathit{prac} \leftarrow \langle 0\rangle & \neg\mathit{phys} \leftarrow \langle 0\rangle \end{array}$$

$$\begin{array}{ccc} \mathit{hire} \leftarrow \langle 3\rangle & \neg\mathit{hire} \leftarrow \neg\mathit{math}\langle 1\rangle & \neg\mathit{hire} \leftarrow \neg\mathit{lang}\langle 1\rangle \\ \neg\mathit{hire} \leftarrow \neg\mathit{psych}\langle 3\rangle & \neg\mathit{hire} \leftarrow \neg\mathit{prac}\langle 2\rangle & \neg\mathit{hire} \leftarrow \neg\mathit{phys}\langle 4\rangle \end{array}$$

Intuitively, the rules with weight 0, i.e. no penalty involved when defeated, represent the choice between passing or not passing a certain test. Furthermore, the last five rules encode which penalty is involved when a person fails a certain test, but still gets hired. E.g., not passing the practical test is the same as failing both math and language. On the other hand, not passing the physical is considered unacceptable while failing the psychological test will be tolerated only if it is the only failed test. Finally, the rule $hire \leftarrow \langle 3 \rangle$ expresses the company's policy: defeating this rule is cheaper from the moment the penalty gets higher than 3.

Some of the program's extended answer sets are $M_1 = \{math, lang, psych, prac, phys, hire\}$, $M_2 = \{\neg math, \neg lang, psych, prac, phys, hire\}$, $M_3 = \{math, lang, psych, \neg prac, phys, hire\}$, $M_4 = \{\neg math, lang, psych, \neg prac, phys, hire\}$ and $M_5 = \{\neg math, lang, psych, \neg prac, phys, \neg hire\}$.

Computing the penalties for these extended answer sets results in $\Phi_P(M_1) = 0$, $\Phi_P(M_2) = \Phi_P(M_3) = 2$ and $\Phi_P(M_4) = \Phi_P(M_5) = 3$. These values imply the following order among the given extended answer sets: $M_1 \prec \{M_2, M_3\} \prec \{M_4, M_5\}$. It can be checked, that M_1 is the only (proper) weighted answer set of P. While M_2 has a penalty of 2 by defeating two rules with weight 1, M_3 only defeats a single rule, but with weight 2, yielding that M_2 and M_3 are incomparable, and thus equally preferred. Similarly, M_4 and M_5 only differ in the $hire$ atom and are incomparable with each other, both having a penalty of 3.

Combining simple programs with weights turns out to be rather expressive.

Theorem 1. *Let P be a SWP and let l be a literal. Deciding whether there exists a weighted answer set M of P containing l is Δ_2^P-complete.*

3 Weight Sequences

In [15] an intuitive semantics is presented for sequences of individual complex qualitative preferences. The idea is to apply each individual preference in the sequence in turn and to let it sort out the preferred answer sets left over by the previous preferences in the sequence. It is shown in [15] that this semantics is quite expressive as it can handle arbitrary complete problems of the polynomial hierarchy. More specifically, for a sequence of n preference relations, the semantics is Σ_{n+1}^P-complete.

It is natural to wonder if a similar semantics for sequences of individual weights will also yield a complexity blow-up depending on the length of the sequence. It turns out that this is not the case as sequences of weights remain Δ_2^P-complete.

Definition 3. *An n-**weight sequence rule** is a rule r of the form $a \leftarrow \beta \langle w_i \rangle_{i=1,\ldots,n}$, where $\{a\} \cup \beta$ is a finite set of literals and $\langle w_i \rangle_{i=1,\ldots,n}$ is a sequence of n associated weight values, i.e. a sequence of non-negative integers. We use $w_i(r)$ to denote the weight w_i of r. A countable set of n-weight sequence rules is an **n-weight sequence program** (nWSP). The **extended answer sets** of an nWSP P coincide with the extended answer sets of the SLP P' obtained from P by removing the weight sequences from the rules.*

*The **penalty** of an extended answer set S w.r.t. the weights i ($1 \leq i \leq n$) and an nWSP P, is defined by $\Phi_P^i(S) = \sum_{r \in P \setminus P_S} w_i(r)$, i.e. the sum of the weights w_i of all defeated rules in P w.r.t. S. Each of the penalties Φ_P^i induces a preference relation $\prec_i$ between the extended answer sets, as in Definition 2.*

We define the preference of extended answer sets up to a certain weight level by induction.

Definition 4. *Let P be a nWSP. An extended answer set S is preferable up to weight level $\prec_i$, $1 \leq i \leq n$, iff*

- *$i = 1$ and S is minimal w.r.t. $\prec_1$, or*
- *$i > 1$, S is preferable up to $\prec_{i-1}$, and there is no T, preferable up to $\prec_{i-1}$, such that $T \prec_i S$.*

An extended answer set S of P is a ***weighted answer set*** *iff it is preferable up to $\prec_n$.*

Example 3. Consider the problem of two people having to decide what to eat for dinner. After checking the available ingredients, the cook preparing the dinner decides to let his wife propose some possible combinations from which he will choose the final one. As his wife is rather hungry, she decides to choose the meal which is quickest to make, the reason for which she assigns weights corresponding with times needed to make a particular part of the meal. On the other hand, her husband is tired and wants to make a meal that is easy to prepare, yielding weights representing the difficulty to make a particular part of the meal. Further, they agree on some constraints that each meal should satisfy, e.g. with french fries they take mayonnaise, etc. The 2WSP corresponding with this problem is shown below.

Note that the rule $\neg v \leftarrow v\langle 200, 200\rangle$ enforces the satisfaction of the common constraints, as it implies that every solution not making one of the rules with v in the head applicable, is better than any solution making one of those rules applicable.

$$\begin{array}{rrr}
\mathit{french\_fries} \leftarrow \langle 0,0\rangle & \mathit{rice} \leftarrow \langle 0,0\rangle & \mathit{steak} \leftarrow \langle 0,0\rangle \\
\neg\mathit{french\_fries} \leftarrow \langle 15,1\rangle & \neg\mathit{rice} \leftarrow \langle 5,1\rangle & \neg\mathit{steak} \leftarrow \langle 10,1\rangle \\
\mathit{stew} \leftarrow \langle 0,0\rangle & \mathit{meat\_ball} \leftarrow \langle 0,0\rangle & \mathit{mayonnaise} \leftarrow \langle 0,0\rangle \\
\neg\mathit{stew} \leftarrow \langle 75,3\rangle & \neg\mathit{meat\_ball} \leftarrow \langle 20,2\rangle & \neg\mathit{mayonnaise} \leftarrow \langle 10,5\rangle \\
\mathit{tomato\_sauce} \leftarrow \langle 0,0\rangle & \neg\mathit{tomato\_sauce} \leftarrow \langle 10,2\rangle &
\end{array}$$

$$\begin{array}{ll}
v \leftarrow \neg\mathit{french\_fries}, \neg\mathit{rice}\langle 0,0\rangle & v \leftarrow \neg\mathit{steak}, \neg\mathit{meat\_ball}, \neg\mathit{stew}\langle 0,0\rangle \\
v \leftarrow \mathit{steak}, \neg\mathit{french\_fries}\langle 0,0\rangle & v \leftarrow \mathit{rice}, \mathit{meat\_ball}, \neg\mathit{tomato\_sauce}\langle 0,0\rangle \\
v \leftarrow \mathit{french\_fries}, \neg\mathit{mayonnaise}\langle 0,0\rangle & \neg v \leftarrow v\langle 200,200\rangle
\end{array}$$

For the extended answer sets[3] $S_1 = \{\mathit{french\_fries}, \mathit{steak}, \mathit{mayonnaise}\}$ and $S_2 = \{\mathit{rice}, \mathit{meat\_ball}, \mathit{tomato\_sauce}\}$ one can check that $\Phi^1_P(S_1) = \Phi^1_P(S_2) = 35$ and no other extended answer sets exists with a smaller penalty for Φ^1_P, yielding that both S_1 and S_2 are preferable up to weight level $\prec_1$. On the other hand, $\Phi^2_P(S_1) = 7$ and $\Phi^2_P(S_2) = 5$, making S_2 preferable up to weight level $\prec_2$, yielding that S_2 is the weighted answer set for this problem.

Finally, rearranging the weight sequence yields, in general, different solutions. E.g., if the cook first decides which meals he wants to make and afterward his wife can choose a particular one, it can be checked that $S_3 = \{\mathit{rice}, \mathit{stew}\}$ will be the weighted answer set of the problem.

In the following theorem we show that an n-weight sequence program can be transformed into a simple weight program such that the weighted answer sets of the former coincide with the weighted answer sets of the latter.

[3] To keep the size of the extended answer sets small, we only provide the positive literals.

Theorem 2. *Let P be an nWSP and let P' be the SWP defined by*

$$P' = \{a \leftarrow \beta\langle w_i \times 10^{\xi_i}\rangle \mid a \leftarrow \beta\langle w_i\rangle_{i=1,\ldots,n}\} \ ,$$

where $\xi_n = 0$ and $\xi_i = \sum_{j\in[i+1\ldots n]} \left(\mathit{length}\left(\sum_{r\in P} w_j(r)\right)\right)$ otherwise, with $\mathit{length}(x)$ the number of digits in x, e.g. $\mathit{length}(2611) = 4$.

Then, S is a weighted answer set of P iff S is a weighted answer set of P'.

Reconsider the rule $\neg stew \leftarrow \langle 75, 3\rangle$ from Example 3. In the SWP version of this program, the rule would yield the rules $\neg stew \leftarrow \langle 3\rangle$ and $\neg stew \leftarrow \langle 75000\rangle$, as $\sum_{r\in P} w_2(r) = 215$, yielding that $\mathit{length}(215) = 3$ and $75 \times 10^3 = 75000$.

The above transformation can be performed in polynomial time, yielding the following complexity result for n-weighted sequence programs.

Corollary 1. *Let P be an nWSP. Deciding whether there exists a weighted answer set S of P containing l is Δ_2^P-complete.*

This result implies that, unlike for sequences of qualitative preferences [15], introducing sequences of weights does not yield an increase of expressiveness. Nevertheless, these sequences allow for a more intuitive expression of certain problems.

4 Largest Common Subgraphs and Approximate Subgraph Isomorphisms

While largest common subgraphs and approximate subgraph isomorphisms are similar to finding largest common subtrees [1], the formalization we introduce in this section is, to the best of our knowledge, new.

A *graph* is a tuple $G = \langle N, E\rangle$, where N is a finite set of *nodes*, and $E \subseteq N \times N$ is a set of tuples representing the *edges* in the graph. We assume that graphs are directed; an undirected edge from n to m can still be represented by having both $\langle m, n\rangle$ and $\langle n, m\rangle$ in E.

Two graphs $G_1 = \langle N_1, E_1\rangle$ and $G_2 = \langle N_2, E_2\rangle$ are said to be *isomorphic*, denoted $G_1 \cong G_2$, if there exists a bijection $f : N_1 \to N_2$ such that $f(E_1) = E_2$, where $f(E_1)$ denotes $\{\langle f(t), f(h)\rangle \mid \langle t, h\rangle \in E\}$. On the other hand, G_2 is called a *subgraph* of G_1, denoted $G_2 \preceq G_1$, iff $N_2 \subseteq N_1$ and $E_2 \subseteq E_1$. Furthermore, G_2 is called *subgraph isomorphic* to G_1, denoted $G_2 \precsim G_1$, if there exists a subgraph $G_3 \preceq G_1$ such that $G_2 \cong G_3$. A graph G_4 is called a *common subgraph* of G_1 and G_2, denoted $G_4 \preceq \{G_1, G_2\}$, if $G_4 \preceq G_1$ and[4] $G_4 \precsim G_2$.

For certain applications the notion of common subgraphs is too weak and one is more interested in finding the largest common subgraph between two given graphs. Formally, a graph G_4 is a *largest common subgraph* of G_1 and G_2, denoted $G_4 \preceq^{max} \{G_1, G_2\}$ if $G_4 \preceq \{G_1, G_2\}$ and there does not exist a graph $G_5 \preceq \{G_1, G_2\}$ such that $|E_5| > |E_4|$. The set of all largest common subgraphs is denoted by $\mathcal{C}(G_1, G_2)$.

[4] Note that the demand that G_4 is a subgraph of G_1 is needed to guarantee the finiteness of the set off all common subgraphs between G_1 and G_2.

On the other hand, subgraph isomorphism is sometimes too strong a notion for certain applications. E.g., when a graph $G_2 = \langle N_2, E_2 \rangle$ is not subgraph isomorphic to a graph $G_1 = \langle N_1, E_1 \rangle$, it may be interesting to know what is "missing" in G_1 for G_2 to be subgraph isomorphic to it. In this context, a graph $G_3 = \langle N_3, E_3 \rangle$ is called an *extension* of G_1 w.r.t. G_2 just when $G_1 \preceq G_3$ and $N_3 = N_1$ when $|N_1| \geq |N_2|$ or $N_3 = N_1 \cup \{x_i \mid 1 \leq i \leq |N_2| - |N_1|\}$ otherwise, where the x_i are new nodes not occurring in N_1. The latter construction of N_3 is necessary to handle the cases in which the graph to search for is bigger than the graph to search in. A graph G_3 is a *subgraph isomorphic approximation* of G_1 w.r.t. G_2 iff G_3 is an extension of G_1 w.r.t. G_2 and $G_2 \precsim G_3$. We use $G_2 \precapprox_{G_1} G_3$ to denote that G_2 is *approximately subgraph isomorphic* to G_3 w.r.t. G_1, i.e. G_3 is a subgraph isomorphic approximation of G_1 w.r.t. G_2. The set of all subgraph isomorphic approximations of G_1 w.r.t. G_2 is denoted by $\mathcal{A}_{G_1}(G_2)$.

Obviously, not every subgraph isomorphic approximation $G_3 \in \mathcal{A}_{G_1}(G_2)$ is equally interesting. E.g., the fully connected graph $\langle N_3, N_3 \times N_3 \rangle$ is, clearly, always a subgraph isomorphic approximation and thus in $\mathcal{A}_{G_1}(G_2)$. However, in most cases there will exist smaller extensions of G_1 in $\mathcal{A}_{G_1}(G_2)$. Therefore, we are particularly interested in elements from $\mathcal{A}_{G_1}(G_2)$ that have a minimal, in some sense, difference with the original graph G_1. Here we use $\Delta_{G_1}(G_3)$ to denote the *unidirectional edge difference* between G_1 and G_3, i.e. $\Delta_{G_1}(G_3) = E_3 \backslash E_1$.

Two minimality criteria, which are widely used in areas like diagnostic reasoning [5, 6, 17], are cardinal minimality and subset minimality. In the former case, we select those elements from $\mathcal{A}_{G_1}(G_2)$ that are minimal w.r.t. cardinality among the elements in $\mathcal{A}_{G_1}(G_2)$. Formally, a graph $G_3 \in \mathcal{A}_{G_1}(G_2)$ is said to be a *subgraph isomorphic c-approximation* iff there does not exist a graph $G_4 \in \mathcal{A}_{G_1}(G_2)$ such that $|\Delta_{G_1}(G_4)| < |\Delta_{G_1}(G_3)|$. The set of all c-approximations is denoted by $\mathcal{A}^c_{G_1}(G_2)$.

Example 4. Consider the four undirected graphs G_1, G_2, G_3 and G_4 represented in Figure 1. It is clear that G_4 is one of the largest common subgraphs between G_1 and

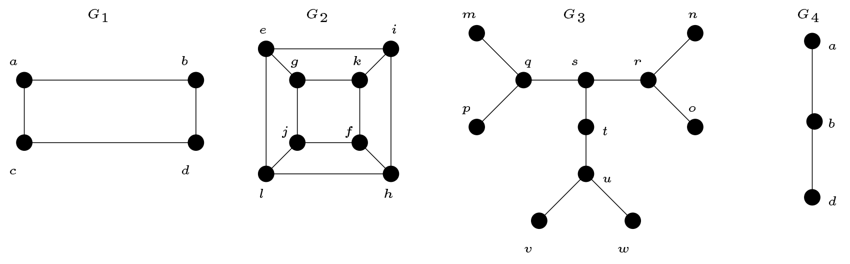

Fig. 1. The graphs G_1, G_2, G_3 and G_4 of Example 4.

G_3, i.e. $G_4 \in \mathcal{C}(G_1, G_3)$.

On the other hand, G_1 is subgraph isomorphic to G_2, i.e. $G_1 \precsim G_2$, but not to G_3. However, adding a single (bidirectional) edge between e.g. m and r in G_3, i.e. $G_5 = \langle N_3, E_3 \cup \{\langle m, r \rangle, \langle r, m \rangle\} \rangle$, results in a subgraph isomorphic approximation of G_3 w.r.t. G_1, i.e. $G_1 \precapprox_{G_3} G_5$. Obviously, G_5 is cardinal minimal yielding that $G_5 \in \mathcal{A}^c_{G_3}(G_1)$.

Subset minimal isomorphic approximations can be defined in a similar way. However, in contrast with diagnostic reasoning, subset minimality is less intuitive in this setting. E.g. adding the edges $\langle p, o\rangle$, $\langle o, w\rangle$, $\langle w, v\rangle$ and $\langle v, p\rangle$ (and their reverses) to G_3 in Example 4 yields a subset minimal isomorphic approximation w.r.t. G_1. However, if we see G_3 as an activity graph and G_1 as a pattern of interest, as is often done by intelligence agencies for detecting possible threats [4], the previously mentioned subset minimal approximation is not very useful as it forces the agency to check 4 possible relations between currently unrelated things. On the other hand, the approximations in $\mathcal{A}^c_{G_3}(G_1)$ are of much more value as they all yield one missing link to complete the pattern, implying that the agency can quickly confirm these solutions (see also the next section).

Obviously, when a graph is subgraph isomorphic to another one, the latter is the only c-approximation of itself.

Theorem 3. *Let G_1 and G_2 be graphs such that $G_2 \precsim G_1$. Then, $\mathcal{A}^c_{G_1}(G_2) = \{G_1\}$.*

Using the weighted answer set semantics, we have the means to effectively compute, for given graphs G_1 and G_2, the largest common subgraphs of G_1 and G_2; or the c-approximations of G_1 w.r.t. G_2. In what follows, we will sometimes use non-grounded rules for clarity, but grounding is performed as usual. Computing the largest common subgraphs can be done using the following transformation.

Definition 5. *Let $G_1 = \langle N_1, E_1\rangle$ and $G_2 = \langle N_2, E_2\rangle$ be graphs. The weighted program, denoted $\mathcal{L}(G_1, G_2)$, computing the largest common subgraphs between G_1 and G_2 is defined by the rules (where $\Upsilon = |N_1|^2 + 1$):*

1. $\{node1(x) \leftarrow \langle 0\rangle \mid x \in N_1\} \cup \{node2(x) \leftarrow \langle 0\rangle \mid x \in N_2\}$;
2. $\{edge1(x, y) \leftarrow \langle 0\rangle \mid \langle x, y\rangle \in E_1\} \cup \{\neg edge1(x, y) \leftarrow \langle 0\rangle \mid \langle x, y\rangle \in E_1^2 \backslash E_1\}$;
3. $\{edge2(x, y) \leftarrow \langle 0\rangle \mid \langle x, y\rangle \in E_2\} \cup \{\neg edge2(x, y) \leftarrow \langle 0\rangle \mid \langle x, y\rangle \in E_2^2 \backslash E_2\}$;
4. $\{\neg no\_con(x, y) \leftarrow \langle 0\rangle \mid x \in N_1, y \in N_2\}$;
5. $\{con(X, Y) \leftarrow node1(X), node2(Y), \neg no\_con(X, Y)\langle \Upsilon\rangle\}$;
6. $\{no\_con(X, Y) \leftarrow con(X', Y), node1(X), X \neq X'\langle \Upsilon\rangle\}$;
7. $\{no\_con(X, Y) \leftarrow con(X, Y'), node2(Y), Y \neq Y'\langle \Upsilon\rangle\}$;
8. $\{common(X, Y) \leftarrow edge1(X, Y), con(X, X'), con(Y, Y'), edge2(X', Y')\langle 0\rangle\}$;
9. $\{threshold \leftarrow \langle \Upsilon\rangle\}$;
10. $\{\neg threshold \leftarrow edge1(X, Y), con(X, X'), con(Y, Y'), \neg edge2(X', Y')\langle 1\rangle\}$;

Intuitively, the rules in (1) introduce the nodes of the given graphs as facts, while the rules in (2) and (3) introduce the edges of the graphs as positive facts and the edges not appearing in the graphs as negative facts. Further, the rules in (4) are used to introduce negation as failure for the $no\_con/2$ predicate, while the rules in (5), (6) and (7) are used to setup an injective relation between the nodes of G_1 and G_2. The rules in (8) are used to retrieve the largest common subgraph of G_1 and G_2. Finally, the rules in (9) and (10) ensure that only the largest common subgraphs are computed, i.e. the rule in (9) will never be defeated due to its high threshold and so the rules in (10) count the missing edges in the subgraph w.r.t. the original graph as missing edges lead to defeated rules.

Reconsidering the graphs G_1 and G_3 from Example 4, one of the possible weighted answer sets for $\mathcal{L}(G_1, G_2)$ will contain, besides the numerous other predicates, the predicates $\{common(a, b), common(b, c), common(b, a), common(c, b)\}$, corresponding to the largest common subgraph G_4 in Example 4. This behavior is confirmed by the following theorem.

Theorem 4. *Let $G_1 = \langle N_1, E_1\rangle$ and $G_2 = \langle N_2, E_2\rangle$ be two graphs. Then, $G_3 = \langle N_3, E_3\rangle \in \mathcal{C}(G_1, G_2)$ iff there exists a weighted answer set M of $\mathcal{L}(G_1, G_2)$, with $\Phi_{\mathcal{L}(G_1,G_2)}(M) < \Upsilon$, such that $\{common(x, y) \mid \langle x, y\rangle \in E_3\} \subseteq M$.*

To compute c-approximations of given graphs, we introduce the edges of G_1 as facts of the form $edge(x, y) \leftarrow \langle 0\rangle$, where $\langle x, y\rangle \in E_1$. For each possible edge $\langle x, y\rangle \notin E_1$, with $x, y \in N_1$, we give a choice to either include it or not in an approximation by introducing the facts $edge(x, y) \leftarrow \langle 0\rangle$ and $\neg edge(x, y) \leftarrow \langle 1\rangle$. The penalty involved in the latter fact is to ensure that the computed approximations are cardinal minimal, i.e. not inserting an edge (defeating the former rule) can be done freely, but inserting an edge (defeating the latter rule) has to be minimized. In case $|N_1| < |N_2|$ we also add edges to the $|N_2| - |N_1|$ new nodes.

To match G_2 with the possible approximations, we need to introduce for each node $n \in N_2$ a unique new variable name N. Searching for a match of G_2 in the approximation is done by the single rule $match \leftarrow \beta\langle 0\rangle$, where $\beta = \{edge(X, Y) \mid \langle x, y\rangle \in E_2\} \cup \{X \neq Y \mid \langle x, y\rangle \in E_2 \wedge x \neq y\}$. Finally, we add the single rule $match \leftarrow$ *not* $match\langle 0\rangle$ which forces any solution to contain a match (note that this rule cannot be defeated).

Definition 6. *Let $G_1 = \langle N_1, E_1\rangle$ and $G_2 = \langle N_2, E_2\rangle$ be graphs. The program computing the c-approximations of G_1 w.r.t. G_2, denoted $\mathcal{L}_{G_1}(G_2)$, is defined by the rules:*

- $\{edge(x, y) \leftarrow \langle 0\rangle \mid \langle x, y\rangle \in E_1\}$;
- $\{edge(x, y) \leftarrow \langle 0\rangle\ ;\ \neg edge(x, y) \leftarrow \langle 1\rangle \mid x, y \in N_1 \cup \{x_i \mid (|N_1| < |N_2|) \wedge (1 \leq i \leq |N_2| - |N_1|)\} \wedge \langle x, y\rangle \notin E_1\}$;
- $\{match \leftarrow \beta\langle 0\rangle\}$, *where* $\beta = \{edge(X, Y) \mid \langle x, y\rangle \in E_2\} \cup \{X \neq Y \mid \langle x, y\rangle \in E_2 \wedge x \neq y\}$; *and*
- $\{match \leftarrow$ *not* $match\langle 0\rangle\}$.

If we reconsider the graphs G_1 and G_3 from Example 4, the program $\mathcal{L}_{G_3}(G_1)$ contains, besides the numerous $edge/2$ facts, the rule

$$\begin{aligned} match \leftarrow\ & edge(A, B), edge(B, D), edge(D, C), edge(C, A), edge(B, A), edge(D, B) \\ & edge(C, D), edge(A, C), A \neq B, B \neq D, D \neq C, C \neq A\ . \end{aligned}$$

One of the possible weighted answer sets of $\mathcal{L}_{G_3}(G_1)$ is e.g. $S = \{edge(x, y) \mid \langle x, y\rangle \in E_3\} \cup \{edge(m, r), edge(r, m)\} \cup (\{\neg edge(x, y) \mid x, y \in N_3 \wedge \langle x, y\rangle \notin E_3\} \setminus \{edge(m, r), edge(r, m)\})$. Clearly, S corresponds with the extension G_4 from Example 4, which is a cardinal minimal approximation of G_3 w.r.t. G_1. This behavior is confirmed by the following theorem.

Theorem 5. *Let $G_1 = \langle N_1, E_1 \rangle$ and $G_2 = \langle N_2, E_2 \rangle$ be graphs. Then, $G_3 = \langle N_3, E_3 \rangle \in \mathcal{A}^c_{G_1}(G_2)$ iff $M = \{edge(x, y) \mid \langle x, y \rangle \in E_3\} \cup \{\neg edge(x, y) \mid x, y \in N_3 \wedge \langle x, y \rangle \notin E_3\} \cup \{match\}$ is a weighted answer set of $\mathcal{L}_{G_1}(G_2)$.*

In the current approach no distinction is made between the edges that can be added to a graph to obtain an approximation. However, one can imagine situations in which adding one edge is more "difficult" than adding another, i.e. the cost of adding an edge may vary. E.g., for an intelligence agency, it may be easier to check a relationship between people in the home country, than between people in foreign countries, but checking 4 internal relationships may be as hard as checking 1 external relationship, resulting in a cost of 4 for edges between externals and a cost of 1 for edges between internals. Such costs represent a quantitative preference relation between edge additions.

In this case, optimal solutions are approximations that minimize the sum of all costs associated with the added edges in the approximation. It is not difficult to see that this kind of minimization can easily be computed by an adapted version of the program in Definition 6: just replace the weights 1 with the cost associated for adding the edge to an approximation. Clearly, Theorem 5 remains valid in this extension.

Similarly, we could think of an agency where possible threats are first selected, by some field agent, depending on the effort needed to check certain relationships. Afterward, the supervisor will apply, on the proposed investigations of his field agent, another kind of quantitative preferences, e.g. using information from other departments. In case there are still a number of possible solutions left over after the supervisor, even a third individual, e.g. the director, could apply his preferences on these possibilities. Again, it is not difficult to see that this problem can be elegantly modeled by an adapted version of the program in Definition 6, this time using the n-weight sequence programs introduced in Section 3. Also in this extension, an adapted version of Theorem 5 remains valid.

5 An Application in Intelligence Analysis

Attributed relational graphs (ARGs), an extension of the abstract directed graphs defined in the previous section, are often used in e.g. intelligence analysis to understand complex, and often uncertain, situations. The nodes in such ARGs are used to describe objects in the observed world, e.g. persons, organizations, ..., while the edges are used to represent relationships between the nodes, e.g. interaction, ownership, trust, ...

In addition, ARG nodes and edges may have additional attributes that describe the details of the specific objects or relationships: e.g. the name of a person, the kind of chemical, the type of conversation. An example of such an ARG, based on an example from [4], can be found in Figure 3. Here, a person named Bill has rented a truck for carrying liquids and that same person resides in a house at 123 Main street together with a person called Ted. Furthermore, Ted has been observing a factory called Acme Inc. and he also bought large quantities of the chemical HCl.

Intelligence analysts normally define small abstract patterns which are believed to be indications of possible threats. An example of such a pattern, based on the same example from [4], can be found in Figure 2. Intuitively, it states that two persons residing

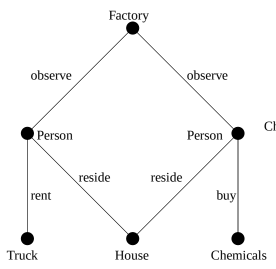

Fig. 2. The pattern graph [4].

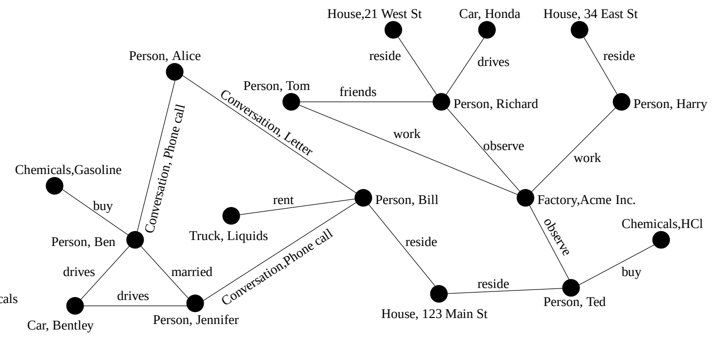

Fig. 3. The observed activity graph [4].

at the same place and both observing the same factory can be dangerous if one person buys some chemical, while the other rents a truck.

Having both an ARG of observed activity and a pattern, the analysts need tools for finding specific regions in the ARG that "closely" match the defined threat pattern. Subgraph isomorphic approximations turn out to be valuable tools to accomplish this task [4]. The framework and results we developed in Section 4 can be intuitively adapted to the setting of ARGs, where the transformation into a weighted program allows an analyst to compute subgraph isomorphic approximations that are minimal in some quantitative sense. In situations where investigating missing additional relationships is equally hard, the analyst can use the cardinal minimal approximations. On the other hand, if investigating some relationship has a higher cost than investigating others, an analyst could rely upon the extension of the framework of Section 4, i.e. defining a cost with each relationship (edge) that can be added to have a subgraph isomorphic approximation and only keeping the approximations that minimize the sum of the costs. Similarly, it could be the case that the analyst is not the only one in charge of making the final decision or that he has multiple equivalent possibilities. In such situations, it can be useful to apply the quantitative preferences of some other people, e.g. a supervisor or the director, to refine the number of solutions, so obtaining the most preferred solution. By using the second extension of the framework of Section 4, also this kind of reasoning with ARGs can be solved, i.e. by using weight sequence programs.

Instead of formally adapting the framework and the results, we illustrate the adaptation, and its usefulness, using the example on intelligence analysis: we will translate the ARG and pattern of Figures 3 and 2 into a weighted program and show that the solutions of the program correspond with the regions of threat in the ARG w.r.t. the given pattern.

First we translate, for convenience, the nodes of the ARG to *node*-predicates. E.g. a person named Bill forces the fact $node(person, bill) \leftarrow \langle 0 \rangle$ into the program, while the factory Acme Inc. is responsible for the fact $node(factory, acme\_inc) \leftarrow \langle 0 \rangle$. In total, we have 17 of such facts in our weighted program.

Next, we have to describe the relationships between the nodes using extended versions of the *edge/2*-predicates used in the previous section. E.g. Ted residing at the house in 123 Main street gives rise to the fact

$$edge(person, ted, reside, house, \mathit{123\_main\_street}) \leftarrow \langle 0 \rangle \ \ ,$$

while the conversation between Jennifer and Bill can be described by the fact

$$edge(person, bill, conversation, phone, person, jennifer) \leftarrow \langle 0 \rangle \ \ .$$

Note that the different *edge*-facts can have different arities, which is not a problem as long as the arities, and the ordering of the arguments, are the same for the same relationship. E.g. *edge*-facts representing the conversation relationship always have six arguments: the first two correspond to a node, the third has to be “conversation”, the fourth the type of conversation and the last two again correspond to a node.

Also note that ARGs are directed graphs, but certain relations are bidirectional, e.g. *friends* and *married*. For these relationships we have to explicitly add both directions using the *edge*-facts: e.g. both $edge(person, richard, friend, person, tom) \leftarrow \langle 0 \rangle$ and $edge(person, tom, friend, person, richard) \leftarrow \langle 0 \rangle$ have to be present in the weighted program. One could argue that a conversation through phone is also bidirectional, but we use a directed edge here to represent who initiated the call.

The pattern in Figure 2 can be translated into the following rule, where names starting with an uppercase letter correspond to a variable:

$$\begin{aligned} match \leftarrow\ & edge(person, \mathit{NamePerson1}, observe, factory, \mathit{NameFactory}), \\ & edge(person, \mathit{NamePerson2}, observe, factory, \mathit{NameFactory}), \\ & edge(person, \mathit{NamePerson1}, reside, house, \mathit{AddressHouse}), \\ & edge(person, \mathit{NamePerson2}, reside, house, \mathit{AddressHouse}), \\ & edge(person, \mathit{NamePerson1}, rent, truck, \mathit{KindOfTruck}), \\ & edge(person, \mathit{NamePerson2}, buy, chemicals, \mathit{KindOfChemical}) \langle 0 \rangle \end{aligned}$$

The above pattern matching rule also matches situations where only one person observes a factory and does both the renting of the truck and the buying of the chemicals. If one wants to have explicitly two different persons, we need to add the condition $\mathit{NamePerson1} \neq \mathit{NamePerson2}$ to the rule.

Finally, we have to add rules for the edges that can eventually be added to our activity graph to obtain a subgraph isomorphic approximation. These edges will directly point out the region of interest in the activity graph as the minimization assures that only edges are added where necessary, i.e. on those places in the activity graph where the pattern (almost) matches. While we introduced all possible edges in the simulation of Section 4, doing the same in the context of ARGs may not be the best way to go. Indeed, ARGs can have multiple edges between the same nodes but with different attributes, which are not always useful to define between certain types of nodes. E.g. $edge(chemical, hcl, buys, chemical, gasoline) \ \leftarrow \ \langle 0 \rangle$ is theoretically possible, but useless in real life. Therefore, one should avoid the introduction of meaningless edges in the program, possibly by adding extra semantical constraints, e.g. typing the attributes in ARGS. Some examples of choices of edges to add are:

$$\begin{array}{l}
\ edge(person, bill, observe, factory, acme\_inc) \leftarrow \langle 0 \rangle \\
\neg\ edge(person, bill, observe, factory, acme\_inc) \leftarrow \langle v \rangle \\
\ edge(person, bill, buy, chemical, hcl) \leftarrow \langle 0 \rangle \\
\neg\ edge(person, bill, buy, chemical, hcl) \leftarrow \langle w \rangle \\
\ edge(person, alice, conversation, phone, person, ted) \leftarrow \langle 0 \rangle \\
\neg\ edge(person, alice, conversation, phone, person, ted) \leftarrow \langle z \rangle
\end{array}$$

In the above rules for possible edges to add, the rules with a positive occurrences of the *edge*-predicate always have a weight of 0, as not adding an edge, i.e. defeating the rule, can be done for free. On the other hand, the negative occurrences have a weight corresponding to the cost associated with adding the edge. In case we use cardinal minimality, the costs (e.g. v, w and z) will all be 1, while in case of total cost minimality we could define $v = 4$, $w = 2$ and $z = 1$ yielding that it is twice as hard to check if someone observed a factory than checking if he bought some chemical, which in turn is twice as hard than checking if he made a phone call.

For simplicity, we only consider cardinal minimality (and no sequences) in what follows, i.e. we take all the weights of the rules with negative occurrence of an *edge*-predicate to be 1. If we consider the weighted program obtained in the way we described above, we will have two weighted answer sets S and T. Both will contain all the edges from the original activity graph together with the fact $match$. Additionally, S will contain the fact $edge(person, bill, observe, factory, acme\_inc)$ together with all negated versions of the other *edge*-predicates we added to the program Similarly, T will contain the fact $edge(person, ted, rent, truck, liquids)$ together with all negated versions, except the one occurring positively. Clearly, both S and T correspond with the only cardinal minimal subgraph isomorphic approximations of the problem.

As said before, we can add the condition $NamePerson1 \neq NamePerson2$ to the pattern rule in our program if we explicitly want two different persons. When we consider the weighted program obtained in that way, S will be the single weighted answer set of the program, corresponding to the single subgraph isomorphic approximation of the problem.

Finally, one can imagine situations in which one has two observed activity graphs, e.g. provided by two different agencies, and one is interested in finding the largest common observed activity in those ARGs. By using an adapted version of Definition 5 in the context of ARGs, the corresponding modified version of Theorem 4 demonstrates that also these kind of problems in intelligence analysis can be conveniently tackled by the presented framework.

6 Conclusions and Directions for Further Research

We presented a simple and intuitive quantitative preferential semantics based on the extended answer set semantics, characterized its expressiveness and illustrated its usefulness using an application in the area of intelligence analysis. Other areas of possible applications that are interesting for further exploration include molecular biology, where e.g. largest common subgraphs are used to detect similarities between large molecules

[11, 13, 1]. Further research also encompasses the efficient implementation of the semantics, e.g. using existing answer set solvers such as dlv [7] or smodels [12]. Furthermore, the relationships between the present proposal and other weighted semantics such as weak constraints [3] need to be investigated.

References

1. Tatsuya Akutsu and Magnús M. Halldórsson. On the approximation of largest common subtrees and largest common point sets. *Theoretical Comp. Science*, 233(1-2):33–50, 2000.
2. Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press, 2003.
3. Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Strong and weak constraints in disjunctive datalog. In *Proceedings of the 4th International Conference on Logic Programming (LPNMR '97)*, pages 2–17, 1997.
4. Thayne Coffman, Seth Greenblatt, and Sherry Marcus. Graph-based technologies for intelligence analysis. *Communications of the ACM*, 47(3):45–47, 2004.
5. L. Console and P. Torasso. A spectrum of logical definitions of model-based diagnosis. *Computational Intelligence*, 7(3):133–141, 1991.
6. Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. The diagnosis frontend of the dlv system. *AI Communications*, 12(1-2):99–111, 1999.
7. Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Declarative problem-solving using the dlv system. *Logic-Based Artificial Intelligence*, pages 79–103, 2000.
8. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Logic Programming, Proceedings of the Fifth International Conference and Symposium*, pages 1070–1080. MIT Press, 1988.
9. R.J. Heuer. Psychology of intelligence analysis. Center for the Study of Intelligence, Central Intelligence Agency, 2001.
10. Vladimir Lifschitz. Answer set programming and plan generation. *Journal of Artificial Intelligence*, 138(1-2):39–54, 2002.
11. R. E. Stobaugh. Chemical substructure searching. *Journal of Chemical Information and Computer Sciences*, 25:271–275, 1985.
12. Syrjänen T. and Niemelä I. The smodels system. In *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 2173 of *Lecture Notes in Computer Science*, pages 434–438, Vienna, Austria, September 2001. Springer.
13. Y. Takahashi, Y. Satoh, H. Suzuki, and S. Sasaki. Recognition of largest common structural fragment among a variety of chemical structures. *Analytical Sciences*, 3:23–28, 1987.
14. J.R. Ullman. An algorithm for subgraph isomorphism. *J. of the ACM*, 23(1):31–42, 1976.
15. Davy Van Nieuwenborgh, Stijn Heymans, and Dirk Vermeir. On programs with linearly ordered multiple preferences. In *Proc. of 20th Intl. Conference on Logic Programming (ICLP 2004)*, volume 3132 of *Lecture Notes in Computer Science*, pages 180–194. Springer, 2004.
16. Davy Van Nieuwenborgh and Dirk Vermeir. Preferred answer sets for ordered logic programs. In *European Conference on Logics in Artificial Intelligence, JELIA 2002*, volume 2424 of *Lecture Notes in Artificial Intelligence*, pages 432–443, 2002.
17. Davy Van Nieuwenborgh and Dirk Vermeir. Ordered diagnosis. In *Proceedings of the 10th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR2003)*, volume 2850 of *LNAI*, pages 244–258. Springer, 2003.
18. Marina De Vos and Dirk Vermeir. Logic programming agents playing games. In *Research and Development in Intelligent Systems XIX (ES2002)*, BCS Conference Series, pages 323–336. Springer-Verlag, 2002.