When Evolving Populations is Better than Coevolving Individuals: The Blind Mice Problem

Thomas Miconi
34 Rue Desbordes-Valmore
75016 Paris, France
thomas.miconi @free.fr

Abstract

This paper is about the evolutionary design of multi-agent systems. An important part of recent research in this domain has been focusing on collaborative revolutionary methods. We expose possible drawbacks of these methods, and show that for a non-trivial problem called the "blind mice" problem, a classical GA approach in which whole populations are evaluated, selected and crossed together (with a few tweaks) finds an elegant and non-intuitive solution more efficiently than cooperative coevolution. The difference in efficiency grows with the number of agents within the simulation. We propose an explanation for this poorer performance of cooperative coevolution, based on the intrinsic fragility of the evaluation process. This explanation is supported by theoretical and experimental arguments.

1 Introduction

Evolutionary algorithms are methods that apply the principles of darwinian evolution to the generation and adaptation of artificial, logical entities (function parameters, rulesets, programs. . .). Their usability as a search technique has been supported both analytically (e.g. the Schema Theorem [Goldberg, 1989] for genetic algorithms), and empirically by uncountable applications. However, the overwhelming majority of these applications are about the generation of individuals.

Collective evolution, that is, the generation and/or adaptation of collaborating populations of agents, has attracted comparatively less attention. There has been significant research in this domain though, especially over the last decade. This research led to algorithms of ever-growing complexity. This paper will first describe some of the work in that field, and more particularily the principle of cooperative coevolution, which seems to be the most popular type of method today.

We then expose what we feel are possible drawbacks of cooperative evolution, and propose a simpler way to adapt the canonical genetic algorithm to the generation of populations. We describe an experiment, based on the "blind mice" problem, and show that while an adapted genetic algorithm works pretty well with that problem, cooperative coevolution has more difficulties. Finally, we give an explanation for these difficulties and for the difference in behaviour with the simpler genetic algorithm.

2 (Some of the) Related work

The simplest way to evolve a team of collaborating agents is to have all agents be identical, that is, to have homogeneous populations. These methods are not really different from individual evolution, except at evaluation time: to evaluate a given genotype, N agents are created out of this genotype instead of just one, and the resulting population is evaluated. Then start again with a different genotype, etc. While being very rigidly constrained, this method makes perfect sense in situations where one does not need heterogeneity at all. This method was used by [N.Zaera *et* «/., 1996) to evolve small groups of fish-like animats, controlled by neural networks, to perform extremely simple tasks (dispersion, aggregation, etc.).

A similar method was used by [Luke, 1998] to evolve competitive teams of soccer players for the Robocup competition [Kitano *et* a/., 1995]. The author used an adapted version of Genetic Programming [Koza, 19921. There was also an attempt at introducing a limited degree of heterogeneity by decomposing teams into small sub-teams (defenders, attackers, etc.) and evolving different program trees for each such subteams. However, because of the enormous search space (and of the delays imposed by the Robocup server software), this approach proved intractable in practice: GP runs took days to produce meaningful results. Lack of time thus prevented the semi-heterogeneous teams from outperforming homogeneous teams.

A way to obtain some degree of heterogeneity is to have only one population and make it change gradually over time, replacing some agents by others based on some evaluation method. These new agents can be obtained by crossover or by duplication with mutation. If there is a way to evaluate the impact of one given individual, it is perfectly possible to perform a simple genetic algorithm over the population. This is, in essence, the idea behind classifier systems (iHolland and Reitman, 1978]), where a set of rules cooperate to control an animat, and where individuals are evaluated after the animates performance through a credit-sharing system.

In the same vein, we proposed a simple scheme in (iMiconi, 2001]), in which all agents were given an arbitrary index, and agents of index K could only mate with agents whose indices fell within the [K-r; K+r] range. Evaluation occured simply by replacing one of the two parents by the offspring, then the second parent, and keeping the best of these two populations (with the possibility of discarding any changes if it decreased the performance of the system). This simple algorithm led to the emergence of sub-species that appeared, grow and shrank according to the needs of the population. An interesting feature of this algorithm was its incrementality, which allowed for long-term, adaptive evolution of the system.

Trying to obtain fully heterogeneous systems brings us to another level of complexity, right into the realm of cooperative coevolution. While coevolution has been most frequently applied in a *competitive* way (by confronting individuals to each other and using the result of this confrontation as an evaluation for individuals), it can also be used in a *cooperative* way, in order to evolve sets of collaborating agents.

In cooperative coevolutionary iPotter and DeJong, 1994] methods, each agent within the system is actually taken from a hidden subpopulation, or *pool.* To evaluate a given individual, it is associated with a set of *collaborators* (one from each other pool) and the resulting population is evaluated as a whole. The resulting score is then attributed to the currently evaluated individual. Based on this evaluation method, the classical GA cycle (evaluate, select and reproduce) is applied to each pool in turn, as many times as needed. In the first version of the cooperative coevolutionary algorithm (CCGA-1) collaborators are chosen by taking the best individual from every pool. However, in the CCGA-2 version, evaluation is refined by re-evalluating every agent with random collaborators, then taking the better score obtained between these two evaluations. The number of collaborators, the way these collaborators are chosen, the way the overall score is computed (averaging the different scores, or taking the best score, or taking the worst score, etc.) are important parameters that can influence the performance of the algorithms. The influence of these parameters has been studied to some extent by [Wiegand *et ai,* 2001], but this study applied only to simple function optimization problems with only two variables.

As happens frequently with good ideas, cooperative coevolution has been (re-)discovered a number of times under different names. Enforced subpopulations (ESP), for example, are exactly like cooperative coevolution, in which each agent is evaluated with only one set of collaborators: the best agents from all other pools. In other words, ESP is the CCGA-1 algorithm. While this method was initially devised for the evolution of neural networks [Gomez and Mikkulainen, 1997], it was successfully applied to multi-agent evolution by [Yong and Mikkulainen, 2001], who used it in a predator-prey simulation. The algorithm managed to find efficient strategies for predators, such as having two predators "chase" the prey while another one blocked it.

For some reason, the idea of simply using the standard genetic algorithm to whole populations seems to have fallen slightly out of fashion. The most probable reason is that it is simply too obvious to be talked about. The second reason is that it does have intrinsic drawbacks, such as a more massive search space. The third one is that it requires a few modifications to be adapted to the evolution of populations. All these aspects are discussed in section 3.2.

3 Two methods for evolving heterogeneous populations

3.1 Cooperative coevolution

Cooperative coevolution is quite an elaborate mechanism. Intuition indicates (and evidence confirms) that by focusing on each and every agent, it requires a huge number of evaluations to converge towards a solution. This algorithm concentrates on optimizing each individual agent in regard with the rest of the population; it is rather different from the more holistic approach of traditional GAs, in which full genomes are manipulated, and the (co-)adaptation of genes emerges naturally from selection, crossover and mutation - at least, in theory.

Why would it not be possible to simply use traditional GAs for the generation of multi-agent systems, regarding whole systems (not just each agent within them) as individuals? A simple answer is that this approach leads to very large genotypes, since the genotype for an "individual" has to code for several agents instead of just one, and the resulting search space might become intractable for GAs. Cooperative coevolution can thus be seen as a simple way to decompose a big problem into several smaller ones, even though these smaller problems are still strongly interlaced with each other.

However cooperative coevolutionary algorithms seem to have an important drawback: they basically evaluate each agent by assessing its impact on the performance of the whole system. The problem is that when the number of agents within the system grows, the influence of one single agent over the system's performance tends to decrease, thus possibly making its assessment more difficult. This may become troublesome when the problem has a stochastic component, as is the case in many simulations. In this case, evaluating the same population several times can lead to different results. The consequence of this may be more important than one might think, as we will see below.

But first, it might be interesting to see how classical genetic algorithms can be adapted to the evolution of populations, and whether these population-oriented genetic algorithms can compete with cooperative coevolution.

3.2 Population-oriented genetic algorithms

Genetic algorithms work by evaluating individuals, selecting some of them according to their performance, crossing them together and mutating them, and starting over again. is possible to apply exactly the same method to whole multi-agent systems. We can evaluate populations, cross them together (thus creating new populations that inherit agents from both parents), mutate them by changing one of their agents, etc.

However, the fact is that multi-agent systems are not simple individuals. They do have an obvious level of decomposition (the agent), and this can be exploited in several ways.

The most obvious idea is that in order to cover the search space efficiently, one must not only make new populations out of existing agents, one must also create new agents. To do this, we may introduce an *inner crossover* operator that allows us to cross two agents together. Thus, when creating a new population by importing agents from both parents, some of these imported agents would actually the result of an inner crossover between agents from the parents.

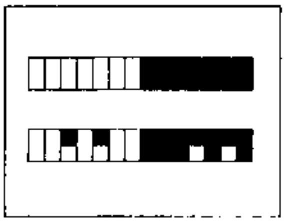

Figure 1: Normal crossover (top) can be "spiced up" with inner crossover between individual agents (bottom). This allows for the creation of new agents, which is necessary to cover the search space efficiently

It is possible to make an analogy with traditional GAs: from the viewpoint of the whole population, these "crossed" agents have some similarity with bit-wise mutations in the standard genetic algorithm. They are part of the children's genotype, yet they were not present in any of the parents' genotypes. However, these are not exactly random mutations, since the genetic material still comes from the parents' genotypes. This suggests that at first sight, "inner crossover" rate should be slightly higher than the usual mutation rate in a classical GA (usually about 2%-5% for each bit).

Another possibility is to enhance traditional crossover by occasionally swapping agents between populations in the final offspring. This, too, could allow for a better covering of the search space. However, we will not explore that possibility in the present paper.

4 Application: The Blind Mice problem

4.1 Description of the experiment

The experiment presented here is based on the "blind mice" problem. A flock of mice, controlled by simple feed-forward neural networks, have to escape a number of cats running after them in a toroidal world.

Now the "game" has three very simple rules:

1. The cats can see the mice and always run after the closest mouse around.
2. The mice run faster than the cats.
3. The mice can *not* see the cats. Neither can they see each other (they are "blind"). Their only input is a pair of numbers: the X and Y coordinates of the center of the flock.

When a cat touches a mouse, the cat is teleported to another, random location, and the population's "catch counter" is increased; nothing else is changed, and the simulation is not interrupted in any way.

Let us consider these rules: they seem to make the problem extremely difficult for the mice. How is it possible to escape predators that can see you, but that you can't sec? Running around as fast as possible will just make them bump into any cat coming from the opposite direction. The same is true for random movement strategies. Given the enormous asymmetry of information between mice and cats, the survival chances of the poor rodents appear to be desperately thin. Even for a human designer, finding a solution to this

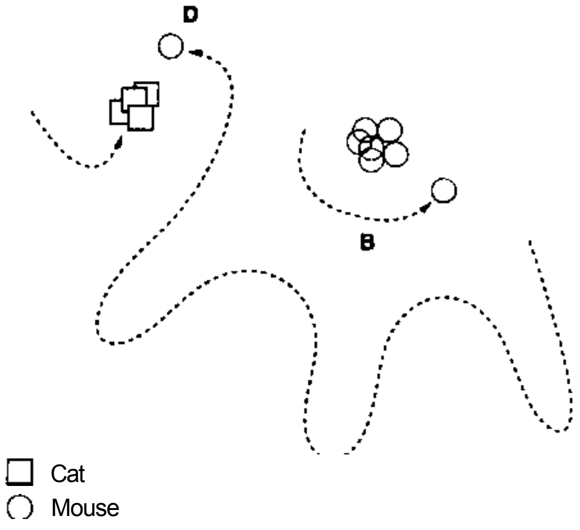

Figure 2: The successful strategy. Mouse D attracts the cats, mouse B plays a "balancing'* role, and other mice move together in a tight flock

problem is not a trivial task. Yet, as we will sec below, evolution managed to come up with an elaborate solution to this problem.

In our experiments, the mice are controlled by simple feed-forward neural networks with 2 inputs, 2 outputs and 5 hidden neurons. The two inputs are the coordinates of the center of gravity of the flock, that is, the sum of the X- (resp. Y-) coordinates of all mice, divided by the number of mice. The two outputs are two real numbers in the [-3.0; +3.0] range, indicating the horizontal and vertical speed of the mouse. All weights are real numbers in the [-1.0; +1.0] range. All simulations use 4 cats.

4.2 The evolved strategy

There seems to be an optimal strategy for this problem. This strategy emerged in all successful runs, sometimes with variants. This strategy is described in Figure 2. No other strategy led to a really efficient behaviour.

Let us explain this strategy: as we can see, most mice are aggregated together and move in a tight flock. This minimizes the probability that a "stray" cat might touch them, but it is not sufficient in itself to ensure a minimal catch rate. The really important behaviour is that of the mouse labelled D (the "Dancer").

This mouse has a strange behaviour: it seems to revolve around the rest of the flock, but not in a strictly circular fashion. Instead, it constantly bounces around the flock, always staying at a respectable distance from it, and moving very fast along its path. The purpose of this behaviour becomes obvious when one sees the position of the cats: they are all following this "dancing" mouse, because it is simply the closest to them, thus leaving the rest of the flock alone.

In other words, the purpose of this dancing behaviour is simply to attract all the cats. The dancer moves very fast (so that it cannot be touched by cats), but along a sinuous path, so that:

- It can "drag" cats more efficiently in the initial stage, when cats and mice are at random position
- It never gets too far from the cats, which allows the cats to follow it endlessly even though it runs much faster than them.

The final touch of this strategy can be seen in the behaviour of the mouse labelled B (as "Balance"), even though it didn't appear in all successful runs. This mouse also revolves around the flock, but much closer to it. In some runs it moves around the flock in a circular fashion, in other runs it bounces around it, but it usually stays on the opposite side of the flock with respect to the dancer. We believe that this mouse has a "balancing" role, in that it counterbalances the effect of the Dancer on the position of the center of gravity of the population, thus allowing the flock to be more stable.

Many variants appeared, such as having several dancers, or no balancer. But the essential traits of the strategy were consistent: aggregation of most mice, except for one or a few to attract the cats away from the flock.

Note that this strategy is very interdependent: each agent's performance is highly dependant on other agents' behaviour. This is even more true when you consider that in oider to behave that way, they must calculate their trajectories out of only one input: the position of the center of gravity of the whole flock, which is based on the position of all other agents. This fact plays a significant role in the results described below.

5 Experimental results

5.1 Experimental settings

We used two algorithms for this problem: a simple genetic algorithm, adapted with an inner crossover operator (as described above), and a full-featured cooperative revolutionary algorithm. Both methods were used with 7, then with 15 mice. We used 100 populations of 7 (resp. 15) mice for the first algorithm, and 7 (resp. 15) pools of 100 mice for the second one. Each algorithm was run several times with different random seeds.

In the first algorithm, reproduction of populations occured through tournament selection and 1-point crossover at a rate of 60%. Every time two populations were thus crossed together, an inner crossover rate of 10% was applied, meaning that each mouse in the offspring had a 10% chance to be the result of the crossing of the parents' corresponding mice. In the second algorithm, reproduction within each pool occured by tournament selection and 1-point crossover at a rate of 30%, which proved to be the most efficient. In both algorithms, mutation appeared only when crossing two mice together, by choosing a new random value for a connection weight with a 5% probability.

Note that in both method, we use a limited form of elitism, in that the best individual from a given generation was preserved in the next generation. This ensured better performance - and made the obtained results even more puzzling, as explained below.

Finally, cooperative coevolution specifies that each agent must be evaluated with a set of collaborators. Several sets of

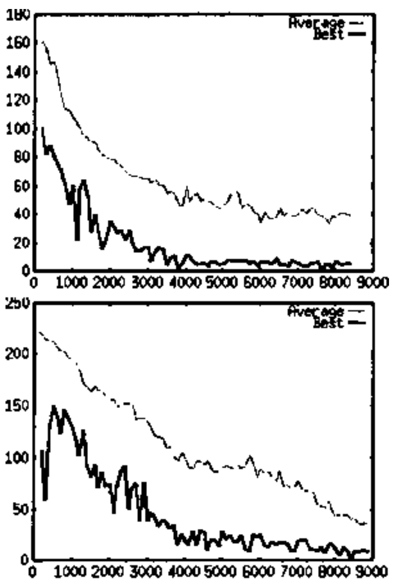

Figure 3: Performance of a population-adapted genetic algorithm, with 7 mice (top) and 15 mice (bottom). The y-axis indicates the number of mice catched during this evaluation round, while the x-axis indicates the number of evaluations. Both the fitness of the best population and the average fitness of all populations are shown.

collaborators can be used in turn in order to refine the evaluation, and the final result can be calculated from these successive evaluations in various ways (average, best, random...) However, we found that with this problem, increasing the number of collaborators (and thus the number of evaluation rounds) brought absolutely nothing, and was even damaging if the final score was anything else than the best score found. The most successful method was simply to evaluate each mouse by joining it with the best individual from each pool, exactly as in the enforced subpopulations algorithm. This is not a surprising result, however, as we will explain it below.

5.2 Comparison of results

The first algorithm (simple genetic algorithm with two levels of crossover) proved remarkably efficient with this problem. All runs led to the strategy described in section 4.2, whatever the number of mice within the simulation, although of course it took more time with 15 mice than with 7. Two typical runs, with respectively 7 and 15 mice, are described in Figure 3.

Cooperative coevolution led to different results. With 7 mice, in some runs, the algorithm failed to evolve any competitive behaviour. In other runs it managed to find the good strategy, but took more evaluations than with the previous algorithm. Many runs, however, achieved performance comparable with that of the simple genetic algorithm. With 15 mice, the success rate was much lower. Most runs did not converge after 10000 evaluations. Others converged, then suddenly diverged quickly. All the runs, with 7 or 15 mice, exhibited a intriguing pattern of oscillation.

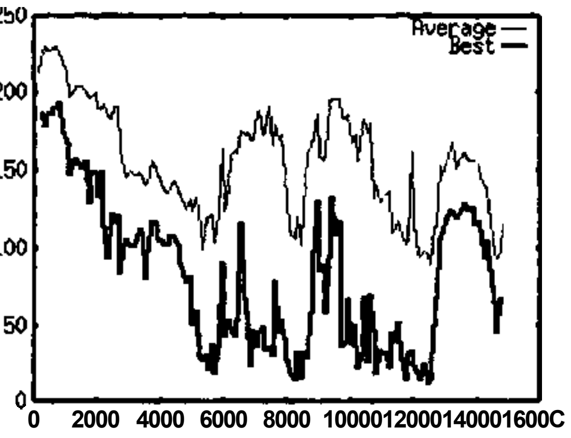

Figure 4: Performance of the best run for a cooperative co-evolutionary algorithm. The curves indicate the performance of the best individual in the currently evaluated pool, and the average fitness of individuals in this pool. Notice the brutal variations in the curves.

The fitness curve described in Figure 4 shows a good example of this pattern. It is the result of the most successful run with 15 mice. The population seems to converge towards a better behaviour, but then it suddenly diverges and seems to loose all that had been found. The pattern starts again a few cycles later.

This is not what one could expect. In this algorithm, the population remains quite stable, being composed of the best individual from each pool. Only one individual can be changed at each cycle, and only for a better one, since the best individual from previous generation is preserved. So how do we explain these brutal changes in fitness happening every now and then?

Our explanation is that this pattern is the result of cooperative revolution's main drawback: when the number of agents grow, and when the result of evaluation undergoes important stochastic variations, its performance is bound to decrease.

What happens is that at some point, the algorithm does find a "good" population, and keeps optimizing it by replacing each agent by a better one in turn - as it should do. But then at some point, because of the stochastic variations in the evaluation process, one of these "good" agents is found to perform not as well as another, non-optimal one. In classical evolutionary algorithms, these occasional mistakes arc blurred over a large number of trials and errors, and besides, one population's performance has no impact on another's. But in cooperative coevolution, this exceptional case is sufficient to wreck the behaviour of the whole population, because it can replace an essential agent (say, a Dancer) by a poorly performing agent (say, a random wanderer).

Then other agents from other pools are evaluated in turn, but since the conditions have changed dramatically, other agents are selected instead of the previous, quasi-optimal ones. Thus, an error in the selection of one agent has an influence over all other pools. When the evaluation cycle comes back to the first pool, the previously essential agent may not be selected again, because the rest of the population has co-adapted towards a different state. However, it is possible that the this agent is selected again, thus resulting in a sharp increase in performance - until the next "error" in the evolutionary process. This seems to cause the up-and-down oscillations in the fitness curve.

There is little we can do about this problem. As we said, increasing the number of collaborators in a classical way (that is, by selecting random sets of collaborators) simply does not work, which is understandable: in this problem, the solution requires highly interdependent behaviours. An agent's performance can only be good if other agents play their role. Evaluating each agent by fitting it in a random population is not likely to help reach this state. In a problem with many strongly interdependant agents, adding random collaborators doesn't seem to be a helpful solution. *

We could devise more elaborate schemes (e.g. using the second-best from each pool as a second set of evaluators) but that would not suppress the problem. It seems that in problems with a non-deterministic component, cooperative coevolution is essentially fragile, or at least, more so than the standard genetic algorithm.

6 Conclusion

We have described possible pitfalls in cooperative coevolution, and we have proposed a way to apply the canonical genetic algorithm to populations of agents. We have applied both algorithms to a non-trivial problem, the blind mice problem, that is both conceptually simple and non-trivial to solve. For this problem, the populations-oriented genetic algorithm significantly outperformed cooperative coevolution. We proposed an explanation for the difference in results, based on the intrinsic fragility of cooperative coevolution in regard to excessive stochastic variations in the evaluation proces.

As a final note, an old French proverb goes: "Quand le chat n'est pas la, les souris dansent" (When the cat is away, the mice dance). Artificial evolution has demonstrated that the opposite can be true as well.

References

[Goldberg, 1989] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, 1989.

[Gomez and Mikkulainen, 1997] F. Gomez and R. Mikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior,* 5:317-342, 1997.

[Holland and Reitman, 1978] J. H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. In

'Strangely enough, in [Potter and DeJong, 1994], random collaborators were introduced precisely for problems with strongly interacting variables, such as the Roscnbrock function; however, for this kind of problem, this idea was only applied to two-variables functions. In this case, choosing a random collaborator in addition to the better one may allow the algorithm to escape a local minima. But as the number of agents grows, it seems that evaluating one agent with a set of totally random collaborators could hardly bring any valuable information.

D. A. Waterman and F. Hayes-Roth, editors, *Pattern Directed Inference Systems,* pages 313-329. Academic Press, 1978.

[Kitanoe/a/., 1995] H. Kitano, M. Asada, Y. Kuniyoshi, 1. Noda, and E. Osawa. Robocup: The robot world cup initiative. In *Proceedings of IJCAI-95 Workshop on Entertainment and AI/ALife*$_y$ 1995.

iKoza, 1992] JohnKoza. *Genetic Programming.* MIT Press, 1992.

[Luke, 1998] Sean Luke. Genetic programming produced competitive soccer softbot teams for robocup97. In John Koza, editor, *Proceedings of the Third Annual Genetic Programming Conference (GP98),* pages 204-222, 1998.

[Miconi, 2001] Thomas Miconi. A collective genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-200J),* pages 876-883. Morgan Kaufmann, 2001.

[N.Zaerae/a/., 1996] N.Zaera, D.Cliff, and J.Bruten. (not) evolving collective behaviours in synthetic fish. In *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (SAB96),* pages 635-644. MIT Press, 1996.

[Potter and DeJong, 1994] M. A. Potter and K. A. DeJong. A cooperative revolutionary approach to function optimization. In *Proceedings of The Third Parallel Problem Solving from Nature,* pages 249-257. Springer-Verlag, 1994.

[Wiegand *etal,* 2001] R. Paul Wiegand, William C. Liles, and Kenneth A. DeJong. An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001),* pages 1235-1242. Morgan Kaufmann, 2001.

[Yong and Mikkulainen, 2001] C.H. Yong and R. Mikkulainen. Cooperative coevolution of multi-agent systems. Technical Report AI01-287, University of Texas at Austin, 2001.