Journal of Artificial Intelligence Research 1 (1993) 1-15 Submitted 6/91; published 9/91

Belief Revision via Lamarckian Evolution

Evelina Lamma ELAMMA@ING.UNIFE.IT
Università di Ferrara, Via Saragat 1,
44100 Ferrara, Italy,

Luís Moniz Pereira LMP@DI.FCT.UNL.PT
Centro de Inteligência Artificial (CENTRIA),
Faculdade de Ciências e Tecnologia,
Universidade Nova de Lisboa,
2829-516 Caparica, Portugal **Fabrizio Riguzzi** FRIGUZZI@ING.UNIFE.IT
Università di Ferrara, Via Saragat 1,
44100 Ferrara, Italy,

Abstract

We propose a genetic algorithm to accomplish belief revision. The algorithm implements a new evolutionary strategy resulting from a combination of Darwinian and Lamarckian approaches. Besides encompassing the Darwinian operators of selection, mutation and crossover, it comprises a Lamarckian operator that mutates the genes in a chromosome that code for the believed assumptions. These self mutations are performed as a consequence of the chromosome phenotype's experience obtained while solving a belief revision problem. They are directed by a belief revision procedure which relies on tracing the logical derivations leading to inconsistency of belief, so as to remove the latter's support on the gene coded assumptions, by mutating the genes.

The algorithm, with and without the Lamarckian operator, has been tested on a number of belief revision problems, and results show that the addition of the Lamarckian operator improves the efficiency of the algorithm.

Additionally, the combination of Darwinian and Lamarckian operators is useful not just for standard belief revision problems, but for problems where different chromosomes may be exposed to different constraints and observations. In these cases, the Lamarckian and Darwinian operators play different rôles: the Lamarckian one is employed to bring a given chromosome closer to a solution (or even to find an exact one) to the current belief revision problem, whereas the Darwinian ones exert the rôle of randomly producing alternative chromosomes in order to deal with unencountered situations, by means of exchanging genes amongst them.

We also tested this hypothesis, with positive results, for the case of a multi-agent belief revision setting.

1. Introduction

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 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 learnt 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, although the more recently evolved concept of "meme" supports *cultural* Lamarckian evolution (cf. [Bla99]). However, in the field of genetic programming, Lamarckian evolution has proven to be a powerful concept and various authors have investigated the combination of Darwinian and Lamarckian evolution [HB96, AL94, LTG96, Gre91].

Herein, we propose a genetic algorithm for belief revision that includes, besides Darwin's operators of selection, mutation and crossover [Mit97], 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 random way.

We venture that the combination of Darwinian and Lamarckian operators will be useful not only for standard belief revision problems, but especially for problems where different chromosomes may be exposed to different constraints and environmental observations. In these cases, the Lamarckian and Darwinian operators play different rôles: 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 rôle of randomly producing alternative belief chromosomes so as to deal with unencountered situations, by means of exchanging genes amongst them.

We tested this hypothesis on multi-agent joint belief revision problems. In such a distributed setting, 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, 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. The new offspring agent chromosomes can be naturally selected according to their gene coded knowledge governing their behaviour.

Crucial to this endeavour, we introduce a logic-based technique for modifying cultural genes, i.e. memes, on the basis of individual agent experience. The technique amounts to a form of belief revision, where a meme codes for an agent's belief or assumption about a piece of knowledge, and which is then diversely modified on the basis of how the present beliefs may be contradicted by observations and laws (expressed as integrity constraints). These self mutations are indeed performed as the outcome of the chromosome phenotype's (i.e. agent's) experience while solving a belief revision problem. They are directed by a belief revision procedure, which relies on tracing the logical derivations leading to inconsistency of belief, so as to remove the latter's support on gene coded assumptions by mutating the memes involved. Each agent possesses a pool of chromosomes containing such diversely modified memes, or alternative assumptions, which cross-fertilize Darwinianly amongst themselves. Such an experience-influenced genetic evolution mechanism is aptly called Lamarckian.

To illustrate how these mechanisms, of individual agent Lamarckian evolution and of Darwinian agent genetics, can jointly lead to improved single agent population behaviour in colaborative problem-solving, we apply them to distributed model-based diagnosis, a natural domain in which belief revision techniques apply [DPS97], and to the n-queen constraint satisfaction problem. But this is just illustrative. 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, though we have not yet delved into exploring this dimension.

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 revison – any assumption based belief revision – to deal with time/space distributed, and possibly intermitent or noisy observations about an albeit varying artifact or environment, possibly by a multiplicity of agents which exchange diversified genetically encoded experience.

Belief revision is indeed an important functionality that agents must exhibit: agents should be able to modify their beliefs in order to model the outside world. We consider a definition of the belief revision problem that consists in removing a contradiction from an extended logic program [PA92, AP96, APP98] by modifying the truth value of a selected set of literals called *revisables*. The program contains as well clauses with false ($\perp$) in the head, representing *integrity constraints*. Any model of the program must ensure the body of integrity constraints 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 [APS99, DPS97] that are based on abductive logic proof procedures.

The problem can be modeled by means of a genetic algorithm, by assigning to each revisable of a logic program a gene in a chromosome. In the case of a two valued revision, the gene will have the value 1 if the corresponding revisable is true and the value 0 if the revisable is false. The fitness functions that can be used in this case are based on the percentage of integrity constraints that are satisfied by a chromosome.

Each agent keeps a population of chromosomes and finds a solution to the revision problem by means of a genetic algorithm. We consider a formulation of the revision problem where each agent has the same set of revisables and the same program expressed theory, but is subjected to possibly different observations and constraints. Observations and 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.

We make our case by experimentally showing that logic-aided multi-agent Lamarckian learning leads to improved agent behaviour with respect to classical Darwinian genetics. The experimental domain we address is that of diagnosing digital circuits.

The paper is organized as follows. We first review some logic programming fundamentals, and give a definition of the belief revision problem in section 2. Then we describe the algorithm together with the Lamarckian operator in section 3. The results of experiments with the algorithm are shown in section 4. We examine related works in section 5, and draw conclusions in section 6.

2. Logic Programming Basis

In this section we first provide some logic programming fundamentals, and then we give a definition of the belief revision problem adapted from [PDA93].

2.1 Language

Given a first order language *Lang*, an extended logic program [PA92, AP96, APP98] is a set of rules and integrity constraints of the form

$$H \leftarrow B_1, \ldots, B_n, not\, C_1, \ldots, not\, C_m \quad (m \geq 0, n \geq 0)$$

where $H, B_1, \ldots, B_n, C_1, \ldots, C_m$ are objective literals, and in integrity constraints H is $\perp$ (false). An objective literal is either an atom A or its explicit negation $\neg A$, where $\neg\neg A = A$. $not\, L$ is called a default or negative literal. Literals are either objective or default ones. The default complement of objective literal L is $not\, L$, and of default literal $not\, L$ is L. A rule stands for all its ground instances with respect to *Lang*. The notation $H \leftarrow \mathcal{B}$ is also used to represent a rule, where the set $\mathcal{B}$ contains the literals in its body. For every pair of objective literals $\{L, \neg L\}$ in *Lang*, we implicitly assume the constraint $\perp \leftarrow L, \neg L$.

The set of all objective literals of a program P is called its *extended Herbrand base* and is represented as $H^E(P)$.

We consider the Extended Well Founded Semantics (*WFSX*) that extends the well founded semantics (*WFS*) [VGRS91] for normal logic programs to programs extended with explicit negation, besides the implicit or default negation of normal programs. *WFSX* is obtained from *WFS* by adding the coherence principle (CP) relating the two forms of negation: "*if L is an objective literal and ¬L belongs to the model of a program, then also not L belongs to the model*", i.e., $\neg L \rightarrow not\ L$. See [AP96, DPP97] or the Appendix for a definition of *WFSX*.

We say that a set of literals S is *contradictory* iff $\perp \in S$. The paraconsistent version of *WFSX*, that allows models to contain the atom $\perp$, is called *WFSXp* [DP97, DP98].

2.2 Revising Contradictory Extended Logic Programs

Extended logic programs are liable to be contradictory because of integrity constraints, either those that are user-defined or those of the form $\perp \leftarrow L, \neg L$ that are implicitely assumed. Let us see an example of a contradictory program.

Example 2.1 Consider $P = \{a; \perp \leftarrow a, not\, b\}$[1]. Since we have no rules for b, by the Closed World Assumption *CWA*, it is natural to accept $not\, b$ as true. However, because of the integrity constraint, we can conclude $\perp$ and thus have contradiction.

It is arguable that the *(CWA)* may not be held of atom b since it leads to contradiction. Revising such *CWAs* is the basis of the contradiction removal method of [PDA93]. In order to select a particular contradiction removal process, three questions must be answered:

1. For which literals is revision of their truth-value allowed ?

1. $\perp \leftarrow a, \neg a$ and $\perp \leftarrow b, \neg b$ are implicitly assumed.

2. To what truth values do we change the revisable literals ?
3. How to choose among possible revisions ?

The options taken here are clarified in the discussion in section 2.4, giving two different answers to these questions. Both use the same criteria to answer 1 and 3, but differ on the second one. For example 2.1 the first way of removing contradiction gives $\{a, not\ \neg a, not\ \neg b\}$ as the intended meaning of P, where b is revised to *undefined*, achievable by adding $b \leftarrow not\ b$ to P. The second gives $\{a, b, not\ \neg a, not\ \neg b\}$, by revising b to true, achievable by adding b to P.

2.3 Contradictory Well Founded Model

To revise contradictions, we need to identify the contradictory sets of consequences implied by the applications of CWA. The main idea is to compute all consequences of the program, even those leading to contradictions, as well as those arising from contradictions. Furthermore, the coherence principle is enforced at each step.

Example 2.2 Consider program P:

$$\texttt{a}\leftarrow\texttt{not b.}\ \text{(i)} \qquad \neg\texttt{a}\leftarrow\texttt{not c.}\ \text{(ii)} \qquad \texttt{d}\leftarrow\texttt{a.}\ \text{(iii)} \qquad \texttt{e}\leftarrow\neg\texttt{a.}\ \text{(iv)}$$

1. *not* b and *not* c hold since there are no rules for either b or c.
2. $\neg a$ and a hold from 1 and rules (i) and (ii).
3. $\perp$ holds from 2 and implicit constraint $\leftarrow a, \neg a$.
4. *not* a and *not* $\neg a$ hold from 2 and inference rule (CP).
5. d and e hold from 2 and rules (iii) and (iv).
6. *not* d and *not* e hold from 4 and rules (iii) and (iv), as they are the only rules for d and e.
7. *not* $\neg d$ and *not* $\neg e$ hold from 5 and inference rule (CP).

The whole set of consequences is the $WFSXp$model:

$$\{\perp, \neg a, a, not\ a, not\ \neg a, not\ b, not\ c, d, not\ d, not\ \neg d, e, not\ e, not\ \neg e\}$$

2.4 Contradiction Removal Sets

To abolish contradiction, the first issue to consider is which default literals true by CWA are allowed to change their truth values. We adopt the approach of [PDA93] where the candidates for revision are all the objective literals that have no rules in the program. By CWA, their default negation is true. These literals are called *revisables*.

Definition 2.1 (Revisables) *The revisables of a program P are the elements of a chosen subset $Rev(P)$, of the set of all objective literals L having no rules for them in P.*

The revisables thus are objective literals that do not appear in rule heads but only in rule bodies, either in a positive or default form. By the *CWA*, every revisable R is false, i.e., *not* R is true. Now we identify the revisables that have to be revised to true or undefined in order to restore consistency. These are the ones that support contradiction. Intuitively, a support of a literal consists of the revisable literals in the leaves of a derivation for it in the *WFSXp* model.

Definition 2.2 (Set of assumptions supporting a literal) *A support set (of assumptions) of a literal L of the* WFSXp *model M_P of a program P, denoted by $SS(L)$, with respect to the set of revisable $Rev(P)$ is obtained as follows:*

1. *If L is an objective literal in M_P then for each rule $L \leftarrow \mathcal{B}$ in P, such that $\mathcal{B} \subseteq M_P$ there is one $SS(L)$ formed by the union of a SS for each $B_i \in \mathcal{B}$. If $\mathcal{B}$ is empty then $SS(L) = \{\}$.*
2. *If L is a default literal not $A \in M_P$:*
   (a) *if no rules for A exist in P then a support set of L is $\{not\ A\}$.*
   (b) *if rules for A exist in P that have a non-empty body, then choose from each such rule a single literal such that its default complement belongs to M_P. There exists one SS for not A which is the union of one SS for the default complement of the chosen literal in each rule.*
   (c) *if $\neg A$ belongs to M_P then there exist,* additionally*, support sets SS for not A equal to each $SS(\neg A)$.*

The definitions of revisable literals and of support sets differ from those given in [PDA93] because there the revisables are the default complement of the literals without definition and support sets there contain all the literals in the nodes of a derivation for L. We have provided these modified definitions because they simplify the introduction of the Lamarckian operator in the next section.

Example 2.3 The *WFSXp* model M_P of:

```
¬p←not c.   p←t.          b←c, a.   ¬b←not e.   a.
            p←a,not b.    b←d.
```

is$\{a, not\ \neg a, not\ b, \neg b, not\ c, not\ \neg c, not\ d, not\ \neg d, not\ e, not\ \neg e, not\ t,$
$not\ \neg t, p, \neg p, not\ p, not\ \neg p, \bot\}$.

Here the revisables are $\{c, d, e\}$. There are two support sets for *not b*:

$$SS_1(not\ b) = SS(not\ c) \cup SS(not\ d) \qquad \textit{by rule 2b}$$
$$SS_1(not\ b) = \{not\ c\} \cup \{not\ d\} = \{not\ c, not\ d\} \qquad \textit{by rule 2a}$$

Notice that the other possibility of choosing literals for $SS(not\ b)$, i.e. $SS_1(not\ b) = SS(not\ a) \cup SS(not\ d)$, can't be considered because *not a* doesn't belong to M_P. The other support set for *not b* is obtained using rule 2c:

$$
\begin{array}{lll}
SS_2(not\ b) = SS(\neg b) & by\ rule\ 2c \\
SS_2(not\ b) = SS(not\ e) & by\ rule\ 1 \\
SS_2(not\ b) = \{not\ e\} & by\ rule\ 2a
\end{array}
$$

Now the support sets for the objective literal p are easily computed:

$$
\begin{array}{ll}
SS(p)=SS(a)\cup SS(not\ b) & by\ rule\ 1 \\
SS(p)=\{\}\cup SS(not\ b) & by\ rule\ 1\ (the\ only\ rule\ for\ a\ is\ fact\ a)
\end{array}
$$

So $SS_1(p) = SS1(not\ b) = \{not\ c, not\ d\}$ and $SS_2(p) = SS_2(not\ b) = \{not\ e\}$. $\neg p$ has the unique support set $\{not\ c\}$. Consequently, because contradiction is obtained only via $\perp \leftarrow p, \neg p$, $SS_1(\perp) = \{not\ c, not\ d\}$ and $SS_2(\perp) = \{not\ e, not\ c\}$.

Proposition 2.1 (Existence of support sets) *Every literal L belonging to the* WFSXp *model of a program P has at least one support set $SS(L)$.*

We define a spectrum of possible revisions using the notion of hitting set:

Definition 2.3 (Hitting set) *A hitting set of a collection C of sets is formed by the union of one non-empty subset from each $S \in C$. A hitting set is minimal iff no proper subset is a hitting set. If $\{\} \in C$, then C has no hitting sets.*

Definition 2.4 (Removal set) *A removal set of a literal L of a program P is a hitting set of all support sets $SS(L)$.*

We can revise contradictory programs by changing the truth value of the literals of some removal set of $\perp$. The truth value can be changed either to undefined or false. It can be changed to undefined by adding, for each literal *not* L in the removal set, the inhibition rule $L \leftarrow not\ L$ to P (making L effectively undefined), while it can be changed to false by adding L to P. In case the literals are revised to undefined, then the contradiction is removed and no new contradiction can arise. In case they are revised to false, a new contradiction may arise and therefore this (convergent) contradiction removal process must be iterated. This defines the possible revisions of a contradictory program.

We answer the second question above by considering only a two-valued revisions, i.e. where the truth value of a revisable can only be changed to true or false. We answer the third question by prefering to revise minimal sets of revisables:

Definition 2.5 (Contradiction removal set) *A contradiction removal set (CRS) of P is a minimal removal set of $\perp$.*

Example 2.3 (cont.) The support sets of $\perp$ are $\{not\ c, not\ d\}$ and $\{not\ c, not\ e\}$. Its removal sets are (RS_1 and RS_4 being minimal):

$$
\begin{array}{ll}
RS_1(\perp, R) = \{not\ c\} & RS_4(\perp, R) = \{not\ d, not\ e\} \\
RS_2(\perp, R) = \{not\ c, not\ e\} & RS_5(\perp, R) = \{not\ c, not\ d, not\ e\} \\
RS_3(\perp, R) = \{not\ c, not\ d\} &
\end{array}
$$

Definition 2.6 (Revisable program) *A program is revisable iff it has a contradiction removal set.*

The $CRSs$ are minimal hitting sets of the collection of support sets of $\perp$. In [PDA93] an algorithm for computing the $CRSs$ is presented.

3. A genetic algorithm for multi-agent belief revision

The algorithm here proposed for belief revision extends the standard genetic algorithm (described for example in [Mit97]) 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 correct revision by changing the value of revisables.

Each agent executes the following algorithm:

GA($Fitness, max\_gen, p, r, m, l$)
  $Fitness$: a function that assigns an evaluation score
    to a hypothesis coded as a chromosome
  max_gen : the maximum number of generations
    before termination
  p: the number of individuals in the population
  r: the fraction of the population to be replaced by Crossover
    at each step
  m: the fraction of the population to be mutated
  l: the fraction of the population that should evolve Lamarckianly

  Initialize population: $P \leftarrow$ generate p hypotheses at random
  Evaluate: for each h in P, compute $Fitness(h)$
  $gen \leftarrow 0$
  While $gen \leq max\_gen$
  Create a new population P_s:
    $Select$: Probabilistically select $(1-r)p$ members of P
      to add to P_s. The probability $Pr(h_i)$ of selecting
      hypothesis h_i from P is given by
      $Pr(h_i) = \frac{Fitness(h_i)}{\Sigma_{j=1}^{p} Fitness(h_j)}$
    $Crossover$:
      For i=1 to rp
        Probabilistically select an hypothesis h_1 from P,
          according to $Pr(h_1)$ given above
        Obtain an hypothesis h_2 from another agent
          chosen at random
        Crossover h_1 with h_2 obtaining h_1'
        Add h_1' to P_s
    $Mutate$: Choose m percent of the members of P_s with

uniform probability. For each, invert one
randomly selected bit in its representation
Learn: Choose lp hypotheses from P_s with uniform
probability and substitute each of them with the
modified hypotheses returned by the procedure *Learn*
Update: $P \leftarrow P_s$
Return the hypothesis from P with the highest fitness

In belief revision, each individual hypothesis is described by the truth value of all the revisables. Since we consider a two-valued revision, each hypothesis gives the truth value true or false to every revisable and therefore it can be considered as a set containing one literal, either positive or default, for every revisable. A chromosome is obtained by associating a bit to each revisable that has value 1 if the revisable is true and 0 if it is false.

Various fitness functions can be used in belief revision. The simplest fitness function is the following

$$Fitness(h_i) = \frac{n_i}{n}$$

where n_i is the number of integrity constraints satisfied by hypothesis h_i and n is the total number of integrity constraints. We will call it an *accuracy* fitness function. Another possible fitness function is the following

$$Fitness(h_i) = \frac{n_i}{n} \times \frac{n}{n+|h_i|} + \frac{f_i}{|h_i|} \times \frac{|h_i|}{n+|h_i|}$$

where f_i is the number of revisables in h_i that are false, and $|h_i|$ is the total number of revisables. We will call it a *hybrid* fitness function. In this way, the fitness function takes into account both the fraction of constraints that are satisfied and the number of revisables whose truth value must be changed to true, preferring hypotheses with a lower number of these, that is minimal revisions are encouraged.

The Lamarckian operator *Learn* changes the values of the revisables in a chromosome C so that a bigger number of constraints is satisfied, thus bringing C closer to a solution. *Learn* differs from a normal belief revision operator because it does not assume that all revisables are false by CWA before the revision but it starts from the truth values that are given by the chromosome C. Therefore, it has to revise some revisables from true to false and others from false to true. As a consequence, the support set does not contain only default literals but also objective literals whose default complement is a revisable.

Learn works in the following way: given a chromosome C, it finds all the support sets for $\perp$ such that they contain literals in C. Therefore, it does not find all support sets for $\perp$ but only those that are subsets of C.

The definition of support set that is used by the Lamarckian operator is therefore different from definition 2.2 and is given as follows:

Definition 3.1 (Lamarckian support set of a literal) *A support set of a literal L of the* WFSXp *model M_P of a program P according to a given set of literals H is denoted by $SS(L,H)$ and is obtained as follows:*

1. *If L is an objective literal in M_P then for each rule $L \leftarrow \mathcal{B}$ in P, such that $\mathcal{B} \subseteq M_P$ there is one $SS(L,H)$ for each $B_i \in \mathcal{B}$. If $\mathcal{B}$ is empty then $SS(L,H) = \{\}$.*

2. *If L is a revisable literal in M_P, then*
   (a) *if L belongs to H, then a support set of L is $\{L\}$.*
   (b) *if the default complement of L belongs to H, then there is no support set for L.*
3. *If L is a default literal not $A \in M_P$:*
   (a) *if L is a revisable then:*
      i. *if L belongs to H, then a support set of L is $\{not\ A\}$.*
      ii. *if A belongs to H, then there is no support set for L.*
   (b) *if rules for A exist in P that have a non-empty body, then choose from each such rule a single literal such that its default complement belongs to M_P. There exists one SS for not A for every SS of each default complement of the chosen literals.*
   (c) *if $\neg A$ belongs to M_P then there exist,* additionally*, support sets SS of not A equal to each $SS(\neg A)$.*

Since the Lamarckian support sets for $\perp$ represent only a subset of all the support sets for $\perp$, a hitting set generated from them is not necessarily a contradiction removal set and therefore it does not represent a solution to the belief revision problem. However, it eliminates some of the derivation paths to $\perp$ and, therefore, may increase the number of satisfied constraints, thus improving the fitness, as required by the notion of Lamarckian operator.

To find support sets we need to know which literals belong to the model of a program. This information is obtainable through some sound and correct procedure for *WFSXp* such as the one described in [ADP95], or the one in [APS99].

In the case of the circuit diagnosis problems in section 4, the support sets procedure becomes simplified in that the occurrences of default negated literals pertain only to revisables.

When computing the support sets, the Lamarckian operator also modifies an extra bit associated to each meme each time the meme is considered in the computation of Lamarckian support sets. This bit indicates whether the meme has been "accessed" by the operator. This is needed for the crossover operator that is described below.

procedure $Learn(C, C')$
  inputs : A chromosome C translated into a set of revisables
  outputs : A revised chromosome C'

  Find the support sets for $\perp$: $Support\_sets([\perp], C, \{\}, \{\}, SS)$
  Find a hitting set HS: $Hitting\_set(SS, HS)$
  Change the value of the literals in the chromosome C
    that appear as well in HS

procedure $Support\_sets(GL, C, S, SSin, SSout)$:
  inputs :
    GL a list of goals

A chromosome H translated into a set of revisables
The current support set S
The current set of support sets $SSin$

outputs :
A set $SSout$ containing the support sets
for the first goal in the list

If GL is empty, then return $SSout = SSin$
Consider the first literal L of the first goal G of GL
($GL = [G|RGL]$ using Prolog notation for lists)
(1) if G is empty then add the current support set to $SSin$
and call recursively the algorithm on the rest of GL
$Support\_sets(RGL, H, \{\}, SSin \cup \{S\}, SSout)$
(2) if G is not empty ($G = [L|RG]$) then:
(2a) if L is a revisable and is in H, then add it to S,
set to 1 L's access bit
and call the algorithm recursively on the rest of G
$Support\_sets([RG|RGL], H, S \cup \{L\}, SSin, SSout)$
(2b) if L is a revisable and it is not in H, or its opposite
is in H, then set to 1 L's access bit, discard S
and call the algorithm recursively on the rest of GL
$Support\_sets(RGL, H, S \cup \{L\}, SSin, SSout)$
(2c) if it is not a revisable then reduce it with all the rules,
obtaining the new goals $G_1, ..., G_n$, one for each
matching rule, add the goals to GL and call
the algorihtm recursively $Support\_sets([[G_1|RG], ...,$
$[G_n|RG]|RGL], H, S, SSin, SSout)$
(2d) if it is not a revisable and there are no rules, then return
without adding S to SS ($SSout = SSin$)

procedure $hitting\_set(SS, HS)$:
Pick a literal from every support set in SS
Add it to HS if it does not lead to contradiction
(i.e. the literal must not be already present
in its complemented form).
If it leads to contradiction pick another literal.

The crossover operator is an extension of a standard uniform crossover operator. The crossover operator produces a new offspring from two parent strings by copying selected bits from each parent. The bit at position i in the offspring is copied from the bit in position i in one of the two parents. The choice of which parent provides the bit for position i is determined by an additional string called crossover mask. This string is a sequence of bits each of which has the following meaning: if bit in position i is 0, then the bit in position i in the offspring is copied from the first parent, otherwise it is copied from the second parent. In uniform crossover, the mask is generated as a bit string where each bit is chosen at random and independently of the others.

In our setting, one of the parents comes from the agent local population, while the other comes from the population of another agent. However, not all the bits in the chromosome are treated equally. In particular, we distinguish genes from memes: genes are modified only by Darwinian operators, while memes are modified by Darwinian and Lamarckian operators. Genes in the offspring can be copied from both parents, while memes can be copied from the parent coming from another agent only if they have been "accessed" by the other agent as a result of the application of the Lamarckian operator.

In this way, an agent can acquire from another agent only memes that have been checked for consistency. Therefore, the flow of memes is asymmetrical and goes from a "teacher" to a "learner", but not vice versa. In particular, in the asymmetrical crossover operator the mask is generated again as a random bit string and crossover is performed in the following way: if the i-th bit in the mask is 1 and the i-th bit in the other agent's chromosome has been accessed, then the i-th bit of the offspring is copied from the other agent's chromosome, otherwise it is copied from the local agent's chromosome. An example of application of the asymmetrical crossover operator is shown in figure 1: as can be seen, bit 1 of the offspring is copied from the local agent (Ag1) chromosome even if the mask bit is set to 1 because its access bit in agent Ag2 is set to 0. Simplified versions of this algorithm have also been

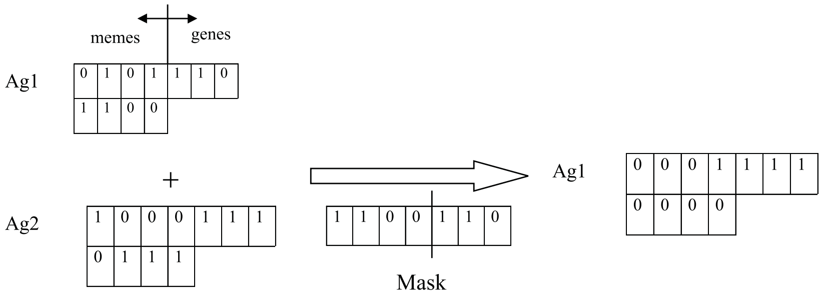

Figure 1: Example of application of the asymmetrical crossover operator.

considered in order to separately test the effectiveness of each of the features added to the standard genetic algorithm. In particular, five algorithms have been considered named in the sequel algorithms 1, 2, 3, 4 and 5. Algorithm 1 is a standard single agent genetic algorithm: crossover is performed only among chromosomes of the same agent and the Lamarckian operator is not used. Algorithm 2 adds to algorithm 1 the use of the Lamarckian operator, with a parameter l (percentage of the population to be mutated Lamarckianly) equal to 0.6. Algorithm 3 is a multi-agent algorithm without the Lamarckian operator, i.e., crossover is performed between chromosomes of different agents but the operator Learn is not applied to them. Algorithm 4 extends algorithm 3 by adding the Lamarckian operator, with a parameter l equal to 0.6. However, it does not distinguish genes from memes, i.e. crossover is always symmetric. Algorithm 5 differs from algorithm 4 because it treats genes and memes differently, exchanging only those memes that have been accessed. For all the algorithms, the mutation rate (parameter m) and the crossover rate (parameter r) have been set to 0.2.

Mark that in algorithms 3, 4 and 5 the agents share the same set of observations and program clauses but have different sets of observations. Each agent scores the chromosomes according to the constraints it has, thus using a local fitness function. At the end of the computation, in order to find a single solution for the revision problem, the best chromosome in each agent is considered and is scored with a fitness function that considers all the constraints (global fitness function). Then the chromosome with the highest global fitness is returned as the solution. In this way the multi-agent system finds a solution for the global belief revision problem.

These algorithms have been used in order to experimentally prove the following theses:

1. Lamarckism plus Darwinism outperforms Darwinism alone in the single agent case;
2. the distributed algorithm (with or without the Lamarckian operator) has a performance that is comparable (and, in particular, not significantly inferior) to that of the non-distributed one, in the same number of generations and the same overall number of individuals;
3. Lamarckism plus Darwinism outperforms Darwinism alone in the multi-agent case;
4. the distributed algorithm with genes and memes distinguished performs better than the one without the distinction.

4. Experiments

The algorithms have been tested on a number of belief revision problems in order to prove the above theses. In particular, we have considered problems of digital circuit diagnosis, as per [DPS97], and the n-queen problem.

4.1 Digital Circuit Diagnosis

In problems of digital circuit diagnosis there is a difference between the observed and the predicted outputs. Figure 4.1 shows a sample circuit together with the observed inputs and outputs of the circuit and the predicted outputs of each gate. The aim of the diagnosis is to find which components are faulty.

A problem of digital circuit diagnosis can be modelled as a belief revision problem by describing it with a logic program consisting of four groups of clauses: one that allows to compute the predicted output of each component, one that describes the topology of the circuit, one that describes the observed inputs and outputs, and one that consists of integrity constraints stating that the predicted value for an output of the system cannot be different from the observed value. The representation formalism we use is the one of [DPS97]. As regards the clauses for computing the predicted output of a gate, let us consider the case of a two-input NAND:

```
val( out(nand2,Name), V ) :-
  not ab(Name),
  val( in(nand2,Name,1), W1),
  val( in(nand2,Name,2), W2),
  nand2_table(W1,W2,V).
```

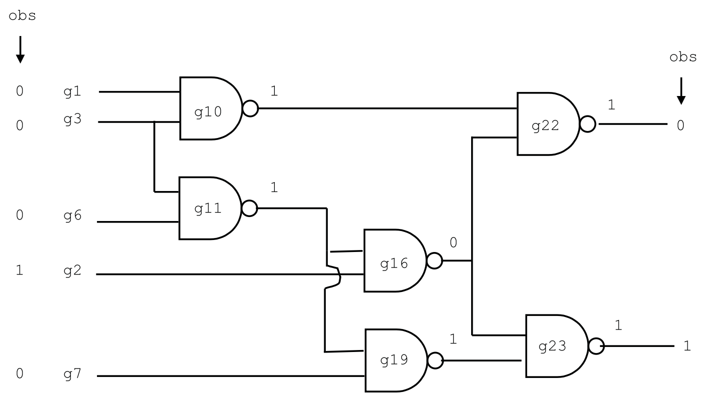

Figure 2: `c17` circuit from ISCAS85's set of benchmark circuits.

```
val( out(nand2,Name), V ) :-
  ab(Name),
  val( in(nand2,Name,1), W1),
  val( in(nand2,Name,2), W2),
  and2_table(W1,W2,V).
```

In these clauses `Name` is the name of the component and the definition of `nand2_table`, `and2_table` consist of facts describing the input/output relation of, respectively, a two-input NAND gate and a two-input AND gate. `ab(Name)` is a revisable that can be assumed true or false. If it is assumed true, it expresses a faulty behaviour of a component of the circuit, described in this case by the second clause above. If it is assumed false, it expresses a correct behaviour of a component of the circuit, being described by the first clause above.

The topology of the circuit is described by a set of facts for the predicate `conn/2`. For example, consider the fact `conn(in(nand2, g10, 1), out(inpt0, g1))` describing a part of the circuit shown in figure 4.1. This fact states that the input 1 of gate `g10` of type `nand2` is connected to the output of gate `g1` of type `inpt0`. The gates of type `inpt0` are the input pins of the circuit.

The clauses that describe the observed values for the input and for the output of the circuit are facts for the `obs/2` predicate. `obs(out(inpt0, g1), 0)` states that the input `g1` has value 0.

As regards the integrity constraints, we have two constraints for each output of the circuit, one stating that the output can not be 0 if it was observed to be 1 and the other stating that the output can not be 1 if it was observed to be 0. For example, the constraint `ic([obs(out(nand2, g22), 0), val(out(nand2, g22), 1)]).` states that the value of the output of `g22` cannot be 1 if it was observed to be 0.

In case the circuit is faulty, one or more of the constraints will be violated. By means of belief revision, the values of the revisables are changed in order to restore consistency. The literals of the form `ab(Name)` that are assigned the value true identify the faulty components.

Usually, the number of faulty components is very small, very often one or two. This means that only one or two revisables of the form `ab(Name)` will be true, while all the other will be false. Therefore, in order to speed up the search for a solution, we have modified the genetic algorithm so that a chromosome with all the genes set to 0 (all revisables false) is added to the initial population. Moreover, the hybrid fitness function is used, in order to take into account not only the number of satisfied constraints but also the number of false literals in the solution.

In order to show the difference between a belief revision operator and the Lamarckian operator, let us show their behaviour on the `c17` circuit supposing the following hypothesis is given:

```
C={ab(g10), not ab(g11), ab(g16), not ab(g19), not ab(g22),
not ab(g23)}
```

In this case, two out of four constraints are violated because the predicted output of `g22` is 1 and of `g23` is 0.

A belief revision operator, as for example REVISE [DPS97], finds a solution where the only abnormality literal that is true is `ab(g22)` while all the others are false. This solution is found independently of the initial starting hypothesis because the support sets for $\perp$ are found independently of the initial values of the revisables. This new hypothesis eliminates both constraint violations.

The Lamarckian operator, instead, will modify `C` into the following hypothesis:

```
C'={ab(g10), ab(g11), ab(g16), not ab(g19), not ab(g22),
not ab(g23)}
```

that differs from `C` only in the values of `ab(g11)`. This new hypothesis eliminates only one constraint violation because the output of `g22` is still different from the observed value.

The system has been tested on some real world problems taken from the ISCAS85 benchmark circuits [BPH85] that has been used as well for testing the belief revision system REVISE [DPS97].[2]

We have considered the `voter` circuit that has 59 gates and 4 outputs, corresponding respectively to 59 revisables and 8 constraints.

Algorithms 1, 2, 3, 4 and 5 have been tested on the `voter` circuit. Each algorithm was run 10 times. The parameters that have been used for the runs are: 10 maximum generations, 40 individuals for algorithms 1 and 2 (single agent), 10 individuals per agent and 4 agents for algorithms 3, 4 and 5. In algorithms 3, 4 and 5 each agent has the same set of observations and program clauses, while the integrity constraints are distributed among the agents so that each agent knows only the constraints that are related to one same output. The hybrid fitness function was adopted.

In table 1 we show, for each algorithm, the value of the fitness function and of its standard deviation for the best hypothesis after ten generations averaged over the 10 runs.

As can be seen, theses 1, 2 and 4 are confirmed. The fitness increment between algorithms 1 and 2, even if small, shows the usefulness of the Lamarckian operator in the single-agent case (thesis 1). The fitness increment between algorithms 1 and 3 shows that in this case the distribution of constraints has helped in finding a good solution and thus proves thesis 2 that required the performance of the multi agent case to be comparable

2. These examples can be found at `http://www.soi.city.ac.uk/~msch/revise/`.

Algorithm	Fitness	Standard Deviation
1	0.9537	0.035494
2	0.9582	0.015414
3	0.9776	0.007866
4	0.9776	0.007866
5	0.9805	0.007209

Table 1: Voter experiments with algorithms 1, 2, 3, 4 and 5

(and, in particular, not significantly inferior) to that of the single agent case. The fitness increment between algorithms 4 and 5 shows the usefulness of the distinction of memes from genes and of the asymmetrical crossover mechanism among memes (thesis 4). The low values for the standard deviation in all cases show that the result does not heavily depend on the initial population. Thesis 3, instead, that states that there should be an increment between algorithms 3 and 4, is not confirmed. In this case, in fact, the use of memes in conjunction with Lamarck is fundamental in order to get an improvement by means of Lamarckian evolution.

4.2 n-queen Problem

The n-queen problem consists in positioning n queens on a $n \times n$ checkboard so that no queen attacks each other. This problem can be seen as Constraint Satisfaction Problem (CSP) where the constraints are: the total number of queens must be n; for each row, the total number of queens must not be bigger than one; for each column, the total number of queens must not be bigger than one and for each diagonal, the total number of queens must not be bigger than one. This problem can be seen as a belief revision problem by assigning a revisable of the form `queen(Row,Column)` to each position `(Row,Column)` in the checkboard. Then, each constraint of the CSP can be written as an integrity constraint.

Algorithms 1, 2, 3, 4 and 5 have been tested also on the n-queen problem with the same parameter as for the `voter` experiment: each algorithm was run 10 times, each run had 10 maximum generations, 40 individuals for algorithms 1 and 2 (single agent), 10 individuals per agent and 4 agents for algorithms 3, 4 and 5. The accuracy fitness function was adopted.

We have considered a problem with $n = 8$. In this case there is a total of 43 constraints: 1 constraint for the total number of queens, 8 constraints for the rows, 8 for the columns and 26 for the diagonals. For multi-agent experiments each agent has the same set of observations and program clauses, while the constraints were divided amongst them: 2 constraints on the rows and 2 on the columns have been assigned to each agent, while the constraints on diagonals have been divided in groups of 6, 6, 7 and 7 and correspondingly assigned to the agents. The constraint on the total number of queens has been assigned to one of the agents with only 6 constraints on the diagonals. Therefore, three agents have 9 constraints and one agent has only 8.

In table 2 we show, for each algorithm, the value of the fitness function for the best hypothesis averaged over the ten runs.

Algorithm	Fitness	Standard Deviation
1	0.7581	0.06686
2	0.8232	0.01200
3	0.7930	0.02992
4	0.7953	0.04753
5	0.8069	0.03110

Table 2: n-queen experiments with algorithms 1, 2, 3, 4 and 5

As can be seen, in this case theses 1, 2, 3 and 4 are confirmed. In fact, there is an increment between algorithms 1 and 2 (thesis 1), the performance in the multiagent case is slightly inferior to that of the single agent case (thesis 2), the performance increases from algorithm 3 to 4 (thesis 3) and also from algorithm 4 to 5 (thesis 4). Again, the low values for the standard deviation in all cases show that the result does not heavily depend on the initial population.

5. Related Work

Various authors have investigated the integration of Darwinian and Lamarckian evolution into a genetic algorithm [HB96, AL94, LTG96, Gre91]. A Lamarckian operator first translates a genotype into its corresponding phenotype and performs a local search in the phenotype's space. The local optimum that is obtained is then translated back into its corresponding genotype and added to the population for further evolution. [HB96] has shown that the traditional genetic algorithm performs well for searching widely separated portions of the search space caused by a scattered population, while Lamarckism is more proficient for exploring localized areas of the population that would otherwise be missed by the global search of the genetic algorithm. Therefore, Lamarckism can play an important rôle when the population has converged to areas of local maxima that would not be thoroughly explored by the standard genetic algorithm. The adoption of a Lamarckian operator provides a significant speedup in the performance of the genetic algorithm.

Similarly to the approaches in [HB96, AL94, LTG96, Gre91], we adopt a procedure for Lamarckian evolution that first translates the chromosome into its phenotype and then modifies it in order to improve its fitness. In our case too the Lamarckian operator improves the performance of the genetic algorithm. Differently from [HB96, AL94, LTG96, Gre91], the procedure does not perform a local search but finds an improvement by tracing logical derivations causally supporting the undesired behaviour.

Cultural evolution is a form of evolution that is related to Lamarckian evolution [SL96b, SL96a]. In cultural evolution, the population of individual shares a common memory where each individual can read or store beliefs. In particular, the authors assume that each chromosome represents a program, i.e., a series of actions. When the chromosome must be scored the program is executed and the effect on the world is evaluated. The authors assume that, among the actions that these programs can perform, there are also the actions `READ` and `WRITE` that allow the programs to read from and write to a shared persistent memory. In this way, experiences can be communicated among different individuals, thus realizing a mechanism similar to the transmission of culture in human societies. Our approach for the transmission of knowledge differs from this one because no global shared memory is kept, rather, the knowledge acquired by each individual during its lifetime is transmitted to its offspring by reproduction.

In [MJO94] the authors propose an approach for performing belief revision in a multi agent context. In their approach, each agent exploits an Assumptions Based Truth Maintenance (ATMS) system in order to perform the revision of beliefs. As in our approach, each agent has a different repository for knowledge and its beliefs may not be consistent with those of other agents, consistency is enforced only locally inside each agent. Differently from us, in [MJO94] the authors consider an exchange of beliefs by means of a number of communications primitives. Communication happens in three cases. The first is when an agent can not establish by itself the truth value of an assumption or a goal: in this case, it asks it to its acquaintances. The second case is when an agent finds a conclusion or an assumption that it knows being of interest for another agent: in this case it communicates the results. The third case is when an agent has revised the truth value of a belief that it had previously communicated to other agents: in this case the agent communicates the other agents the new truth value for the belief. Therefore, in [MJO94] the cooperation among agents is explicit, while in our work the cooperation emerges as the result of the continuous exchange of chromosomes among agents.

In [MJO94] the system is able to answer uniquely to queries posed to the system by means of a meta-level algorithm that works in the following way: a fact is false if it is considered false by at least one agent, a fact is true if no agent considers it false and there is at least one agent that considers it true. In our system, instead, the global result of the belief revision process is given by the chromosome with the best global accuracy, also computed by means of a meta-level algorithm that considers all the constraints. Mark that in our case we consider only approximate belief revision, i.e., we do not ensure the consistency of all the constraints.

6. Conclusions and Future Work

We have presented a genetic algorithm for performing belief revision in a multi-agent environment. In this setting, different individuals are exposed to different experiences. This may happen because the world surrounding an agent changes over time or because we allow agents exploring different parts of the world. The algorithm permits the exchange of chromosomes from different agents and combines two different evolution strategies, one based on Darwin's and the other on Lamarck's evolutionary theory. The algorithm therefore includes also a Lamarckian operator that changes the memes of an agent in order to improve its fitness. The operator is implemented by means of a belief revision procedure that, by tracing logical derivations, identifies the memes leading to contradiction. Moreover, the algorithm comprises a special crossover mechanism for memes in which a meme can be acquired from another agent only if the other agent has "accessed" the meme, i.e. if an application of the Lamarckian operator has read or modified the meme.

Experiments on a problem of digital circuit diagnosis and on the n-queen problem show that each feature added to the standard genetic algorithm can provide an improvement of the resulting fitness.

Lamarckian and Darwinian operators have complementary functions: Lamarckian operators are used to get closer to a solution of a given belief revision problem, while Darwinian operators are used in order to distribute the acquired knowledge amongst agents. We could consider as well Lamarckian operators that not only bring a chromosome closer to a solution but actually turn it into a solution. In this case, when a new constraint is presented to an agent, it first applies a Lamarckian operator to find a chromosome satisfying the new constraint and then it applies a Darwinian operator to distribute the "knowledge" so acquired to other chromosomes in the same or other agents. In this way chromosomes may be prepared in advance for meeting new constraints. Moreover, by means of the Lamarckian operator, low values for the percentage of individuals to be mutated and crossed-over could be used, thus saving computation time.

The exchange of genetic material is useful also in the case in which the chromosomes do not have all the relevant revisables to start with (three-valued revision). When they acquire new revisables from other chromosomes they are obtaining specialized knowledge from others. This is for example the case of the diagnosis of a car fault performed by different experts: the expert mechanic, the expert electrician, the expert car designer, etc. Each of them makes a diagnosis about the part of the car that concerns their speciality. Next they all have to come to a joint diagnosis by exchanging information about each others' revisables.

We conjecture that in this new problem setting, where there is dynamicity in the data, the integration of the Lamarckian and Darwinian operators will fully exhibit, and be extolled in, its potential.

7. Acknowledgements

L. M. Pereira acknowledges the support of PRAXIS project MENTAL "An Architecture for Mental Agents". E. Lamma and F. Riguzzi acknowledge the support of the MURST project "Intelligent Agents: Interaction and Knowledge Acquisition".

References

[ADP94] J. J. Alferes, C. V. Damásio, and L. M. Pereira. SLX - A top-down derivation procedure for programs with explicit negation. In M. Bruynooghe, editor, *Proc. Int. Symp. on Logic Programming.* The MIT Press, 1994.

[ADP95] J. J. Alferes, C. V. Damásio, and L. M. Pereira. A logic programming system for non-monotonic reasoning. *Journal of Automated Reasoning*, 14:93–147, 1995.

[AL94] D. H. Ackely and M. L. Littman. A case for lamarckian evolution. In C. G. Langton, editor, *Artificial Life III.* Addison Wesley, 1994.

[AP96] J. J. Alferes and L. M. Pereira. *Reasoning with Logic Programming*, volume 1111 of *LNAI.* Springer-Verlag, 1996.

[APP98] J. J. Alferes, L. M. Pereira, and T. C. Przymusinski. "Classical" negation in non-monotonic reasoning and logic programming. *Journal of Automated Reasoning*, 20:107–142, 1998.

[APS99] J. J. Alferes, L. M. Pereira, and T. Swift. Well-founded abduction via tabled dual programs. In D. De Schreye, editor, *Procs. of the 16th International Conference on Logic Programming*, pages 426–440, Las Cruces, New Mexico, 1999. MIT Press.

[Bla99] Susan Blackmore. *The Meme Machine.* Oxford U.P., Oxford, UK, 1999.

[BPH85] F. Brglez, P. Pownall, and R. Hum. Accelerated ATPG and fault grading via testability analysis. In *Proceedings of IEEE Int. Symposium on Circuits and Systems*, pages 695–698, 1985. The ISCAS85 benchmark netlist are available via ftp `mcnc.mcnc.org`.

[DP97] C. V. Damásio and L. M. Pereira. Abduction on 3-valued extended logic programs. In V. W. Marek, A. Nerode, and M. Trusczynski, editors, *Logic Programming and Non-Monotonic Reasoning - Proc. of 3rd International Conference LPNMR'95*, volume 925 of *LNAI*, pages 29–42, Germany, 1997. Springer-Verlag.

[DP98] C. V. Damásio and L. M. Pereira. A survey on paraconsistent semantics for extended logic programs. In D.M. Gabbay and Ph. Smets, editors, *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, volume 2, pages 241–320. Kluwer Academic Publishers, 1998.

[DPP97] J. Dix, L. M. Pereira, and T. Przymusinski. Prolegomena to logic programming and non-monotonic reasoning. In J. Dix, L. M. Pereira, and T. Przymusinski, editors, *Non-Monotonic Extensions of Logic Programming - Selected papers from NMELP'96*, number 1216 in LNAI, pages 1–36, Germany, 1997. Springer-Verlag.

[DPS97] C. V. Damásio, L. M. Pereira, and M. Schroeder. REVISE: Logic programming and diagnosis. In *Proceedings of Logic-Programming and Non-Monotonic Reasoning, LPNMR'97*, volume 1265 of *LNAI*, Germany, 1997. Springer-Verlag.

[GL88] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. A. Bowen, editors, *Proceedings of the 5th Int. Conf. on Logic Programming*, pages 1070–1080. MIT Press, 1988.

[Gre91] J. J. Grefenstette. Lamarckian learning in multi-agent environments. In *Proc. 4th Intl. Conference on Genetic Algorithms.* Morgan Kauffman, 1991.

[HB96] W. E. Hart and R. K. Belew. Optimization with genetic algorithms hybrids that use local search. In R. K. Belew and M. Mitchell, editors, *Adaptive Individuals in Evolving Populations.* Addison Wesley, 1996.

[LTG96] Y. Li, K. C. Tan, and M. Gong. Model reduction in control systems by means of global structure evolution and local parameter learning. In D. Dasgupta and Z. Michalewicz, editors, *Evolutionary Algorithms in Engineering Applications.* Springer Verlag, 1996.

[Mit97] T. M. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[MJO94] B. Malheiro, N. R. Jennings, and E. Oliveira. Belief revision in multiagent systems. In *Proceedings of the 11th European Conference on Artificial Intelligence*, 1994.

[PA92] L. M. Pereira and J. J. Alferes. Well founded semantics for logic programs with explicit negation. In *Proceedings of the European Conference on Artificial Intelligenece ECAI92*, pages 102–106. John Wiley and Sons, 1992.

[PDA93] L. M. Pereira, C. V. Damásio, and J. J. Alferes. Diagnosis and debugging as contradiction removal. In L. M. Pereira and A. Nerode, editors, *Proceedings of the 2nd International Workshop on Logic Programming and Non-monotonic Reasoning*, pages 316–330. MIT Press, 1993.

[SL96a] Lee Spector and Sean Luke. Cultural transmission of information in genetic programming. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 209–214. The MIT Press, 1996.

[SL96b] Lee Spector and Sean Luke. Culture enhances the evolvability of cognition. In *Proceedings of the 1996 Cognitive Science Society Meeting*, 1996.

[VGRS91] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

The definition of *WFSX* that follows is taken from [ADP94] and is based on the alternating fix points of Gelfond-Lifschitz Γ-like operators.

Definition .1 (The Γ-operator) *Let P be an extended logic program and let I be an interpretation of P. $\Gamma_P(I)$ is the program obtained from P by performing in the sequence the following four operations:*

- *Remove from P all rules containing a default literal $L = not\ A$ such that $A \in I$.*
- *Remove from P all rules containing in the body an objective literal L such that $\neg L \in I$.*
- *Remove from all remaining rules of P their default literals $L = not\ A$ such that $not\ A \in I$.*
- *Replace all the remaining default literals by proposition* $\mathbf{u}$.

In order to impose the coherence requirement, we need the following definition.

Definition .2 (Seminormal Version of a Program) *The seminormal version of a program P is the program P_s obtained from P by adding to the (possibly empty) Body of each rule $L \leftarrow Body$ the default literal $not \neg L$, where $\neg L$ is the complement of L with respect to explicit negation.*

In the following, we will use the following abbreviations: $\Gamma(S)$ for $\Gamma_P(S)$ and $\Gamma_s(S)$ for $\Gamma_{P_s}(S)$.

Definition .3 (Partial Stable Model) *An interpretation $T \cup not\ F$ is called a* partial stable model *of P iff $T = \Gamma\Gamma_s T$ and $F = H^E(P) - \Gamma_s T$.*

Partial stable models are an extension of *stable models* [GL88] for extended logic programs and a three-valued semantics. Not all programs have a partial stable model (e.g., $P = \{a, \neg a\}$) and programs without a partial stable model are called *contradictory*.

Theorem .1 (*WFSX* Semantics) *Every non-contradictory program P has a least (with respect to $\subseteq$) partial stable model, the well-founded model of P denoted by $WFM(P)$. To obtain an iterative "bottom-up" definition for $WFM(P)$ we define the following transfinite sequence $\{I_\alpha\}$:*

$$I_0 = \{\}; \quad I_{\alpha+1} = \Gamma\Gamma_S I_\alpha; \quad I_\delta = \bigcup\{I_\alpha | \alpha < \delta\}$$

where δ is a limit ordinal. There exists a smallest ordinal λ for the sequence above, such that I_λ is the smallest fix point of $\Gamma\Gamma_S$. Then, $WFM(P) = I_\lambda \cup not\ (H^E(P) - \Gamma_S I_\lambda)$.