Belief Revision via Lamarckian Evolution

Evelina Lamma, Fabrizio Riguzzi
Department of Engineering, University of Ferrara,
Via Saragat 1, 44100 Ferrara, Italy

Luís Moniz Pereira
Centro de Inteligência Artificial - CENTRIA,
Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa,
2829-516 Caparica, Portugal

Abstract

We present 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. Experiments on a problem of digital circuit diagnosis and on the n-queen problem show that the addition of the Lamarckian operator in the single agent case improves the fitness of the best solution, even if in the digital circuit case the fitness difference is not significant. Moreover, the experiments show that the distribution of constraints, even if it leads to a decrease of the fitness of the best solution, does not produce a significant difference. *Keywords:* Evolutionary Systems, Belief Revision, Multi-agent Systems.

1 Introduction

Herein, we propose a genetic algorithm for belief revision that includes, besides Darwin's operators of selection, mutation and crossover [14], a logic based Lamarckian operator as well. This operator differs from Darwinian ones precisely because it modifies a chromosome 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 un-encountered 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. As multiple-population GAs (see [5], for discussion), we allow knowledge and experience to be coded as genes in an agent and consider several sub-populations which exchange individuals occasionally. In particular, 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 collaborative problem-solving, we apply them to distributed model-based diagnosis of digital circuits, a natural domain in which belief revision techniques apply [6], and to the n-queen constraint satisfaction problem.

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 Preliminaries

We consider belief revision of first order theories expressed in the language of extended logic programs [3]. For this language, we adopt the Extended Well Founded Semantics (*WFSX*) that extends the well founded semantics (*WFS*) [18] for normal logic programs to programs extended with explicit negation, besides the implicit or default negation of normal 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 implicitly assumed. The *revisables* of a program P are the elements of a chosen subset $Rev(P)$, of the set of all literals L having no rules for them in P. The set $Rev(P)$ contains also, for each objective literal L in $Rev(P)$, its default complement *not* L.

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 [14]) 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 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
    at each step
  $l$: the fraction of the population that should
    evolve Lamarckianly at each step

  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 $h'$
        Add $h'$ 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.

This is done by modifying the belief revision techniques presented in [15]. In particular, in [15] an algorithm for belief revision is presented that is based on the notions of *support sets, hitting sets* and *removal sets.* Intuitively, a support set of a literal is the set of revisables supporting the derivation of the literal. The 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. A removal set of a literal is a hitting set of all the support sets of the literal. Contradiction in [15] is removed by finding the removal set of $\perp$.

The Lamarckian operator *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. These support sets are called *Lamarckian support sets* (a formal definition for them is given in [12]). Therefore, it does not find all support sets for $\perp$ but only those that are subsets of C.

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.

In the case of the experiments we consider in section 4, the support sets procedure becomes simplified in that the occurrences of default negated literals pertain only to revisables. This simplification results in the procedure that is reported 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 each 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 $GL$
        $Support\_sets(RGL, H, \{\}, SSin, SSout)$
      (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 algorithm
        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 call the algorithm on
        the rest of $GL$
        $Support\_sets(RGL, H, \{\}, SSin, SSout)$

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.
```

When computing the support sets, the Lamarckian operator also modifies an extra bit associated with 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.

The crossover operator is an extension of a standard uniform crossover operator. The crossover operator produces a new offspring from two parent chromosomes 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.

Simplified versions of this algorithm have also been considered in order to test the effectiveness of the features added to the standard genetic algorithm. In particular, four algorithms have been considered named in the sequel algorithms 1, 2, 3 and 4. 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 and the special treatment of memes in crossover. 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, and the special treatment of memes in crossover. 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 and 4 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;

As regards thesis 2, we require only that the difference is not significant because we can not expect an improvement from distributing constraints with respect to the centralized case where all the information is available. Therefore our aim is to prove that the decrease in fitness is not significant.

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 [6], and the n-queen problem.

In order to evaluate if the accuracy differences between algorithms are significant or not, we have computed a 10-fold cross-validated paired t test for every pair of algorithms (see [**?**] for an overview of statistical tests for the comparison of machine learning algorithms). This test is computed as follows. Given two algorithms A and B, let $p_A(i)$ (respectively $p_B(i)$) be the maximum fitness achieved by algorithm A (respectively B) in trial i. If we assume that the 10 differences $p(i) = p_A(i) - p_B(i)$ are drawn independently from a normal distribution, then we can apply the Student t-test by computing the statistic

$$t = \frac{\bar{p}\sqrt{n}}{\sqrt{\frac{\sum_{i=1}^{n}(p^{(i)}-\bar{p})^2}{n-1}}}$$

where n is the number of folds (10) and $\bar{p}$ is

$$\bar{p} = \frac{1}{n}\sum_{i=1}^{n} p^{(i)}$$

In the null hypothesis, i.e. that A and B obtain the same fitness, this statistic has a t distribution with $n-1$ (9) degrees of freedom. If we consider a probability of 90%, then the null hypothesis can be rejected if

$$|t| > t_{9,0.90} = 1.383$$

A problem of digital circuit diagnosis they 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 revisables are literals of the form `ab(Name)` which, if true, state that the gate `Name` is faulty. The representation formalism we use is the one of [6].

If the digital 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.

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.

The system has been tested on some real world problems taken from the ISCAS85 benchmark circuits [4] that has been used as well for testing the belief revision system REVISE [6].[1]

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 and 4 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 and 4. In algorithms 3 and 4 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.

Table 1 shows, for each algorithm, the value of the fitness function for the best hypothesis averaged over the 10 runs together with its standard deviation, while

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

Algorithm	Fitness	Standard Deviation
1	0.9537	0.0349
2	0.9582	0.0154
3	0.9776	0.0079
4	0.9806	0.0072

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

Comparison	$|t|$ value
1-2	0.394
1-3	**1.751**
2-4	**3.737**

Table 2: Result of the t-test for different couples of algorithms on the voter dataset.

table 2 shows the value of the t statistics for the various couples of algorithms.

As can be seen, algorithm 2 has a higher fitness than algorithm 1 but the difference is not significant, therefore the usefulness of the Lamarckian operator (thesis 1) is only partly verified. 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 (and, in particular, not significantly inferior) to that of the single agent case. Similarly, the fitness increment between algorithms 2 and 4 also proves thesis 2 when the Lamarckian operator is used. In both cases a fitness significantly higher than the single agent case has been obtained, thus showing that in this case the distribution of constraints has helped. The low values for the standard deviation in all cases show that the results do not heavily depend on the initial population.

The n-queen problem consists in positioning n queens over a $n \times n$ checkerboard so that no queen attacks each other. This problem can be seen as a 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 checkerboard. Then, each constraint of the CSP can be written as an integrity constraint.

Algorithms 1, 2, 3 and 4 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

Algorithm	Fitness	Standard Deviation
1	0.7581	0.0669
2	0.8233	0.0120
3	0.7930	0.0299
4	0.8070	0.0311

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

Comparison	$\|t\|$ value
1-2	**1.497**
1-3	1.255
2-4	0.937

Table 4: Result of the t-test for different couples of algorithms on the n-queen dataset.

per agent and 4 agents for algorithms 3 and 4. 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.

Table 3 shows, for each algorithm, the value of the fitness function for the best hypothesis averaged over the 10 runs while table 4 shows the value of the t statistics for the various couples of algorithms.

As can be seen, in this case theses 1 and 2 are confirmed. In fact, there is a significant increment between algorithms 1 and 2 (thesis 1) and the performance in the multi-agent case is inferior to that of the single agent case but not significantly (thesis 2). Again, the low values for the standard deviation in all cases show that the results do 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 [11, 1, 13, 9]. 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. [11] 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 [11, 1, 13, 9], 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 [11, 1, 13, 9], the procedure does not perform a local search but finds an improvement by tracing the logical derivations causally supporting the undesired behaviour.

Our work is also related to co-evolutive approaches and distributed GAs (see [16, 5]. It can be considered a cooperative co-evolutionary approach (see [16]) to belief revision since knowledge about the domain problem (and constraints in particular) are spread among the agents, each of which is ruled by a GA. In this respect, each species represents a possibly partial solution to the belief revision problem. While in [16] the complete solutions (to the problem of function optimization, in that paper) are obtained by assembling the representative members of each of the species present, in our work the solution is obtained by evolution and exchange between species, and by the application of the crossover operator to members of two species, the foreigner of which may have already gained in experience (i.e., it evolved Lamarckianly).

Relating our work to the field of belief revision, it must be noted that the kind of belief revision we consider, namely the removal of inconsistency from an inconsistent knowledge base, differs from the definition of belief revision given by Alchourrón, Gärdenfors and Makinson in [2], i.e. the incorporation of new information into a knowledge base such that the new knowledge base is consistent. The belief revision problem we consider has been called consolidation by Hansson [10].

Other authors in the belief revision field have considered the problem of updating the knowledge in a multi-agent system. [8] considers the case of an agent that has to update its knowledge base with information coming from different sources: in this case the agent must be able to assign a different credibility to the different sources. We consider as well the problem of knowledge updating from multiple sources (the other agents) but we assign the same credibility to the other sources. [7] considers a setting where an agent maintains a set of beliefs regarding another agent and updates the beliefs from the observation of the other agent's behaviour. In [7] the authors provide a belief revision methodology that is able to cope particularly well with this interesting special case. In our work, instead, we provide a general belief revision approach, without considering any special cases.

6 Conclusions and Future Work

We have proposed a novel way of looking at belief revision, which is GA based, and hence a new application domain for GAs. Since it is still in the initial development stages it cannot be expected yet to compete with hard-boiled methods for belief revision. On the other hand, we believe our method to be important for situations where classical belief revision methods hardly apply: Those where environments are non-uniform and time changing. These can be explored by distributed agents that evolve genetically to accomplish cooperative belief revision, using our approach. Notwithstanding, some type of efficient hybrid implementation approach might emerge, combining hard-boiled belief revision techniques with the newly introduced GA supplement. Our contribution has been to get the new approach off the ground.

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 EC project IST-2001-32530 SOCS .

References

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

[2] Carlos Alchourrón, Peter Gärdenfors, and David Makinson. On the logic of theory change. *Journal of Symbolic Logic*, 50(2):510–530, 1985.

[3] 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.

[4] 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`.

[5] Erick Cant-Paz. A survey of parallel genetic algorithms.

[6] 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.

[7] Fiorella de Rosis, Rino Falcone, Emanuele Covino, and Cristano Castelfranchi. Bayesian cognitive diagnosis in believable multiagent systems, 1999.

[8] A. Dragoni and P. Giorgini. Revising beliefs received from multiple source, 1999.

[9] J. J. Grefenstette. Lamarckian learning in multiagent environment. In *Proc. 4th Intl. Conference on Genetic Algorithms*. Morgan Kauffman, 1991.

[10] S. Hansson. Belief base dynamics, 1991.

[11] 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.

[12] E. Lamma, L. M. Pereira, and F. Riguzzi. Multiagent logic aided lamarckian learning. Technical Report DEIS-LIA-00-004, Dipartimento di Elettronica, Informatica e Sistemistica, University of Bologna (Italy), 2000. LIA Series no. 44.

[13] 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.

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

[15] 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.

[16] M. Potter and K. de Jong. A cooperative coevolutionary approach to function optimization, 1994.

[17] Mitchell A. Potter, Kenneth A. De Jong, and John J. Grefenstette. A coevolutionary approach to learning sequential decision rules. In Larry Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 366–372, San Francisco, CA, 1995. Morgan Kaufmann.

[18] 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.