Adaptive Reasoning for Cooperative Agents

Luís Moniz Pereira and Alexandre Miguel Pinto
(lmp|amp)@di.fct.unl.pt

Centro de Inteligência Artificial - CENTRIA
Universidade Nova de Lisboa

Abstract. Using explicit affirmation and explicit negation, whilst allowing for a third logic value of undefinedness, can be useful in situations where decisions have to be taken on the basis of scarce, ambiguous, or downright contradictory information. In a three-valued setting, we consider an agent that learns a definition for both the target concept and its opposite, considering positive and negative examples as instances of two disjoint classes. Explicit negation is used to represent the opposite concept, while default negation is used to ensure consistency and to handle exceptions to general rules. Exceptions are represented by examples covered by the definition for a concept that belong to the training set for the opposite concept.
One single agent exploring an environment may gather only so much information about it and that may not suffice to find the right explanations. In such case, a cooperative multi-agent strategy, where each agent explores a part of the environment and shares with the others its findings, might provide better results. We describe one such framework based on a distributed genetic algorithm enhanced by a Lamarckian operator for belief revision. The agents communicate their candidate explanations — coded as chromosomes of beliefs — by sharing them in a common pool. Another way of interpreting this communication is in the context of argumentation. In the process of taking all the arguments and trying to find a common ground or consensus we might have to change, or review, some of assumptions of each argument.
The resulting framework we present is a collaborative perspective of argumentation where arguments are put together at work in order to find the possible 2-valued consensus of opposing positions of learnt concepts in an evolutionary pool in order to find the "best" explanation to the observations.

1 Introduction

Expressing theories as Logic Programs has become more natural and common as the field of Computational Logic has grown mature and other fields started to use its tools and results. Theories are usually expressed as a set of 'if-then' rules and facts which allow for the derivation, through the use of logic inference, of non-obvious results. When writing such rules and facts, explicit negation, just like explicit affirmation, can be used to formalize sure knowledge which provides inevitable results.

Theories can be further refined by adding special rules taking the form of Integrity Constraints (ICs). These impose that, whatever the assumptions might be, some conditions must be met. One implicit constraint on every reasonable theory is overall consistency, i.e, it must not be possible to derive one conclusion and its opposition.

Since in the real world the most common situation is one where there is incomplete and updatable information, any system making a serious attempt at dealing with real situations must cope with such complexities. To deal with this issue, the field of Computational Logic has also formalized another form of negation, Default Negation, used to express uncertain knowledge and exceptions, and used to derive results in the absence of complete information. When new information updates the theory, old conclusions might no longer be available (because they were relying on assumptions that became false with the new information), and further new conclusions might now be derived (for analogous reasons).

The principle we use is thus the *Unknown World Assumption* (UWA) where everything is unknown or undefined until we have some solid evidence of its truthfulness or falseness. This principle differs from the more usual *Closed World Assumption* (CWA) where everything is assumed false until there is solid evidence of its truthfulness. We believe the UWA stance is more skeptical, cautious, and even more realistic than the CWA. We do not choose a fuzzy logic approach due to its necessity of specific threshold values. For such an approach we would need to compute those values *a priori*, possibly recurring to a probabilistic frequency-based calculation. Accordingly, we use a 3-valued logic (with the *undefined* truth value besides the *true* and *false* ones) instead of a more classical 2-valued logic.

We start by presenting the method for theory building from observations we use — a 3-valued logic rule learning method, — and in the following section we focus on a method to analyze observations and to provide explanations for them given the learned theory. We show how the possible alternative explanations can be viewed as arguments for and against some hypotheses, and how we can use these arguments in a collaborative way to find better consensual explanations. Conclusions and outlined future work close this paper.

2 Theory building and refinement

Complete information about the world is impossible to achieve and it is necessary to reason and act on the basis of the available partial information coming from distinct agents. In situations of incomplete knowledge, it is important to distinguish between what is true, what is false, and what is unknown or undefined.

Such a situation occurs, for example, when each agent incrementally gathers information from the surrounding world and has to select its own actions on the basis of acquired knowledge. If the agent learns in a two-valued setting, it can encounter the problems that have been highlighted in [10]. When learning in a specific to general way, it will learn a cautious definition for the target concept and it will not be able to distinguish what is false from what is not yet known. Supposing the target predicate represents the allowed actions, then the agent will not distinguish forbidden actions from actions with a still unknown outcome and this can restrict the agent's acting power. If the agent learns in a general to specific way (i.e., the agent starts with a most general concept and progressively restricts it by adding exceptions as he learns), instead, it will not know the difference between what is true and what is unknown and, therefore, it can try actions with an unknown outcome. Rather, by learning in a three-valued setting,

it will be able to distinguish between allowed actions, forbidden actions, and actions with an unknown outcome. In this way, the agent will know which part of the domain needs to be further explored and will not try actions with an unknown outcome unless it is trying to expand its knowledge.

In [25] the authors showed that various approaches and strategies can be adopted in Inductive Logic Programming (ILP, henceforth) for learning with Extended Logic Programs (ELP) — including explicit negation — under an extension of well-founded semantics. As in [18, 19], where answer-sets semantics is used, the learning process starts from a set of positive and negative examples plus some background knowledge in the form of an extended logic program. Positive and negative information in the training set are treated equally, by learning a definition for both a positive concept p and its (explicitly) negated concept $\neg p$.

Default negation is used in the learning process to handle *exceptions* to general rules. Exceptions are examples covered by the definition for the positive concept that belong to the training set for the negative concept or examples covered by the definition for the negative concept that belong to the training set for the positive concept.

In this work, we consider standard ILP techniques to learn a concept and its opposite. Indeed, separately learned positive and negative concepts may conflict and, in order to handle possible *contradiction*, contradictory learned rules are made defeatable by making the learned definition for a positive concept p depend on the default negation of the negative concept $\neg p$, and vice-versa, i.e., each definition, possibly arising from distinct agents, is introduced as an exception to the other.

The learned theory will contain rules of the form:

$$p(\boldsymbol{X}) \leftarrow Body^{+}(\boldsymbol{X}) \qquad\qquad \neg p(\boldsymbol{X}) \leftarrow Body^{-}(\boldsymbol{X})$$

for every target predicate p, where $\boldsymbol{X}$ stands for a tuple of arguments. In order to satisfy the completeness requirement, the rules for p will entail all positive examples while the rules for $\neg p$ will entail all (explicitly negated) negative examples. The consistency requirement is satisfied by ensuring that both sets of rules do not entail instances of the opposite element in either of the training sets.

The ILP techniques to be used depend on the level of generality that we want to have for the two definitions: we can look for the Least General Solution (LGS) or the Most General Solution (MGS) of the problem of learning each concept and its complement. In practice, LGS and MGS are not unique and real systems usually learn theories that are not the least nor most general, but closely approximate one of the two. In the following, these concepts will be used to signify approximations to the theoretical concepts.

2.1 Strategies for Combining Different Generalizations

The generality of concepts to be learned is an important issue when learning in a three-valued setting. In a two-valued setting, once the generality of the definition is chosen, the extension (i.e., the generality) of the set of false atoms is automatically decided, because it is the complement of the true atoms set. In a three-valued setting, rather, the extension of the set of false atoms depends on the generality of the definition learned for the negative concept. Therefore, the corresponding level of generality may be chosen

independently for the two definitions, thus affording four epistemological cases. The adoption of ELP allows case combination to be expressed in a declarative and smooth way.

When classifying an as yet unseen object as belonging to a concept, we may later discover that the object belongs to the opposite concept. The more we generalize a concept, the higher is the number of unseen atoms covered by the definition and the higher is the risk of an erroneous classification. Depending on the damage that may derive from such a mistake, we may decide to take a more cautious or a more confident approach. If the possible damage from an over extensive concept is high, then one should learn the LGS for that concept, if the possible damage is low then one can generalize the most and learn the MGS. The overall risk will depend also on the use of the learned concepts within other rules: we need to take into account the damage that may derive from mistakes made on concepts depending on the target one.

The problem of selecting a solution of an inductive problem according to the cost of misclassifying examples has been studied in a number of works. PREDICTOR [15] is able to select the cautiousness of its learning operators by means of meta-heuristics. These meta-heuristics make the selection based on a user-input penalty for prediction error. In [33] Provost provides a method to select classifiers given the cost of misclassifications and the prior distribution of positive and negative instances. The method is based on the Receiver Operating Characteristic (ROC) [16] graph from signal theory that depicts classifiers as points in a graph with the number of false positives on the X axis and the number of true positive on the Y axis. In [28] it is discussed how the different costs of misclassifying examples can be taken into account into a number of algorithms: decision tree learners, Bayesian classifiers and decision list learners.

As regards the confidence in the training set, we can prefer to learn the MGS for a concept if we are confident that examples for the opposite concept are correct and representative of the concept. In fact, in top-down methods, negative examples are used in order to delimit the generality of the solution. Otherwise, if we think that examples for the opposite concept are not reliable, then we should learn the LGS.

2.2 Strategies for Eliminating Learned Contradictions

The learned definitions of the positive and negative concepts may overlap. In this case, we have a contradictory classification for the objective literals[1] in the intersection. In order to resolve the conflict, we must distinguish two types of literals in the intersection: those that belong to the training set and those that do not, also dubbed *unseen* atoms.

In the following we discuss how to resolve the conflict in the case of unseen literals and of literals in the training set. We first consider the case in which the training sets are disjoint, and we later extend the scope to the case where there is a non-empty intersection of the training sets, when they are less than perfect. From now onwards, $\boldsymbol{X}$ stands for a tuple of arguments.

For unseen literals, the conflict is resolved by classifying them as undefined, since the arguments supporting the two classifications are equally strong. Instead, for literals

[1] An 'objective literal' in a Logic Program is just an atom, possibly explicitly negated.

in the training set, the conflict is resolved by giving priority to the classification stipulated by the training set. In other words, literals in a training set that are covered by the opposite definition are considered as *exceptions* to that definition.

Contradiction on Unseen Literals For unseen literals in the intersection, the undefined classification is obtained by making opposite rules mutually defeasible, or "non-deterministic" (see [3, 6]). The target theory is consequently expressed in the following way:

$$p(\boldsymbol{X}) \leftarrow p^+(\boldsymbol{X}), not\ \neg p(\boldsymbol{X}) \qquad\qquad \neg p(\boldsymbol{X}) \leftarrow p^-(\boldsymbol{X}), not\ p(\boldsymbol{X})$$

where $p^+(\boldsymbol{X})$ and $p^-(\boldsymbol{X})$ are, respectively, the definitions learned for the positive and the negative concept, obtained by renaming the positive predicate by p^+ and its explicit negation by p^-. From now onwards, we will indicate with these superscripts the definitions learned separately for the positive and negative concepts.

We want both $p(\boldsymbol{X})$ and $\neg p(\boldsymbol{X})$ to act as an exception to the other. In case of contradiction, this will introduce mutual circularity, and hence undefinedness according to *WFSX*. For each literal in the intersection of p^+ and p^-, there are two stable models, one containing the literal, the other containing the opposite literal.

Contradiction on Examples Theories are tested for consistency on all the literals of the training set, so we should not have a conflict on them. However, in some cases, it is useful to relax the consistency requirement and learn clauses that cover a small amount of counterexamples. This is advantageous when it would be otherwise impossible to learn a definition for the concept, because no clause is contained in the language bias that is consistent, or when an overspecific definition would be learned, composed of many specific clauses instead of a few general ones. In such cases, the definitions of the positive and negative concepts may cover examples of the opposite training set. These must then be considered exceptions, which are then due to abnormalities in the opposite concept.

Let us start with the case where some literals covered by a definition belong to the opposite training set. We want of course to classify these according to the classification given by the training set, by making such literals *exceptions*. To handle exceptions to classification rules, we add a negative default literal of the form $not\ abnorm_p(\boldsymbol{X})$ (resp. $not\ abnorm_{\neg p}(\boldsymbol{X})$) to the rule for $p(\boldsymbol{X})$ (resp. $\neg p(\boldsymbol{X})$), to express possible abnormalities arising from exceptions. Then, for every exception $p(\boldsymbol{t})$, an individual fact of the form $abnorm_p(\boldsymbol{t})$ (resp. $abnorm_{\neg p}(\boldsymbol{t})$) is asserted so that the rule for $p(\boldsymbol{X})$ (resp. $\neg p(\boldsymbol{X})$) does not cover the exception, while the opposite definition still covers it. In this way, exceptions will figure in the model of the theory with the correct truth value. The learned theory thus takes the form:

$$p(\boldsymbol{X}) \leftarrow p^+(\boldsymbol{X}), not\ abnorm_p(\boldsymbol{X}), not\ \neg p(\boldsymbol{X}) \tag{1}$$

$$\neg p(\boldsymbol{X}) \leftarrow p^-(\boldsymbol{X}), not\ abnorm_{\neg p}(\boldsymbol{X}), not\ p(\boldsymbol{X}) \tag{2}$$

3 Explaining Observations and Meta-Learning

After a theory is built it can now be used to analyze observations and to provide explanations for them. Such explanations are sets of abductive hypotheses which, when

assumed true under the theory at hand, yield the observations as conclusions. There can be, of course, many different possible explanations. In the end, most of the times, we want to find the single "best" explanation for the observations, and hence we must have some mechanism to identify the "best" solution among the several alternative ones.

3.1 Abduction

Deduction and abduction differ in the direction in which a rule like "a entails b" is used for inference. Deduction allows deriving b as a consequence of a ; i.e., deduction is the process of deriving the consequences of what is known. Abduction allows deriving a as a hypothetical explanation of b.

3.2 Finding alternative explanations for observations

Trying to find explanations for observations can be implemented by simply finding the alternative abductive models that satisfy both the theory's rules and the observations. The latter can be coded as Integrity Constraints (ICs) which are added to the theory thereby imposing the truthfulness of the observations they describe.

Example 1. Running example

We will use this running example throughout the rest of the chapter.

Consider the following Logic Program consisting of four rules. According to this program a 'professional' is someone who is a regular employee or someone who is a boss in some company. Also, a non-employee is assumed to be a student as well as all those who are junior (all children should go to school!).

$$professional(X) \leftarrow employee(X) \qquad student(X) \leftarrow not\ employee(X)$$
$$professional(X) \leftarrow boss(X) \qquad student(X) \leftarrow junior(X)$$

For now keep this example in mind as we will use it to illustrate the concepts and methods we are about to describe. Assume that '$employee/1$', '$boss/1$', and '$junior/1$' are abducible hypotheses.

Adding one single IC to the theory might yield several alternative 2-valued models (sets of abductive hypotheses) satisfying it, let alone adding several ICs.

In the example above, adding just the Integrity Constraint
'$\bot \leftarrow not\ professional(john)$' — coding the fact that John is a professional — would yield two alternative abductive solutions: $\{employee(john)\}$ and $\{boss(john)\}$.

When the information from several observations comes in at one single time, several ICs must be added to the theory in order to be possible to obtain the right explanations for the corresponding observations.

Our concern is with finding explanations to observations. In a nutshell, we split the set of observations into several smaller subsets; then we create several agents and give each agent the same base theory and a subset of the observations coded as ICs. We then allow each agent to come up with several alternative explanations to its ICs; the explanations need not be minimal sets of hypotheses.

Going back again to our running example, if we also know that John is a student, besides adding the '$\bot \leftarrow \ not\ professional(john)$' IC we must also add the '$\bot \leftarrow not\ student(john)$' IC.

Finding possible alternative explanations is one problem; finding which one(s) is(are) the "best" is another issue. In the next section we assume "best" means minimal set of hypotheses and we describe the method we use to find such best.

3.3 Choosing the best explanation

One well known method for solving complex problems widely used by creative teams is that of 'brainstorming'. In a nutshell, every agent participating in the 'brainstorm' contributes by adding one of his/her ideas to the common idea-pool shared by all the agents. All the ideas, sometimes clashing and oppositional among each other, are then mixed, crossed and mutated. The solution to the problem arises from the pool after a few iterations of this evolutionary process.

The evolution of alternative ideas and arguments in order to find a collaborative solution to a group problem is the underlying inspiration of this work.

Evolutionary Inspiration Darwin's theory is based on the concept of natural selection: only those individuals that are most fit for their environment survive, and are thus able to generate new individuals by means of reproduction. Moreover, during their lifetime, individuals may be subject to random mutations of their genes that they can transmit to offspring. Lamarck's [21] theory, instead, states that evolution is due to the process of adaptation to the environment that an individual performs in his/her life. The results of this process are then automatically transmitted to his/her offspring, via its genes. In other words, the abilities learned during the life of an individual can modify his/her genes.

Experimental evidence in the biological kingdom has shown Darwin's theory to be correct and Lamarck's to be wrong. However, this does not mean that the process of adaptation (or learning) does not influence evolution. Baldwin [4] showed how learning could influence evolution: if the learned adaptations improve the organism's chance of survival then the chances for reproduction are also improved. Therefore there is selective advantage for genetically determined traits that predisposes the learning of specific behaviors. Baldwin moreover suggests that selective pressure could result in new individuals to be born with the learned behavior already encoded in their genes. This is known as the Baldwin effect. Even if there is still debate about it, it is accepted by most evolutionary biologists.

Lamarckian evolution [22] has recently received a renewed attention because it can model cultural evolution. In this context, the concept of "meme" has been developed. A meme is the cognitive equivalent of a gene and it stores abilities learned by an individual during his lifetime, so that they can be transmitted to his offspring.

In the field of genetic programming [20], Lamarckian evolution has proven to be a powerful concept and various authors have investigated the combination of Darwinian and Lamarckian evolution.

In [24] the authors propose a genetic algorithm for belief revision that includes, besides Darwin's operators of selection, mutation and crossover, a logic based Lamarckian

operator as well. This operator differs from Darwinian ones precisely because it modifies a chromosome coding beliefs so that its fitness is improved by experience rather than in a random way. There, the authors showed that the combination of Darwinian and Lamarckian operators are useful not only for standard belief revision problems, but especially for problems where different chromosomes may be exposed to different constraints, as in the case of a multi-agent system. In these cases, the Lamarckian and Darwinian operators play different roles: the Lamarckian one is employed to bring a given chromosome closer to a solution (or even find an exact one) to the current belief revision problem, whereas the Darwinian ones exert the role of randomly producing alternative belief chromosomes so as to deal with unencountered situations, by means of exchanging genes amongst them.

Evolving Beliefs Belief revision is an important functionality that agents must exhibit: agents should be able to modify their beliefs in order to model the outside world. What's more, as the world may be changing, a pool of separately and jointly evolved chromosomes may code for a variety of distinct belief evolution potentials that can respond to world changes as they occur. This dimension has been explored in [24] with specific experiments to that effect. Mark that it is not our purpose to propose here a competitor to extant classical belief revision methods, in particular as they apply to diagnosis. More ambitiously, we do propose a new and complementary methodology, which can empower belief revision — any assumption based belief revision — to deal with time/space distributed, and possibly intermittent or noisy laws about an albeit varying artifact or environment, possibly by a multiplicity of agents which exchange diversified genetically encoded experience. We consider a definition of the belief revision problem that consists in removing a contradiction from an extended logic program by modifying the truth value of a selected set of literals corresponding to the abducible hypotheses. The program contains as well clauses with *falsum* ($\bot$) in the head, representing ICs. Any model of the program must ensure the body of ICs false for the program to be non-contradictory. Contradiction may also arise in an extended logic program when both a literal L and its opposite $\neg L$ are obtainable in the model of the program. Such a problem has been widely studied in the literature, and various solutions have been proposed that are based on abductive logic proof procedures. The problem can be modeled by means of a genetic algorithm, by assigning to each abducible of a logic program a gene in a chromosome. In the simplest case of a two valued revision, the gene will have the value 1 if the corresponding abducible is *true* and the value 0 if the abducible is *false*. The fitness functions that can be used in this case are based on the percentage of ICs that are satisfied by a chromosome. This is, however, an over-simplistic approach since it assumes every abducible is a predicate with arity 0, otherwise a chromosome would have as many genes as the number of all possible combinations of ground values for variables in all abducibles.

Specific Belief Evolution Method In multi-agent joint belief revision problems, agents usually take advantage of each other's knowledge and experience by explicitly communicating messages to that effect. In our approach, however, we introduce a new and complementary method (and some variations of it), in which we allow knowledge and

experience to be coded as genes in an agent. These genes are exchanged with those of other agents, not by explicit message passing but through the crossover genetic operator. Crucial to this endeavor, a logic-based technique for modifying cultural genes, i.e. memes, on the basis of individual agent experience is used.

The technique amounts to a form of belief revision, where a meme codes for an agent's belief or assumptions about a piece of knowledge, and which is then diversely modified on the basis of how the present beliefs may be contradicted by laws (expressed as ICs). These mutations have the effect of attempting to reduce the number of unsatisfied constraints. Each agent possesses a pool of chromosomes containing such diversely modified memes, or alternative assumptions, which cross-fertilize Darwinianly amongst themselves. Such an experience in genetic evolution mechanism is aptly called Lamarckian.

Since we will subject the sets of beliefs to an evolutionary process (both Darwinian and Lamarckian) we will henceforth refer to this method as "Belief Evolution" (BE) instead of the classical "Belief Revision" (BR).

General Description of the Belief Evolution Method Each agent keeps a population of chromosomes and finds a solution to the BE problem by means of a genetic algorithm. We consider a formulation of the distributed BE problem where each agent has the same set of abducibles and the same program expressed theory, but is exposed to possibly different constraints. Constraints may vary over time, and can differ because agents may explore different regions of the world. The genetic algorithm we employ allows each agent to cross over its chromosomes with chromosomes from other agents. In this way, each agent can be prepared in advance for situations that it will encounter when moving from one place to another.

The algorithm proposed for BE extends the standard genetic algorithm in two ways:

- crossover is performed among chromosomes belonging to different agents,
- a Lamarckian operator called Learn is added in order to bring a chromosome closer to a correct revision by changing the value of abducibles

The Structure of a Chromosome In BR and BE, each individual hypothesis is described by the truth value of all the abducibles. In our present setting, however, each gene encodes a ground literal, i.e., all its variables are bound to fixed values.

So, we represent a chromosome as a list of genes and memes, and different chromosomes may contain information about different genes. This implies a major difference to traditional genetic algorithms where every chromosome refers exactly to the same genes and the crossover and mutation operations are somewhat straightforward.

The memes in a chromosome will be just like genes — representing abducibles — but they will have extra information. Each meme has associated with it a counter keeping record of how many times the meme has been confirmed or refuted. Each time a meme is confirmed this value is increased, and each time it is refuted the value decreases. This value provides thus a measure of confidence in the corresponding meme.

Example 2. Running example (cont.)

Continuing with our running example, let us assume that both $professional(john)$ and $student(john)$ have been observed. We can create two agents, each with the same rule-set theory, and split the observations among them. We would have thus

Agent 1:	Agent 2:
$\leftarrow\ not\ professional(john)$	$\leftarrow\ not\ student(john)$

Agent 1 and Agent 2:

$$\begin{aligned} professional(X) &\leftarrow employee(X) \\ professional(X) &\leftarrow boss(X) \\ student(X) &\leftarrow not\ employee(X) \\ student(X) &\leftarrow junior(X) \end{aligned}$$

In the simplest case where a gene encodes an abductive ground literal Agent 1 would come up with two alternative abductive solutions for its IC '$\bot \leftarrow\ not\ professional(john)$': $\{employee(john)\}$ and $\{boss(john)\}$. Moreover, Agent 2 would come up with two other alternative abductive solutions for its IC '$\bot \leftarrow not\ student(john)$': $\{not\ employee(john)\}$ and $\{junior(john)\}$.

Crossover Since each agent knows only some ICs the abductive answer the algorithm seeks should be a combination of the partial answers each agent comes up with. In principle, the overlap on abducibles among two chromosomes coming from different agents should be less than total — after all, each agent is taking care of its own ICs which, in principle, do not refer to the exact same abducibles. Therefore, crossing over such chromosomes can simply turn out to be the merging of the chromosomes, i.e., the concatenation of the lists of abducibles.

If several ICs refer to the exact same abducibles the chromosomes from different agents will contain either the same gene — in which case we can see this as an 'agreement' between the agents as far as the corresponding abducible is concerned — or genes stating contradictory information about the same abducible. In this last case if the resulting concatenated chromosome turns out to be inconsistent in itself the fitness function will filter it out by assigning it a very low value.

Example 3. Running example (cont.)

Continuing with our running example, recall that Agent 1 would come up with two alternative abductive solutions for its IC '$\bot \leftarrow\ not\ professional(john)$': $\{employee(john)\}$ and $\{boss(john)\}$. Moreover, Agent 2 would come up with two other alternative abductive solutions for its IC '$\bot \leftarrow not\ student(john)$': $\{not\ employee(john)\}$ and $\{junior(john)\}$.

The crossing over of these chromosomes will yield the four combinations $\{employee(john), not\ employee(john)\}$, $\{employee(john), junior(john)\}$, $\{boss(john), not\ employee(john)\}$, and $\{boss(john), junior(john)\}$.

The first resulting chromosome is contradictory so it will be filtered out by the fitness function. The second chromosome correspond to the situation where John is a junior employee who is still studying — a quite common situation, actually. The third chromosome corresponds to the situation where John is a senior member of a company — a 'boss' — who is taking some course (probably a post-graduation study). The last

chromosome could correspond to the situation of a young entrepreneur who, besides owning his/hers company, is also a student — this is probably an uncommon situation and, if necessary, the fitness function can reflect that "unprobability".

Mutation When considering a list of abducible literals the mutation operation resembles the standard mutation of genetic algorithms by changing one gene to its opposite; in this case negating the truth value of the abducted literal.

Example 4. Running example (cont.)

In the example we have been using this could correspond to mutating the chromosome $\{not\ employee(john)\}$ to $\{employee(john)\}$, or to mutating the chromosome $\{junior(john)\}$ to $\{not\ junior(john)\}$.

The Lamarckian $Learn$ operator The Lamarckian operator Learn can change the values of variables of an abducible in a chromosome c_i so that a bigger number of constraints is satisfied, thus bringing c_i closer to a solution. Learn differs from a normal belief revision operator because it does not assume that all abducibles are false by CWA before the revision but it starts from the truth values that are given by the chromosome c_i. Therefore, it has to revise the values of variables of some abducibles and, in the particular case of an abducible without variables, from *true* to *false* or from *false* to *true*. This Lamarckian $Learn$ operator will introduce an extra degree of flexibility allowing for changes to a chromosome to induce the whole belief evolution algorithm to search a solution considering new values for variables.

In the running example this could correspond, for example, to changing the chromosome $\{junior(john)\}$ to $\{junior(mary)\}$, where 'mary' is another value in the domain range of the variable for abducible $junior/1$.

The Fitness Functions Various fitness functions can be used in belief revision. The simplest fitness function is the following $Fitness(c_i) = (n_i/n)/(1 + NC)$, where n_i is the number of integrity constraints satisfied by chromosome c_i, n is the total number of integrity constraints, and NC is the number of contradictions in chromosome c_i. We will call it an accuracy fitness function.

4 Argumentation

In [12], the author shows that preferred maximal scenarios (with maximum default negated literals — the hypotheses) are always guaranteed to exist for NLPs; and that when these yield 2-valued complete (total), consistent, admissible scenarios, they coincide with the Stable Models of the program. However, preferred maximal scenarios are, in general, 3-valued. The problem we address now is how to define 2-valued complete models based on preferred maximal scenarios. In [31] the authors took a step further from what was achieved in [12], extending its results. They did so by completing a preferred set of hypotheses rendering it approvable, ensuring whole model consistency and 2-valued completeness.

The resulting semantics thus defined, dubbed Approved Models [31], is a conservative extension to the widely known Stable Models semantics [14] in the sense that every

Stable Model is also an Approved Model. The Approved Models are guaranteed to exist for every Normal Logic Program, whereas Stable Models are not. Concrete examples in [31] show how NLPs with no Stable Models can usefully model knowledge, as well as produce additional models. Moreover, this guarantee is crucial in program composition (say, from knowledge originating in divers sources) so that the result has a semantics. It is important too to warrant the existence of semantics after external updating, or in Stable Models based self-updating [1].

For the formal presentation and details of the Approved Models semantics see [31].

4.1 Intuition

Most of the ideas and notions of argumentation we are using here come from the Argumentation field — mainly from the foundational work of Phan Minh Dung in [12]. In [31] the *Reductio ad Absurdum* reasoning principle is also considered. This has been studied before in [29], [30], and [32]. In this paper we consider an *argument* (or set of hypotheses) as a set S of abducible literals of a NLP P.

We have seen before examples of Extended Logic Programs — with explicit negation. In [8] the authors show that a simple syntactical program transformation applied to an ELP produces a Normal Logic Program with Integrity Constraints which has the exact same semantics as the original ELP.

Example 5. Transforming an ELP into a NLP with ICs

Taking the program

$$\begin{aligned} dangerous\_neighborhood &\leftarrow not\ \neg dangerous\_neighborhood \\ \neg dangerous\_neighborhood &\leftarrow not\ dangerous\_neighborhood \end{aligned}$$

we just transform the explicitly negated literal $\neg dangerous\_neighborhood$ into the positive literal $dangerous\_neighborhood^-$, and the original $dangerous\_neighborhood$ literal is converted into $dangerous\_neighborhood^+$

Now, in order to ensure consistency, we just need to add the IC $\bot \leftarrow dangerous\_neighborhood^+, dangerous\_neighborhood^-$. The resulting transformed program is

$$\begin{aligned} dangerous\_neighborhood^+ &\leftarrow not\ dangerous\_neighborhood^- \\ dangerous\_neighborhood^- &\leftarrow not\ dangerous\_neighborhood^+ \\ \bot &\leftarrow dangerous\_neighborhood^+, dangerous\_neighborhood^- \end{aligned}$$

Now know that we can just consider NLPs with ICs without loss of generality and so, henceforth, we will assume just that case. NLPs are in fact the kind of programs most Inductive Logic Programming learning systems produce.

4.2 Assumptions and Argumentation

Previously, we have seen that assumptions can be coded as abducible literals in Logic Programs and that those abducibles can be packed together in chromosomes. The evolutionary operators of genetic and memetic crossover, mutation and fitness function

applied to the chromosomes provide a means to search for a consensus of the initial assumptions since it will be a consistent mixture of these.

Moreover, the 2-valued contradiction removal method presented in subsection 2.2 is a very superficial one. That method removes the contradiction between $p(\boldsymbol{X})$ and $\neg p(\boldsymbol{X})$ by forcing a 2-valued semantics for the ELP to choose either $p(\boldsymbol{X})$ or $\neg p(\boldsymbol{X})$ since they now are exceptions to one another. It is a superficial removal of the contradiction because the method does not look into the reasons why both $p(\boldsymbol{X})$ and $\neg p(\boldsymbol{X})$ hold simultaneously. The method does not go back to find the underlying assumptions supporting both $p(\boldsymbol{X})$ and $\neg p(\boldsymbol{X})$ to find out which assumptions should be revised in order to restore overall consistency. Any one such method must fall back into the principles of argumentation: to find the arguments supporting one conclusion in order to prevent it if it leads to contradiction.

4.3 Collaborative Opposition

In [12] the author shows that the Stable Models of a NLP coincide with the 2-valued complete Preferred Extensions which are self-corroborating arguments.

The more challenging environment of a Semantic Web is one possible 'place' where the future intelligent systems will live in. Learning in 2-values or in 3-values are open possibilities, but what is most important is that knowledge and reasoning will be shared and distributed. Different opposing concepts and arguments will come from different agents. It is necessary to know how to conciliate those opposing arguments, and how to find 2-valued consensus as much as possible instead of just keeping to the least-commitment 3-valued consensus. In [31] the authors describe another method for finding such 2-valued consensus in an incremental way. In a nutshell, we start by merging together all the opposing arguments into a single one. The conclusions from the theory plus the unique merged argument are drawn and, if there are contradictions against the argument or contradictions inside the argument we non-deterministically choose one contradicted assumption of the argument and revise its truth value. The iterative repetition of this step eventually ends up in a non-contradictory argument (and all possibilities are explored because there is a non-deterministic choice).

In a way, the evolutionary method we presented in subsection 3.3 implements a similar mechanism to find the consensus non-contradictory arguments.

5 Conclusions

The two-valued setting that has been adopted in most work on ILP and Inductive Concept Learning in general is not sufficient in many cases where we need to represent real world data.

Standard ILP techniques can be adopted for separate agents to learn the definitions for the concept and its opposite. Depending on the adopted technique, one can learn the most general or the least general definition and combine them in different ways.

We have also presented an evolution-inspired algorithm for performing belief revision in a multi-agent environment. The standard genetic algorithm is extended in two

ways: first the algorithm combines two different evolution strategies, one based on Darwin's and the other on Lamarck's evolutionary theory and, second, chromosomes from different agents can be crossed over with each other. The Lamarckian evolution strategy is obtained be means of an operator that changes the genes (or, better, the memes) of agents in order to improve their joint fitness.

We have presented too a new and productive way to deal with oppositional concepts in a cooperative perspective, in different degrees. We use the contradictions arising from opposing agents' arguments as hints for the possible collaborations. In so doing, we extend the classical conflictual argumentation giving a new treatment and new semantics to deal with the contradictions.

References

1. Alferes, J. J., Brogi, A., Leite, J. A., and Pereira, L. M. Evolving logic programs. In S. Flesca et al., editor, *JELIA*, volume 2424 of *LNCS*, pages 50–61. Springer, 2002.
2. Alferes, J. J., Damásio, C. V., and Pereira, L. M. (1994). SLX - A top-down derivation procedure for programs with explicit negation. In Bruynooghe, M., editor, *Proc. Int. Symp. on Logic Programming*. The MIT Press.
3. Alferes, J. J. and Pereira, L. M. (1996). *Reasoning with Logic Programming*, volume 1111 of *LNAI*. Springer-Verlag.
4. http://www.psych.utoronto.ca/museum/baldwin.htm
5. Baral, C., Gelfond, M., and Rushton, J. Nelson. Probabilistic reasoning with answer sets. In Vladimir Lifschitz and Ilkka Niemelä, editors, *LPNMR*, volume 2923 of *Lecture Notes in Computer Science*, pages 21–33. Springer, 2004.
6. Baral, C. and Gelfond, M. (1994). Logic programming and knowledge representation. *Journal of Logic Programming*, 19/20:73–148.
7. Damásio, C. V., Nejdl, W., and Pereira, L. M. (1994). REVISE: An extended logic programming system for revising knowledge bases. In Doyle, J., Sandewall, E., and Torasso, P., editors, *Knowledge Representation and Reasoning*, pages 607–618. Morgan Kaufmann.
8. Damásio, C. V. and Pereira, L. M. Default Negated Conclusions: Why Not?. In *ELP'96*, pages 103–117. Springer, 1996
9. De Raedt, L. and Bruynooghe, M. (1989). Towards friendly concept-learners. In *Procs. of the 11th Intl. Joint Conf. on Artificial Intelligence*, pages 849–856. Morgan Kaufmann.
10. De Raedt, L. and Bruynooghe, M. (1990). On negation and three-valued logic in interactive concept learning. In *Procs. of the 9th European Conf. on Artificial Intelligence*.
11. De Raedt, L. and Bruynooghe, M. (1992). Interactive concept learning and constructive induction by analogy. *Machine Learning*, 8(2):107–150.
12. Dung, P. M. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–358, 1995.
13. Esposito, F., Ferilli, S., Lamma, E., Mello, P., Milano, M., Riguzzi, F., and Semeraro, G. (1998). Cooperation of abduction and induction in logic programming. In Flach, P. A. and Kakas, A. C., editors, *Abductive and Inductive Reasoning*, Pure and Applied Logic. Kluwer.
14. Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. In Kowalski, R. and Bowen, K. A., editors, *Procs. of the 5th Int. Conf. on Logic Programming*, pages 1070–1080. MIT Press.
15. Gordon, D. and Perlis, D. (1989). Explicitly biased generalization. *Computational Intelligence*, 5(2):67–81.
16. Green, D.M., and Swets, J.M. (1966). Signal detection theory and psychophysics. New York: John Wiley and Sons Inc.. ISBN 0-471-32420-5.

17. Greiner, R., Grove, A. J., and Roth, D. (1996). Learning active classifiers. In *Procs. of the Thirteenth Intl. Conf. on Machine Learning (ICML96)*.
18. Inoue, K. (1998). Learning abductive and nonmonotonic logic programs. In Flach, P. A. and Kakas, A. C., editors, *Abductive and Inductive Reasoning*, Pure and Applied Logic. Kluwer.
19. Inoue, K. and Kudoh, Y. (1997). Learning extended logic programs. In *Procs. of the 15th Intl. Joint Conf. on Artificial Intelligence*, pages 176–181. Morgan Kaufmann.
20. Koza, J.R. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection MIT Press
21. Jean Baptiste Lamarck: http://www.ucmp.berkeley.edu/history/lamarck.html
22. Grefenstette, J. J., (1991). Lamarckian learning in multi-agent environments
23. Lamma, E., Riguzzi, F., and Pereira, L. M. (1988). Learning in a three-valued setting. In *Procs. of the Fourth Intl. Workshop on Multistrategy Learning*.
24. Lamma, E., Pereira, L. M., and Riguzzi, F. Belief revision via lamarckian evolution. *New Generation Computing*, 21(3):247–275, August 2003.
25. Lamma, E., Riguzzi, F., and Pereira, L. M. Strategies in combined learning via logic programs. *Machine Learning*, 38(1-2):63–87, January 2000.
26. Lapointe, S. and Matwin, S. (1992). Sub-unification: A tool for efficient induction of recursive programs. In Sleeman, D. and Edwards, P., editors, *Procs. of the 9th Intl. Workshop on Machine Learning*, pages 273–281. Morgan Kaufmann.
27. Lavrač, N. and Džeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood.
28. Pazzani, M. J., Merz, C., Murphy, P., Ali, K., Hume, T., and Brunk, C. (1994). Reducing misclassification costs. In *Procs. of the Eleventh Intl. Conf. on Machine Learning (ML94)*, pages 217–225.
29. Pereira, L. M. and Pinto, A. M. Revised stable models - a semantics for logic programs. In G. Dias et al., editor, *Progress in AI*, volume 3808 of *LNCS*, pages 29–42. Springer, 2005.
30. Pereira, L. M. and Pinto, A. M. Reductio ad absurdum argumentation in normal logic programs. In *Argumentation and Non-monotonic Reasoning (ArgNMR'07) workshop at LPNMR'07*, pages 96–113, 2007.
31. Pereira, L. M. and Pinto, A. M. Approved Models for Normal Logic Programs. In *LPAR*, pages 454–468. Springer, 2007.
32. Pinto, A. M. Explorations in revised stable models — a new semantics for logic programs. Master's thesis, Universidade Nova de Lisboa, February 2005.
33. Provost, F. J. and Fawcett, T. (1997). Analysis and visualization of classifier performance: Comparison under imprecise class and cost distribution. In *Procs. of the Third Intl. Conf. on Knowledge Discovery and Data Mining (KDD97)*. AAAI Press.
34. Quinlan, J. (1990). Learning logical definitions from relations. *Machine Learning*, 5:239–266.
35. Vere, S. A. (1975). Induction of concepts in the predicate calculus. In *Procs. of the Fourth Intl. Joint Conf. on Artificial Intelligence (IJCAI75)*, pages 281–287.
36. Whitley, D., Rana, S., and Heckendorn, R.B. (1998) The Island Model Genetic Algorithm: On Separability, Population Size and Convergence