Belief Revision by Lamarckian Evolution

Evelina Lamma[1], Luís Moniz Pereira[2], and Fabrizio Riguzzi[1]

[1] Dipartimento di Ingegneria, Università di Ferrara,
Via Saragat 1, 44100 Ferrara, Italy
elamma@ing.unife.it, friguzzi@ing.unife.it
[2] Centro de Inteligência Artificial (CENTRIA), Departamento de Informática,
Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2825-114
Caparica, Portugal
lmp@di.fct.unl.pt

Abstract. We propose a multi-agent 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.

1 Introduction

Herein, we propose a genetic algorithm for belief revision that includes, besides Darwin's operators of selection, mutation and crossover [10], 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. As multiple-population GAs (see [4], 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 [5].

2 Preliminaries

We consider belief revision of first order theories expressed in the language of extended logic programs [2]. For this language, we adopt the Extended Well Founded Semantics (*WFSX*) that extends the well founded semantics (*WFS*) [14] 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 $\bot \leftarrow L, \neg L$ that are implicitely 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 , and code revisable beliefs.

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 [10]) 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.

In two-valued belief revision, each individual hypothesis is described by the truth value of all the revisables. Therefore each hypothesis 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.

The fitness function that has been used takes the following form:

$$Fitness(h_i) = \frac{n_i}{n} + \frac{f_i}{|h_i|} \times 0.5$$

where n_i is the number of integrity constraints satisfied by hypothesis h_i, n is the total number of integrity constraints, f_i is the number of revisables in h_i that are false, and $|h_i|$ is the total number of revisables. 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. Assuming that the initial value of the revisables is false, this means that minimal revisions are encouraged. The factor 0.5 was chosen in order to give more importance to the accuracy, rather than to the number of unchanged revisables.

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 [11]. In particular, in [11] 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 revisable 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 [11] is removed by finding the removal set of $\perp$.

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

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 [8]). 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 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 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.

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

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 $\mathit{Support\_sets}(GL, C, S, SSin, SSout)$:
inputs : GL: list of goals, C: a chromosome translated into a set of revisables, S: the current support set, $SSin$: the current set of support sets

outputs : $SSout$: a set 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
$\mathit{Support\_sets}(RGL, C, \{\}, SSin \cup \{S\}, SSout)$
(2) if G is not empty ($G = [L|RG]$) then:
(2a) if L is a revisable and is in C, then add it to S,
set to 1 L's access bit
and call the algorithm recursively on the rest of G
$\mathit{Support\_sets}([RG|RGL], C, S \cup \{L\}, SSin, SSout)$
(2b) if L is a revisable and it is not in C, or its opposite
is in C, then set to 1 L's access bit, discard S
and call the algorithm recursively on the rest of GL
$\mathit{Support\_sets}(RGL, C, S \cup \{L\}, SSin, SSout)$
(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 $\mathit{Support\_sets}([[G_1|RG], ...,$
$[G_n|RG]|RGL], C, S, SSin, SSout)$
(2d) if it is not a revisable and there are no rules, then return
without adding S to SS ($SSout = SSin$)

procedure $\mathit{hitting\_set}(SS, HS)$:
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 at position i in one of the two parents. The choice of which parent provides the bit for position i is determined by the crossover mask that, in uniform crossover, 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 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.

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) performs better than the non-distributed one, in the same number of generations, because of parallel exploration;
3. Lamarckism plus Darwinism outperforms Darwinism alone in the multi-agent case;
4. the distributed algorithm with the distinction between genes and memes performs better than the one without the distinction.

Circuit	Algorithm	Fitness	Standard Deviation	Correct solution
voter	1	1.295	0.00634	100 %
	2	1.312	0.01728	100 %
alu4_flat	1	1.193	0.03939	20 %
	2	1.213	0.01765	33 %

Table 1. Experiments on digital circuits debugging

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 [5]. 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 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 [5].

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.

The system has been tested on some real world problems taken from the ISCAS85 benchmark circuits [3] that has been used as well for testing the belief revision system REVISE [5].[1]

We have considered the `voter` and `alu4_flat` circuits: `voter` has 59 gates and 4 outputs, corresponding respectively to 59 revisables and 8 constraints, while `alu4_flat` has 100 gates and 8 outputs, corresponding respectively to 100 revisables and 16 constraints.

We have first tested algorithms 1 and 2 on the `voter` and `alu4_flat` circuits. The parameters of the genetic algorithms were 30 for the population and 10 for the number of generations. Both algorithms were run 5 times and the resulting maximum fitness averaged. In table 1 the Fitness column shows the value of the fitness function for the best hypothesis after ten generations averaged over the 5 runs together with its standard deviation, while the Correct solution column shows the percentage of times in which a correct solution was found.

From these results it can be seen that thesis 1 is proved, i.e., that the use of a Lamarckian operator improves the fitness of the best hypothesis. Moreover, the algorithm does not heavily depend on the initial population, as shown by the low values for the standard deviation. Finally, the Lamarckian operator does not greatly influence the dependency on the initial population, as can be

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

Circuit	Algorithm	Fitness	Standard Deviation	Correct solution
voter	2	1.319	0.00415	100 %
	3	1.314	0.00928	100 %
	4	1.325	0.03321	100 %
	5	1.392	0.06296	100 %

Table 2. Experiments with algorithms 2, 3, 4 and 5

seen from the fact that in one case (`voter`) the use of the Lamarckian operator has increased the standard deviation but in the other case (`alu4_flat`) it has decreased it.

Algorithms 2, 3, 4 and 5 have been tested on the `voter` circuit. Each algorithm was run 5 times. The parameters that have been used for the runs are: 10 maximum generations, 40 individuals for algorithm 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.

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

As can be seen, theses 2, 3 and 4 are also confirmed. If we compare the results of algorithm 1 (table 1) and 3 and those of algorithms 2 and 4 we realize that the cooperation among agents improves the quality of the results with respect to the single agent case in the same number of generations (thesis 2). The fitness increment between algorithms 3 and 4 shows the usefulness of the Lamarckian operator in the multi-agent case (thesis 3). Finally, 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). 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 [7, 1, 9, 6]. 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. [7] 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 [7, 1, 9, 6], 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 [7, 1, 9, 6], the procedure does not perform a local search but finds an improvement by tracing logical derivations causally supporting the undesired behaviour.

Our work is also related to coevolutive approaches and distributed GAs (see [12, 13, 4]. It can be considered a cooperative coevolutionary approach(see [12, 13] 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 [12] 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).

According to the classification given in [4], our approach is a multiple-population coarse-grained GA. Multiple-population (or distributed) GAs consist of several subpopulations which exchange individuals occasionally by migration. Rather than migration, we consider instead selection of members of different sub-population to be merged by crossover. This form of *virtual* migration is synchronous with the application of crossover operator. Furthermore, the topology we consider is in fully connected, but again the application of crossover chooses a chromosome from a selected agent.

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, and 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 suplement. Our contribution has been to get the new approach off the ground.

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. 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.
3. 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`.
4. Erick Cant-Paz. A survey of parallel genetic algorithms.
5. 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.
6. J. J. Grefenstette. Lamarckian learning in multi-agent environment. In *Proc. 4th Intl. Conference on Genetic Algorithms*. Morgan Kauffman, 1991.
7. 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.
8. E. Lamma, L. M. Pereira, and F. Riguzzi. Multi-agent 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.
9. 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.
10. T. M. Mitchell. *Machine Learning*. McGraw Hill, 1997.
11. 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.
12. M. Potter and K. de Jong. A cooperative coevolutionary approach to function optimization, 1994.
13. 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.
14. 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.