Incremental Evolution with Minimal Resources

Joseba Urzelai and Dario Floreano

Microprocessor and Interface Lab (LAMI)
Swiss Federal Institute of Technology (EPFL), CH-1015 Lausanne

E-mail: {name.lastname}@epfl.ch

Abstract

This papers describes an evolutionary algorithm based on a statistical representation of populations of individuals. Experiments on robot navigation and on numerical fitness functions are presented in order to measure the performance of the algorithm compared to traditional genetic algorithms. Results show that the method is suitable for onboard online evolution because it requires low amount of memory resources. Furthermore, it allows for incremental evolution in dynamic environments in order to cope with complex tasks that require several evolutionary stages.

1 Introduction

Evolutionary algorithms are often used for design and control of electronic hardware where traditional engineering displays its limits, such as in evolvable hardware [11], or where the system is expected to have some form of adaptation to variable external conditions, such as in autonomous robotics [8]. The most interesting features of evolutionary methods, which are exploited by both evolvable hardware and evolutionary robotics, are stochastic parallel sampling and selectionist update of a population of encoded solutions (genetic strings). This method not only allows the user to encode and co-evolve several components at different levels of the system, but it also requires only a (simple) specification of the desired performance in order to guide its search. This is qualitatively different from the type and amount of information required by gradient descent/ascent techniques. If crossover of selected individuals is not employed, evolutionary methods might have some similarities with simulated annealing [4].

However, evolutionary approaches might require significative time and memory resources in order to produce the desired solution. Time is necessary to test all the individuals of the populations across several generations, memory is required for storing all the individuals (encoded solutions) of the current population. These requirements might be critical constraints for applications where it is necessary to use fast and compact systems, such as in adaptive autonomous robots [10].

Time constraints can be alleviated by adopting an incremental strategy where controllers are first evolved in carefully-designed simulations, then incrementally evolved in a stripped-down version of the physical robot and of the task, and later fine-tuned on the robot and task required by the final application [6]. An incremental approach has also the advantage of finding solutions to problems that could not be otherwise learned from scratch [9].

In order to allow incremental learning and reduce the demand on memory resources, we introduce two variations of an evolutionary algorithm named *Population Based Incremental Learning* [3]. These variations further reduce storage requirements, accelerate learning, and allow online adaptation to dynamically changing environments. We call the new algorithm APBIL (Adaptive PBIL). In section 2 we describe the implementation of the original PBIL algorithm on a mobile robot and compare its performance with a standard genetic algorithm; in section 3 we introduce an adaptive learning rate and run a benchmark test. Then, in section 4 we introduce a decay factor and test it in dynamic environments, including an incremental situation. Section 5, gives a discussion of APBIL in light of evolutionary robotics and dynamic environments.

2 Probability vs Population

PBIL works on a statistical representation of the population of individuals. A population of n binary vectors I^p can be encoded on a single real-valued vector P by storing at each position P_i the probability of finding a 1 at that position in all the n individuals of the population. The probability vector, which is initialised so that $P_i = 0.5\,\forall i$, is updated by moving the probability values P_i into the direction of the average value $\hat{I}_i$ observed at the same location in the best s individuals of the sampled population

$$P_i^g = (1-\eta)\,P_i^{g-1} + \eta \hat{I}_i^g \tag{1}$$

where g is the current generation, i.e. the evaluation of n sampled solutions from the probability vector P, and η is the learning rate. At any given time PBIL requires only storage of the probability vector and of the s best sampled individuals used for update.

We have implemented PBIL algorithm on a Khepera robot in order to solve a simple navigation task and compared its performance to a standard genetic algorithm. The environment consisted in a sort of circular corridor whose external size was 80x50 cm large [5]. The Khepera could sense the walls with its IR proximity sensors. The goal was to evolve a strategy that could maximize straight navigation while avoiding all the obstacles in its way. The fitness function was the same for both algorithms and it was directly measured on the robot as follows,

$$\Phi = V\left(1-\sqrt{\Delta v}\right)(1-i), \qquad 0 \leq V, \Delta v, i \leq 1 \tag{2}$$

where V is a measure of the average rotation speed of the two wheels, Δv is the algebraic difference between the signed speed values of the wheels (positive is one direction, negative the other) transformed into positive values, and i is the activation value of the proximity sensor with the highest activity. The function Φ has three components: the first one is maximized by speed, the second by straight direction, and the third by obstacle avoidance.

Both a simple genetic algorithm (SGA) [7] and PBIL were evolved for 180 generations. A population of the standard genetic algorithm was composed of 50 individuals, and mutation and crossover rate were 0.2 and 0.1, respectively. In the case of PBIL, 50 individuals were sampled at each generation and three best individuals were used for probability update, with 0.05 learning rate. In both cases each individual of the population was tested on the real robot during 80 motor actions. Each motor action lasted 300 ms.

Results show that standard genetic algorithm converges faster than standard PBIL, although PBIL is capable of reaching the same performance level given enough time.

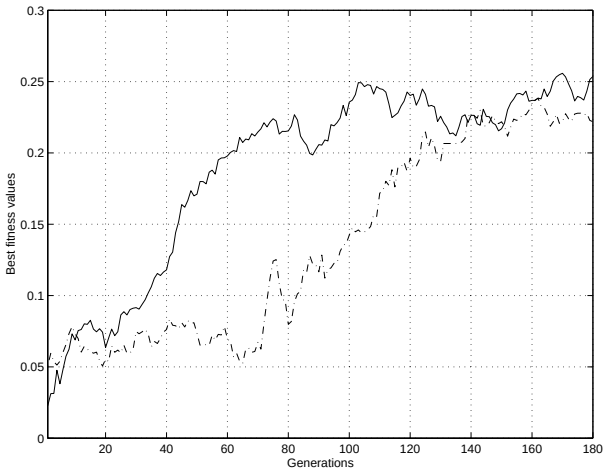


Figure 1: Fitness values of best individuals for the robot navigation task. Solid line plots values obtained with a standard genetic algorithm. Dashed line reports values obtained with standard PBIL. Plotted data are rolling averages (window size=5).

3 Adaptive Learning Rate

In its original version, PBIL uses a subset s of individuals at each generation for updating the probability vector. If s is too small or too large, PBIL search might be too weak or too noisy to find a good solution.

However, the number of individuals that might be usefully exploited for updating the probability vector could vary from generation to generation. Ultimately, whether an individual can positively contribute to the update—and to what extent—depends on the quality gain of its solution with respect to the average performance of the population encoded in the probability vector. In principle, all the individuals that display a better solution to the problem should be enabled to give a contribution proportional to their performance gain. On the basis of this idea, we propose an automatic method for selecting the best individuals and adjusting the extent of their contribution to the update of the probability vector. The method consists in setting $s = n$ (where n is the population size, i.e. the number of individuals sampled from the probability vector) and using an adaptive learning rate

$$\eta^p = \begin{cases} \lambda(f^p - \hat{f}_{g-1}) & : \quad f^p > \hat{f}_{g-1} \\ 0 & : \quad otherwise \end{cases} \tag{3}$$

where $0 \leq \eta^p \leq 1$ is the proportion of individual p used for updating the probability values, $0 < \lambda < 1$ is a limiting constant, and $(f^p - \hat{f}_{g-1})$ is the difference between the fitness f^p achieved by individual p and the average fitness $\hat{f}_{g-1}$ of the population at the previous generation $g - 1$.

Simply put, all the individuals reporting a fitness score higher than the average fitness score of the population at the previous generation are used to update the probability vector proportionally to this difference. At the beginning of the evolutionary process, good individuals might perform strong updates but, as the population converges towards a better solution, the updates become smaller and smaller because the average population fitness approximates that of the best individuals. The constant $\lambda < 1$ limits the entity of initial updates, preventing premature convergence or strong oscillations when the fitness differences are very high.

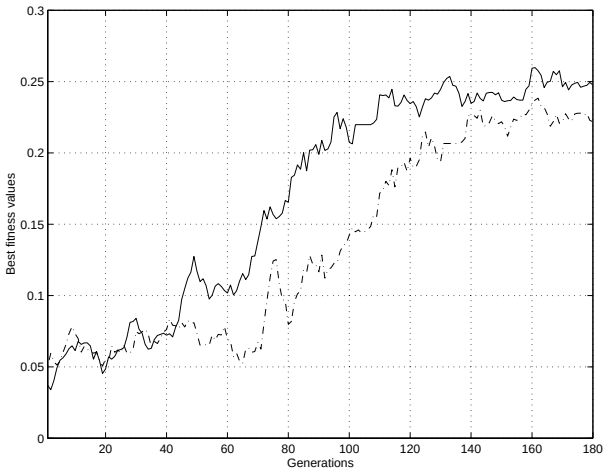


Figure 2: Fitness values of best individuals for the robot navigation task described in section 2. Solid line plots values obtained with adaptive learning rate (λ=0.15). Dashed line reports values obtained with standard PBIL (see figure 1). Plotted data are rolling averages (window size=5).

When the robot navigation experiment described in the previous section was repeated using the adaptive learning rate (figure 2), results were better than those obtained with the standard PBIL algorithm and comparable to those obtained with a standard genetic algorithm. Moreover, the adaptive learning rate suggested here allows an even further reduction in memory space. Since all individuals are sequentially sampled from the probability vector and all might differentially contribute to the update, it is necessary to allocate space for only three vectors, whatever the population size might be: the probability vector $\mathbf{P}^g$ at that generation g, the currently sampled vector $\mathbf{I}^p$, and an incrementally-updated copy of the probability vector $\hat{\mathbf{P}}^{g+1}$. Once a desired number n of individuals has been sampled and evaluated, the updated copy of the probability vector substitutes the previous probability vector and a new copy is created for incremental update.

4 Dynamic Environments and Probability Decay

The probability vector is a compact representation of a population of binary solutions. It can be visualized as a point in a hypercube whose dimensions are equal to the components of the vector. Initially, the population vector occupies the central position of the hypercube ($\mathbf{P}_i = 0.5\ \forall i$) and during learning it tends toward one of its vertices ($\mathbf{P}_i = \{0.0, 1.0\}\ \forall i$). As the probability values get closer to either 0.0 or 1.0, the sampled individuals tend to be very similar until they will eventually be equal to the probability vector itself. Once the probability vector has reached a vertex, it can no longer move to a different zone of the search space. Provided that the learning phase has been carefully completed and that the function to be discovered is static (as in all the experiments carried out by Baluja [3, 2]), this convergence is not harmful. However, a partially or fully converged probability vector might have serious limitations in a dynamic environment where either the external conditions or the selection criterion might unpredictably change. In such case, probabilities that have reached either 0.0 or 1.0 cannot reverse their value.

We propose the utilization of a decay function that moves each probability value toward the equilibrium position 0.5. The influence of this function should be weak when a probability is close to 0.5 in order to allow the algorithm to move toward higher fitness areas, and strong when

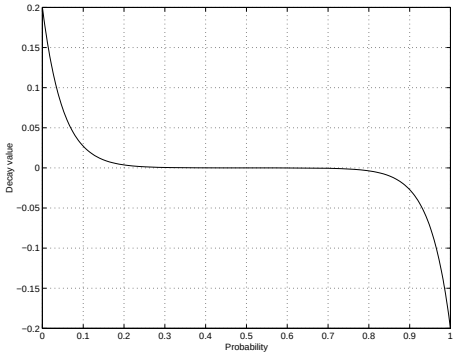


Figure 3: Probability decay function (α=0.2, β=20).

a probability is closer to 0.0 or 1.0 in order to preserve population diversity. There are several simple functions that satisfy this criterion. We have used the difference of two exponentials to define the decay function $D(x)$,

$$D(x) = e^{-\beta x + \ln \alpha} - e^{\beta(x-1) + \ln \alpha}, \tag{4}$$

where α is the maximum decay value when probabilities are either 0.0 or 1.0, and β controls the curvature of the function (figure 3).

The decay function (4) is applied by adding it to the update rule (1) as follows,

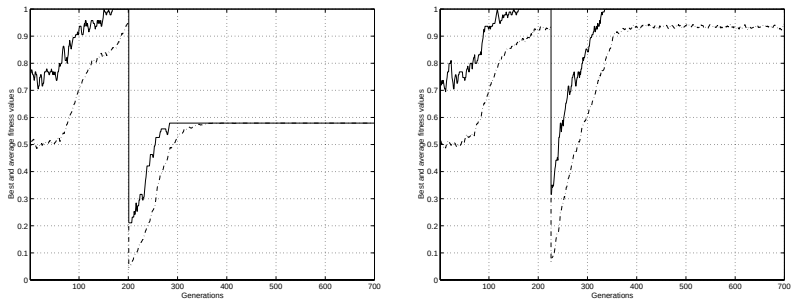
$$P_i = (1 - \eta)P_i + \eta I_i + D(P_i). \tag{5}$$

This new version of the algorithm that includes adaptive learning rate and decay is named Adaptive PBIL (APBIL). We experimentally studied the effect of probability decay in a simple dynamic environment where the fitness function changes while APBIL is converging to a solution. Initially, the fitness function was proportional to the number of adjacent positions on the sampled individual with the same value (either 0 or 1). A standard PBIL without probability decay and APBIL with the decay function described above were run until average fitness of sampled population was 0.95. This means that 95% of the bits have the same value. The fitness function was then changed so that it becomes proportional to the number of adjacent positions with different value (e.g., 010101..., or 101010...). Whereas PBIL was not capable of fully re-adapting to the new fitness function (figure 4, left), APBIL quickly generated individuals that completely satisfied it (figure 4, right). It should be noticed that when probability decay is applied the probability vector never saturates and therefore the average fitness of the sampled population is smaller than that of the best individual.

4.1 Incremental Evolution

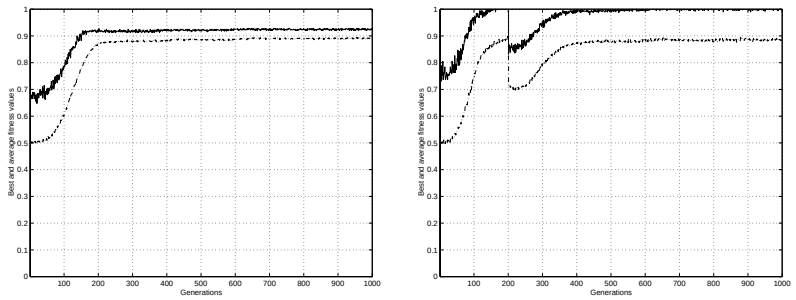
Another way of exploiting APBIL consists of evolving the system on two related tasks, where the first is a subset (simpler) of the second. This is a case of incremental evolution useful for complex problems that cannot be solved starting from scratch [9].

We have performed a numerical experiment in order to analyze incremental evolution of complex tasks using APBIL. The task to be accomplished was the second one described in Section 4. The goal was to generate alternating sequences of 0 and 1 on the chromosomes (i.e. 010101...



Standard PBIL APBIL

Figure 4: Fitness values during incremental evolution on two antagonist tasks. Dashed line shows the average population fitness, continuous line the fitness of the best individual at each generation. Left: In PBIL transition between two tasks is not feasible. Right: The decay function in APBIL makes possible the transition from one task to the other. Plotted data are rolling averages (window size=5).



Evolution in one step Incremental evolution

Figure 5: Fitness values during evolution of a complex task with APBIL. Dashed line shows the average population fitness, continuous line the fitness of the best individual at each generation. Left: APBIL is not capable of finding an optimal solution in one step. Right: APBIL is evolved on a simpler version of the task for 200 generations and then incrementally evolved on the full task. Plotted data are averaged over 10 replications.

or 101010...), and the fitness function was proportional to the number of adjacent positions with different value. The task here was more complex than in the previous case because the chromosomes (and the probability vector) were longer (40 bits here and 20 bits in the previous case). This is shown by the fact that now APBIL is not capable of solving the task in one step. After 200 generations, the evolutionary process got stuck in a sub-optimal solution caused by the *border effect* (Figure 5, left). During the first generations, APBIL created groups of alternating series in the chromosome. These series became longer and longer until they met each other. At this point, in some cases borders of different series did not match correctly to form a unique long series and fitness values remained always below the optimal value. In the second run of the experiment, we divided the problem in 2 parts. First, we solved the problem for the left half of the chromosome evolving the initial population (where all the probabilities were 0.5) for 200 generations. Notice that at this stage the chromosomes are always 40 bits long. After this initial stage, the population showed a high degree of convergence and most of individuals had an optimal configuration (alternation of 0 and 1 on the portion of the chromosome considered by the fitness function). In the next stage, we changed the fitness function in order to take into account both halves and solve the problem for the entire length of the chromosome. After additional 200 generations of incremental evolution, some of the samples displayed perfect solutions (5, right). APBIL was capable of incrementally solving a task that in principle was not possible to

solve in one step.

5 Conclusion

We have showed that adaptive learning rate and a decay function improve the performance of standard PBIL algorithm. In particular, adaptive learning rate minimizes memory resource requirements and speeds up convergence. These are crucial aspects when evolution is carried onboard. Large memory resources demanded by standard genetic algorithms can be a problem for embedded evolution of robots with small memory size (mainly miniature systems). The extremely low memory requirements make APBIL a *hardware-friendly* algorithm.

Adaptivity to dynamic environments is another important aspect of APBIL. Although high convergence of the population in standard PBIL may be harmful in the case of static function approximation, it has been shown that it has catastrophic effects in the case of dynamic environments. Baluja proposed to apply a mutation to the probability vector in order to increase exploration when probabilities converge towards 0 and 1 [1]. Here instead, we have incorporated a decay function that automatically preserves diversity of the population. APBIL maintains high levels of performance in dynamic environments even with highly converged probability vectors. Future work will study the performance of APBIL on incremental evolution applied to robotic tasks.

Acknowledgements

J.U. is supported by grant nr. BF197.136-AK from the Basque government. D.F. acknowledges support by the Swiss National Science Foundation, project nr. 21-49174.26. Thanks to Wulfram Gerstner for useful comments on adaptive learning rate.

References

[1] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Carnegie Mellon University, 1994.

[2] S. Baluja. Genetic algorithms and explicit search statistics. In M.C. Mozer, M.I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 319–325, Cambridge, MA, 1997. MIT Press.

[3] S. Baluja and R. Caruana. Removing the genetics from the standard genetic algorithm. In A. Prieditis and S. Russel, editors, *Proceedings of the Twelfth International Conference on Machine Learning (ML'95)*, pages 38–46, San Mateo, CA, 1995. Morgan Kaufmann. Also available as Technical Report CMU-CS-95-141, School of Computer Science, Carnegie Mellon University.

[4] L. M. Davis. *Genetic Algorithms and Simulated Annealing*. Morgan Kaufmann, San Mateo, CA, 1989.

[5] D. Floreano and F. Mondada. Automatic Creation of an Autonomous Agent: Genetic Evolution of a Neural-Network Driven Robot. In D. Cliff, P. Husbands, J. Meyer, and S. W. Wilson, editors, *From Animals to Animats III: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 402–410. MIT Press-Bradford Books, Cambridge, MA, 1994.

[6] D. Floreano and F. Mondada. Hardware Solutions for Evolutionary Robotics. In P. Husbands and J-A. Meyer, editors, *Proceedings of the first European Workshop on Evolutionary Robotics*. Springer Verlag, Berlin, 1998.

[7] D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, Redwood City, CA, 1989.

[8] T. Gomi. Robotics and Emerging Business Principles. In T. Gomi, editor, *Evolutionary Robotics. From Intelligent Robots to Artificial Life*, Kanata, Canada, 1997. AAI Books.

[9] I. Harvey, P. Husbands, and D. Cliff. Seeing The Light: Artificial Evolution, Real Vision. In D. Cliff, P. Husbands, J. Meyer, and S. W. Wilson, editors, *From Animals to Animats III: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*. MIT Press-Bradford Books, Cambridge, MA, 1994.

[10] M. Mataric and D. Cliff. Challenges in Evolving Controllers for Physical Robots. *Robotics and Autonomous Systems*, 19(1):67–83, 1996.

[11] E. Sanchez and M. Tomassini, editors. *Towards Evolvable Hardware*. Springer-Verlag Notes in Computer Science, Berlin, 1996.