Some Thoughts on the use of Sampled Fitness Functions for the Multilevel Darwinist Brain

F. Bellas and R. J. Duro

Grupo de Sistemas Autónomos, Universidade da Coruña, Spain
fran@cdf.udc.es, richard@udc.es

Corresponding Author:

Prof. Dr. Richard J. Duro
Grupo de Sistemas Autónomos
Escuela Politécnica Superior
Universidade da Coruña
c/ Mendizábal s/n
15403 Ferrol (A Coruña)
Spain

Phone: 34-981-337400 ext 3281
Fax: 34-981-337410
e-mail: richard@udc.es

Some Thoughts on the use of Sampled Fitness Functions for the Multilevel Darwinist Brain

F. Bellas and R. J. Duro

Grupo de Sistemas Autónomos, Universidade da Coruña, Spain

fran@cdf.udc.es, richard@udc.es

Abstract. When attempting to evolve models of the real world through the information obtained by interacting with it, we always come across the same problem: the fitness function for the problem, that is, the real world, can only be known in a sampled manner. In this article we study the effect of different parameters and algorithmic strategies when working with sampled fitness functions in evolutionary processes. The results presented here correspond to a study of the effect of the size of the Short Term Memory and the number of generations between updates for a generic genetic algorithm that will be operating within the Multilevel Darwinist Brain. From these results, several critical points may be considered in order to define the limits to which computations can be simplified when working with sampled fitness functions while maintaining the same representational power. We provide a study of different proposals for the construction of Short Term Memories and their replacement strategies in order to obtain the maximum information with the minimum use of resources.

1. Introduction

The Multilevel Darwinist Brain (MDB) is a proposal for a Darwinist cognitive mechanism intended for autonomous artificial organisms that must generate original solutions and use previously acquired experience when negotiating new environments and situations. The initial ideas for MDB were introduced in [1], [2] and [3]. In this paper we will contemplate a particular aspect of its implementation, that is, the unavoidable use of sampled fitness functions, and provide some results on how to benefit from the dynamicity of environments so as to implement coherent models with limited computational resources. Most of the results obtained may be extended to other evolutionary based problems which require the use of sampled fitness functions.

In the realm of evolutionary computation many applications involve obtaining fitness data from partial samplings of the fitness function [4-11]. This fitness determination may be carried out through an explicit mathematical function that is partially sampled, or through some type of environment that provides performance information for different actions carried out by the entity resulting from the phenotypic representation of the genotype. In any case, for every individual and determination of fitness, the data employed is different, although part of the same function. Consequently, if the required optimal individual must achieve optimality for the whole fitness function, obtaining it implies the integration of series of partial snapshots of the fitness function. The contents of these snapshots depend on how the fitness information is presented to the evolutionary process.

If we translate this into a more formal computational environment, we have a problem of obtaining a general representation of some type of function through the presentation of sequences of partial data samples or frames. The traditional approach to the problem

has been to wait until enough frames had been compiled and use the whole of the information to obtain the representation [6]. This is what we would call a full (or very close to full) information fitness function. In the case of the agent-environment interaction, each agent would be made to interact with the environment for a very long period of time. This is all so well when we have a simple and static process which does not change, independently of how long it takes us to gather the information, and where the relevant data, which must be considered to adequately model the process, can be gathered and stored in reasonable time and space. It does not perform so well when we are working with complex dynamic processes. In this case, performing evolution with anything close to full information fitness functions becomes computationally very expensive. In fact, it often becomes unapproachable.

In this paper, we will try to provide some insight into the problem and present some results that will help to decide how to apply sampled fitness functions in order to obtain results that are often as good as with the complete fitness function.

The paper is structured as follows: in section 2 we will review the main features of the Multilevel Darwinist Brain and its operation. Section 3 is devoted to the presentation of the main problem derived from sampled fitness functions, the parameters we considered and the approach we follow to obtain some generalizable information on how to benefit from the dynamicity of the world. In Section 4 we formalize a series of critical points that may be used in order to determine the limits of behavior regions. Finally, in section 5 we propose some Short Term Memory replacement strategies which greatly improve results and in section 6 we present some conclusions.

2. Multilevel Darwinist Brain

A Multilevel Darwinist Brain (MDB), which is extensively explained in [1-3], is an approach to implement a mechanism that is able to allow an organism to interact with the environment and learn from it so that it can improve its performance in time without the designer predetermining future behaviours, in a computationally efficient way. To this end we have considered several concepts. On one hand, we have contemplated *Strategies* as sequences of actions applied to the effectors of the agent. An ideal strategy would depend on the current external perceptions, internal state and motivations. On the other hand, as stated elsewhere, a *World model (𝒲)* is a function that relates the sensory inputs of the agent in the instant of time t to the sensory inputs in the instant t+1 after applying a strategy. A world model permits evaluating the consequences of actions on the environment as perceived by the agent (sensed values in t+1). An *Internal model (ℐ)* is a function that relates the sensory inputs in instant of time t+1 with the internal state according to the motivation for the agent. An internal model permits predicting the internal state in instant t+1 from the sensory inputs for t+1 and the motivation. That is, it provides an indication of a satisfaction factor the agent uses to make decisions. Finally, an a*ction-perception pair* is a set of values made up by the sensorial inputs and the internal state obtained after the execution of a strategy in the real world. It is used when perfecting world and internal models.

The problem is how to connect these elements into an on line mechanism that permits the agent to generate original solutions making use of previous experience, and without the intervention of a designer, so that the motivations are fulfilled.

The way we have solved the problem is shown in the block diagram of Fig. 1, which represents the whole mechanism. The final objective of the mechanism is to obtain the

strategy the agent must execute in the real world to satisfy its motivations. This selected strategy is represented in the block diagram of Fig. 1 by the block labeled *Current Strategy* and is applied to the *Environment* through the *Effectors* providing new *Sensing* values for the agent. Thus, after each iteration, we have an action-perception pair obtained from the real world. These action-perception pairs are stored in the *Short-Term Memory* and are used as the fitness function for the evolution of the world and internal models.

We have three concurrent evolutionary processes in our mechanism that are represented in Fig. 1:

- *World Model Evolver:* structure that manages world models through evolution. This structure evolves a population of world models and each moment of time selects the best one according to the information it has available about the real world.
- *Internal Model Evolver:* manages an internal model base, evolving it and providing the best internal model available when it is requested.
- *Strategy Evolver:* manages and evolves strategies and when the agent needs one it provides it.

Each one of these evolutionary processes starts from the population stored in a memory (*World Model Memory, Internal Model Memory and Strategy Memory*). The contents of these memories are random for the first iteration of the mechanism. These contents are maintained from one iteration to the next in order to obtain a generally good population of models and not a superindividual. The individual with the highest fitness value after evolution according to the information of the short-term memory is selected as the current model, both in the case of world and internal models. These

current models are represented in Fig. 1 by the *Current World Model* and *Current Internal Model* blocks .

In the case of strategies, these are tested during their evolution by implementing a virtual environment using the current models, that is, the current world model provides the sensory inputs in instant t+1 for the strategy we want to test. The current internal model uses these predicted values as input to predict the internal state corresponding to the strategy according to the agent's motivation. The strategy with the highest fitness value is selected as the *Current Strategy* and is applied to the *Environment* through the *Effectors*, obtaining again new *Sensing* values and thus a new *action-perception pair* that is stored in the *Short-Term Memory*.

This basic cycle is repeated and, as time progresses, the models become better adapted to the real world and the selected strategies improve in their efficiency to fulfil the agent's motivations.

As we display in Fig. 1, the operation of the mechanism can be divided into two parts: first the internal operation, which may be assimilated to a thought process. That is, during this stage, and without any interaction with the external world, the mechanism explores possible solutions to its problems. It uses the information it has compiled through its models and what it knows of the environment to obtain better models, deciding in the end the best strategy to employ. In external operation the selected strategies are applied in the environment, producing new values the sensors can measure and thus, producing new information the agent can use to improve its models of the world and of itself. Consequently, we can say that the environment through the sensory inputs evaluates the world models and the internal models. The current model through the predicted satisfaction of the motivations evaluates the strategies.

2. Our Problem

As shown in the previous section, the Multilevel Darwinist Brain bases its efficiency in its ability to obtain good world and internal models for autonomous agents through their on-line interaction with the world, whether real or simulated. The information the agent receives from the environment in any given instant of time, and which it must use to evaluate its models, is partial and, in many cases, noisy or irrelevant. Any model extracted from this instantaneous information will be, in most cases, useless for other instants of time. To generalize appropriate models, the agent will have to consider and relate information obtained in many different instants of time in an efficient manner.

In more mathematical terms, and considering a set of data that define a function, the fitness of the model is given by how well it fits the whole function and not by how well it fits the individual point, or the subset of points, it is considering in a given instant of time. This is what we have called a Sparse or Sampled Fitness Function (SFF); the fitness of the individual is given by its response to several different instantaneous local fitness functions, which together conform the desired global one. Thus, perfect knowledge of the fitness function in a given instant of time is impossible, in addition, if one attempts to improve instantaneous fidelity of the fitness function through the use of lengthy historical records of events the operation of the mechanism becomes very slow and cumbersome. Consequently, one would like to know the shortest sequence of previous values one would require to obtain a given level of fitness for the model, that is, the optimal Short Term Memory length and the length of time (in terms of evolution generations) a given snapshot of the STM should be used before updating it.

In order to illustrate the problems encountered, we have performed tests on three benchmark functions of increasing difficulty in order to obtain generalizable indications.

- Sin function: $f(x)= \sin(x)$
- Logistic series: $x(t) = 4x(t-1)/1-x(t-1)$
- 3D function: $f(x,y) = x\sin(x) + y\sin(y)$

The idea was to obtain a neural network that would model these functions through evolution. Each network consisted of 1 input neuron, 2 hidden layers with 4 nodes and 1 output neuron both for the *sin()* and the logistic functions. In the case of the 3D function we have used a neural network with 2 input neurons, 2 hidden layers of 6 neurons each and 1 output neuron. The evolution was carried out using 350 individuals (700 for the 3D function). The fitness of each individual was obtained by determining the mean squared error (MSE) between the output of the network and each one of the points of the signals contained in a Short Term Memory (STM). This STM contains a window of data, with one item per instant of time, of the signal that is fed to the STM replacement mechanism.

How big the STM is and how many generations of the evolutionary algorithm we perform per data input are the main two parameters in terms of computing resource usage. Each item in the STM must be run through the current model once per individual and generation so as to obtain an error term. Obviously, the smaller the number of individuals in a population, the smaller the STM, and the less generations per data input cycle we run the evolutionary algorithm, the better in terms of resources. The problem

regarding population size is that, as commented in [4], there is a lower limit we can use before running into premature convergence problems. Consequently, we will have to concentrate on the other two terms.

Figure 2 displays the evolution of the MSE of the network output with respect to the real function for different STM sizes and number of generations per interaction. The three graphs that make up the picture correspond to the three functions. The MSE is calculated by running the resulting networks over the whole signal and comparing the result to the original one. It is clear that, for STMs larger than 8, the MSE becomes almost a constant, independently of the number of generations per update. This implies that, for a randomly filled STM on average, eight points of the signal provide enough information to model the whole signal. Obviously, if we take, for example, a STM of 10, we will be on the safe side, no matter how many generations of the evolutionary process we run. This is not necessarily the optimal solution, just the safest one. If we look at this graph more closely, we see that for a STM of 4 and running 2 generations per iteration we obtain almost the same error, but the computing required is now much less. Let us take a look at this graph in another way (figure 3). We can see that there are two types of behaviours with the number of generations per interaction with the world. In the first case, for STMs smaller than the critical size, if we carry out few generations per interaction, that is, if we evolve very little with a given frame of the STM, the results improve with the number of generations up to a point, in this case about 4 or 6 generations per interaction. More evolution without changing data items in the STM leads to a problem of overfitting the current STM frame and not generalizing to the whole fitness function. In short, for these values of the STM, we have a concave function with a minimum for a given number of generations per iteration. After the

critical STM value is surpassed, there is no minimum; the function is monotonously decreasing with the number of generations, implying that it is very difficult to overfit to the current STM, as it provides a good enough representation of the signal to model it correctly without having to use many frames.

Figure 4 displays the variation of the MSE during evolution for 6 cases. We have first considered a STM of 2 and have evolved the networks for 2 generations per interaction. There is a certain improvement, but, as 2 points is a very poor representation of the signal, evolution on 2 points just leads to oscillations in MSE, with very poor values. The networks never really learn the fitness function; they tend to evolve towards the contents of the STM in a given instant of time. This is more marked if we take 100 generations per interaction for a Short Term Memory of 2; now the MSE is basically pure noise. The network evolves to predict the STM contents very well (2 points) but not the signal. Consequently, the MSE with respect to the signal becomes really deficient. As STM increases in size, as in the case of the second set of graphs for STM 8 (middle graphs), we see that, now, the information in the STM starts to be relevant. There is an evolution towards a reasonable representation of the signal (low MSE). With new interactions with the world, the STM changes and some oscillations in the behaviour of the MSE can be observed, but, especially in the case where not many generations of the evolutionary algorithm are run between interactions, the networks are not overfitted to the instantaneous contents of the STM, but tend to obtain a more general solution. Obviously, the more generations per interaction we run the evolution, the more marked the overfitting of the instantaneous STM becomes. Finally, in the third case, we present a STM of 20 (top graphs), which is well above the critical point for this function. Now the networks evolve to a good prediction of the whole signal (low MSE)

with very little difference between running many or few generations of the evolutionary process between interactions with the world. The only appreciable difference is that with more generations, the process is evidently faster.

This is all very well for the logistic function, but what happens with other types of functions? We have considered a simpler and a more difficult function (*sin()* and 3D). If we produce the same representations for these functions as we did for the logistic one (figures 2 and 3, top and bottom in both cases), it can be clearly seen that the behaviour is basically the same. On one hand, the *sin()* function provides representations that are almost indistinguishable from those of the logistic function except, maybe, for the fact that they are smoother. This is a surprising result, indicating that for the network we are using, a *sin()* function is about as hard, or easy to model as a logistic function. If we look at the state space this is probably true. Thus, it is good to see that, for similar modelling difficulty, we basically obtain the same results. On the other hand, if we go to a much harder function to model, such as the 3D one we indicated above, we see that, although the general behaviour is similar, now the critical point has changed; it has gone up to a STM of about 20. What is really interesting about this is that, by looking at the slope of the MSE function as a function of the number of generations per interaction, one knows what side of the critical STM point one is for a given function. As commented above for the logistic function, if one is below the critical point, a small number of generations per interaction will provide better results in the global error. If one is above the critical point, two considerations may be made. On one hand, a smaller STM could probably be used, thus reducing computational load, on the other, the number of generations per interaction will not really affect the outcome. Consequently, by delimiting where the critical point is located, one could think of tweaking with

parameters such as the size of the STM, the evolutionary pressure and the number of generations per interaction with the world in order to optimize the computational load and the probability of obtaining a successful evolution. We must take into account that, if these parameters are not properly set, two consequences may arise. If the STM is too small or the evolutionary pressure per interaction too high (through many generations per interaction or an evolutionary algorithm with high pressure) the network will never learn the global function and will overfit the current STM frame, thus producing a global oscillating MSE. Besides, large STMs will lead to very long processing times.

4. Critical Points

In the previous sections, we have seen that, when we study the evolution of the Mean Squared Error of the signal model provided by an evolved ANN when the fitness function during evolution was sampled, a critical point arises for Short Term Memory (STM) size whereby the behaviour over this critical point is quite different from that below it. Below, the MSE is very dependent on Short Term Memory (STM) size and evolutionary pressure. Above it, this dependence disappears. In fact, above it, it seems that no matter how many generations we evolve on a given frame of the STM, the results remain the same; there is no overfitting. Consequently, if the STM is above the critical point, why not just evolve for a very large number of generations on a static STM without introducing any new data? There are two answers to this. First, this strategy would provide no capability of dealing with changing environments. Secondly, the critical point we are considering is what we would call a Dynamic sampling critical point, this is, the contents of the STM are enough to model the signal if they change at a reasonable rate.

In fact, one could define three different types of critical STM sizes. On one hand, what we would call static consecutive sampling critical point (SCS-CP). This critical size indicates what the minimum size of the STM must be in order to provide a good enough representation of the signal if the STM does not input any new data and the data it contains are consecutive samples of the signal. This point obviously depends on how good a representation of the signal we desire and on the sampling frequency. For a 63 sample per cycle representation of the *sin()* function, in our case it results that a STM of 63 is required as SCS-CP.

The second type of critical point is given by the average STM required to reliably represent the signal when this STM is static (no updating) and the points in it are random samples of the signal. This is what we call the static random sampling critical point (SRS-CP). For the *sin()* function, the SRS-CP would be around 20.

Finally, we have the critical point we have derived in Part I, that is, the Dynamic sampling critical point (DS-CP). In this case, the STM will be updated with some criterion. As in the case of static critical points, it will also depend on how we sample the signals or environments. Obviously, a DCS-CP (dynamic consecutive sampling critical point) will in general present a higher value than a DRS-CP (dynamic random sampling critical point) given the better probability of a random sampling of providing a good representation of the signal. In fact, for the *sin()* function, the DRS-CP turns out to be about 10 for the same sampling frequency as above.

From a practical point of view, all of the above comments lead to the consideration that the minimum size necessary for the STM to be able to provide a good enough representation of the signal, in order to obtain a model within a given error margin, depends on two factors: sampling frequency and dynamicity. Sampling frequency, if

above the frequency established by Shannon's theorem, is only relevant if the data are taken in a consecutive manner. Any other STM updating strategy will reduce the influence of this factor.

Regarding dynamicity, any evolutionary strategy acting on a dynamic STM, that is, a STM that is updated with new information as new interactions with the function or environment occur, will see a virtual STM that is larger than the real one, just as long as the evolutionary pressure on a single frame of the STM is not too large. Thus, for practical use, one would think that the optimal usage of a STM would occur when the maximum information on the signal is introduced in it through an appropriate update or replacement strategy, and when this information changes with time at a rate compatible with the number of generations and evolutionary pressure of the algorithm in between interactions with the function or environment. When evolving between interactions or updates of a dynamically changing STM, we don't want the population to converge to the current frame of the STM, otherwise, when the next frame comes in, unless the STM is well above the static critical point, the performance of the solution evolved will be very poor. The result of something like this would be to have an algorithm that is constantly converging to the contents of the STMs, but never really modelling the underlying function or environment.

5. Replacement mechanism

One of the most relevant aspects in managing dynamic STMs, as mentioned above, is the replacement mechanism; the better the replacement strategy, the better the representative power of each frame. Thus, a good replacement strategy maximizes the information content of the memory.

To achieve this objective, we have studied the effect of introducing a strategy that maximizes the minimum distance from every item in the STM to every other item. This approach implies defining a distance between items. In our case, we have just used a standard Euclidean distance taking each item in the STM as an n-dimensional vector.

This approach usually results in a very good distribution of the items in the n-dimensional space conformed by the representations of the items in the Short Term Memory. Evidently, this strategy does not take into account the importance of the items towards the task in hand; it is just a maximization of the coverage of the n-dimensional space by a limited number of vectors. It is just a geometric property. Notwithstanding this fact, this type of approach has provided a good selection of the points to be used for the evolution of representations as shown in figure 8.

The problem with this approach is that, after a maximum-minimum distance configuration of the STM has been achieved, no changes occur in the STM, thus obviating the dynamicity principle. We are basically obtaining the best static STM possible, but we are missing the possibility of taking advantage of the dynamic properties of the STM in terms of reducing computational load.

To achieve a compromise between the maximum minimum distance criterion for the STM and the dynamicity principle, we have introduced some stochasticity in the replacement algorithm. This level of randomness can be regulated through a parameter we will call stochastic control of the STM. Basically, a new item has a certain probability of being introduced in the STM and, if it is introduced, it will substitute another item which is chosen through a tournament type selection. The strength of the contenders in the tournament is given by the inverse of their minimum distance to the other elements in the STM. Depending on how large we make the tournament window,

we can go from an worst leaves first like STM by setting the tournament window to *STMsize-1* to an almost random replacement strategy by setting the tournament window to 2.

Figures 5 and 6 display the model of the *sin()* function obtained by the best network resulting from evolution under different replacement strategies. In every case, we used a Short Term Memory of size 4 and 8 generations of evolution between updates. For the three models in figure 5, the samples of the function were taken consecutively. For the three of figure 6, the samples were taken randomly. If we are working with real problems, both cases may arise. When an agent is exploring an environment, what it will perceive will be relatively random depending on its actions. On the other hand, when a signal is being analyzed, it is usually received in a consecutive fashion.

The top graphs in both figures correspond to a static STM, that is, once it is filled there is no replacement. As we see, in the case of the sequential sampling strategy, the network learns to model the first points of the *sin()* function perfectly, as they are the only ones in the STM. Obviously, the model with respect to the whole signal is very poor. When a randomly filled static STM is used, the model improves, but, as it only considers four points of the signal, it is quite sensitive to the randomness of the points and usually overfits them without really modelling the signal. This is shown in the top graph of figure 6. In the middle graphs of both figures, the STM works as a FIFO. The oldest element is substituted by the new input without considering if the information provided by the new element is useful. It could be thought that this would be an appropriate strategy in autonomous robotics, where older information is usually considered less relevant. The results are not much better than in the previous case. The network tends to overfit the local contents of the STMs and, as these change

continuously, it does not obtain a general representation of the signal.

Finally, in the bottom two graphs we display the case of the replacement strategy we introduced above. Now, the models obtained are much better, even in the case of the sequential sampling of the signal. This is mainly due to the fact that, now, a new data item has a low probability of being input to the STM if it provides little information (if it is not different from those present in the STM), but this probability is not zero, so the STM does not become static after a while.

The same results can be obtained for other functions. In figure 7 we display the evolution of fitness of the best network for an *xsin(x)+ysin(y)* function. We have used a STM size of 40. The top figure displays the case of using a FIFO-like replacement strategy. For this case, the overfit of the current contents of the STM leads to very large oscillations in the fitness with respect to the global function.

If we consider a maximum-minimum replacement strategy, as shown in the bottom part of the figure, the fitness of the model in terms of the MSE improves until it reaches a relatively constant value. If the network is tested the model it provides is quite good. As an illustration of the effects of this strategy, in figure 8 we present the final contents of the Short Term Memory for this last function. The homogeneity of the distribution of the data points within the [-10,10] interval, both on the x and on the y axis (inputs), is clear. We have also represented the z value (output) as bars and it can be seen that, when points are close in the x-y plane, their z values are quite different. The replacement strategy obviously maximizes the descriptive power of the STM. The problem here is that even though the descriptive power of the STM is maximized, if it is static it may still not be enough to obtain an appropriate representation of the signal. This can be appreciated in figure 9. In the top left part of the figure we display a

representation of the function. In the bottom left part we have plotted the contents of the 40 element STM after 3000 interactions with the function using the replacement strategy indicated above. It is clear that even though the 40 elements may be arranged as to maximize information, they are clearly not enough to represent the function. If we evolve a network on these elements as a static representation, we will obtain the result presented in the bottom right part of figure 9. That is, we obtain an almost perfect representation of the STM, but not of the function. On the other hand, as commented previously, if we introduce stochasticity in the replacement strategy with a dynamic STM the result obtained is the one presented in the top right part of figure 9. Now the system, never stops being updated, and, using the same 40 element STM, the process contemplates a larger virtual STM which allows it to achieve a much better representation of the function we are seeking.

6. Conclusions

In the context of the evolutionary processes that take place when considering the Multilevel Darwinist Brain, we have taken a look at the relevant parameters in evolutionary processes that must use sampled fitness functions. The size of the Short Term Memory that stores the samples of the fitness function. presents a critical point for each problem above which the error obtained is basically constant independently of the number of generations evolved in between interactions with the world. On the other hand, the number of generations of evolution between interactions also presents a critical point over which the evolutionary process tends to oscillate due to the fact that the population becomes specific to the snapshot of the world provided by the STM in a given instant of time. By adequately choosing these two parameters we obtain the

optimal results with the minimal computation benefiting from the inertias in the evolutionary process. These permit evolving models using STMs that in a given instant do not represent the fitness functions very well but which become appropriate by means of their dynamic behavior through the interaction with the world.

To maximize the performance of these processes we have made use of replacement strategies for the elements included in the STM that permit maximum information snapshots, thus maximizing the efficiency of the evolution.

Consequently, by appropriately choosing STM size, evolutionary pressure of the algorithm and STM replacement strategies in dynamic STMs, the quality of the results improve drastically, leading to processes that require much less computational resources for the same result quality. In addition, the oscillation in the fitness resulting from the sampling of the fitness function can be reduced and, thus, the convergence to an adequate solution is faster.

Acknowledgements

This work was supported by the MCYT of Spain through project TIC2000-0739C0404.

References

[1] F. Prieto, R. J. Duro, J. Santos, Modelo de Mecanismo Mental Darwinista para robots móviles autónomos, *Proceedings of SAAEI'96*. Zaragoza, Spain, 1996.

[2] R. J. Duro, J. Santos, F. Bellas, A. Lamas, On Line Darwinist Cognitive Mechanism for an Artificial Organism, *Proceeding Supplement Book SAB2000*. pp 215-224.

[3] F. Bellas, J.A. Becerra, R. J. Duro, Using evolution for thinking and deciding, *Advances in Fuzzy Systems and Evolutionary Computation. pp 242-247.*

[4] Santos, J., Duro, R.J., Becerra, J.A., Crespo, J.L., and Bellas F. (2001), "Considerations in the Application of Evolution to the Generation of Robot Controllers", *Information Sciences* 133:127-148

[5] Beer, R.D. and Gallagher, J.C. (1992), "Evolving Dynamical Neural Networks for Adaptive Behavior", *Adaptive Behavior*, Vol. 1, No. 1, pp. 91-122.

[6] Cheng, M.Y. and Lin, C.S. (1997), "Genetic Algorithm for Control Design of Biped Locomotion", *Journal of Robotic Systems*, Vol. 14, No. 5, pp. 365-373.

[7] Dain, R. A. (1998), "Developing Mobile Robot Wall-Following Algorithms Using Genetic Programming", *Applied Intelligence*, Vol. 8, pp. 33-41.

[8] Floreano, D. and Mondada, F. (1998), "Evolutionary Neurocontrollers for Autonomous Mobile Robots", *Neural Networks*, Vol. 11, pp. 1461-1478.

[9] Gomez, F. and Miikkulainen, R. (1997), "Incremental Evolution of Complex General Behavior", *Adaptive Behavior*, Vol. 5, No. 3/4, pp. 317-342.

[10] Jakobi, N. (1997), "Evolutionary Robotics and the Radical Envelope of Noise Hypothesis", *Adaptive Behavior*, Vol. 6, No. 2, pp. 325-368.

[11] Nolfi, S. and Parisi, D. (1997), "Learning to Adapt to Changing Environments in Evolving Neural Networks", *Adaptive Behavior*, Vol. 5, No. 1, pp. 75-98.

Figure Captions:

Figure 1: Block Diagram of the cognitive mechanism where the shadowed area represents the external operation of the mechanism and the remaining area represents the internal operation.

Figure 2: MSE of the network output after evolution with respect to the real function for different number of generations per interaction and STM sizes.

Figure 3: MSE of the network output after evolution with respect to the real function for different STM sizes and number of generations per interaction.

Figure 4: Evolution of the MSE for STMs of sizes 2, 8 and 20 using the logistic function. The top figure shows 2 generation per update of the STM and the bottom figure shows 100 generations per update.

Figure 5: Model of the *sin()* function provided by de best network evolved using different replacement strategies for the STM. It is a size 4 STM with 8 generations of evolution per update. The strategies considered are: static (top), FIFO (middle) and maximum-minimum replacement (bottom). The functions were sampled sequentially.

Figure 6: Model of the *sin()* function provided by the best network evolved using different replacement strategies for the STM. It is a size 4 STM with 8 generations of evolution per update. The strategies considered are: static (top), FIFO (middle) and maximum-minimum replacement (bottom). The functions were sampled randomly.

Figure 7: Evolution of the MSE of the best individual for the $f(x,y)=x\sin(x)+y\sin(y)$ function using a FIFO (top) and a maximum minimum replacement (bottom) over a STM size of 40.

Figure 8: Distribution of contents of the STM.

Figure 9: Top left: F(x,y)=xsin(x)+ysin(y). Top right: Prediction obtained by a network using a 40 element dynamic STM with a max-min replacement strategy including stochasticity. Bottom Left: representation provided by the STM after 3000 interactions. Bottom right: Prediction by the same network trained on the static STM on the left.

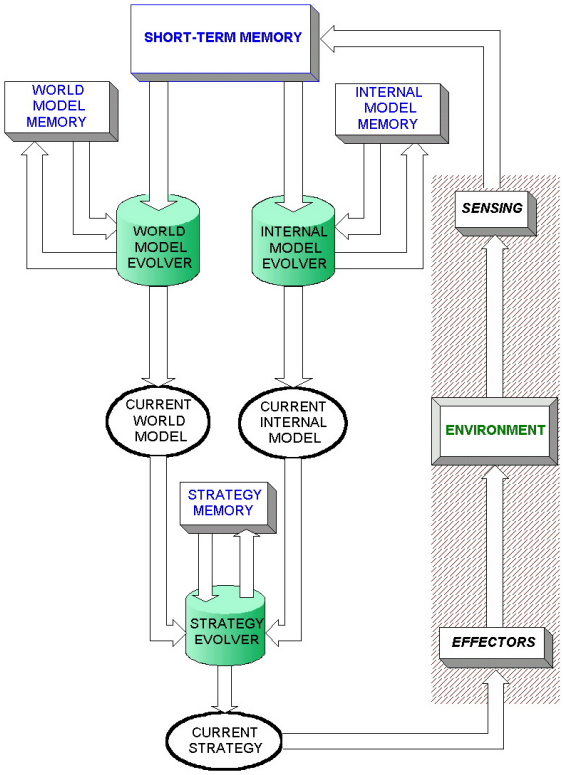
SHORT-TERM MEMORY
WORLD MODEL MEMORY
INTERNAL MODEL MEMORY
SENSING
WORLD MODEL EVOLVER
INTERNAL MODEL EVOLVER
CURRENT WORLD MODEL
CURRENT INTERNAL MODEL
ENVIRONMENT
STRATEGY MEMORY
STRATEGY EVOLVER
EFFECTORS
CURRENT STRATEGY

Figure 1

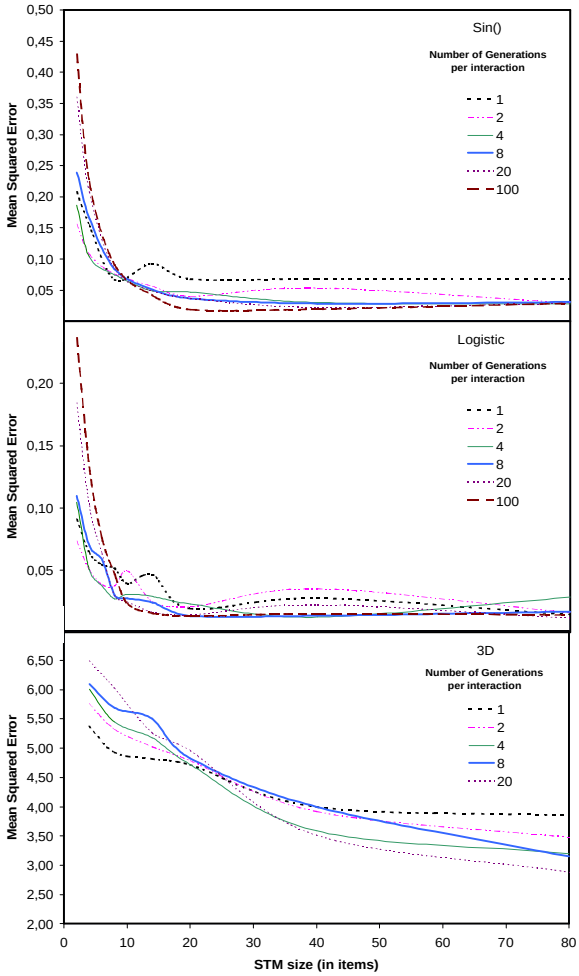
Sin()
Number of Generations per interaction
1
2
4
8
20
100
Mean Squared Error
0,50
0,45
0,40
0,35
0,30
0,25
0,20
0,15
0,10
0,05
Logistic
Number of Generations per interaction
1
2
4
8
20
100
Mean Squared Error
0,20
0,15
0,10
0,05
3D
Number of Generations per interaction
1
2
4
8
20
Mean Squared Error
6,50
6,00
5,50
5,00
4,50
4,00
3,50
3,00
2,50
2,00
0
10
20
30
40
50
60
70
80
STM size (in items)

Figure 2

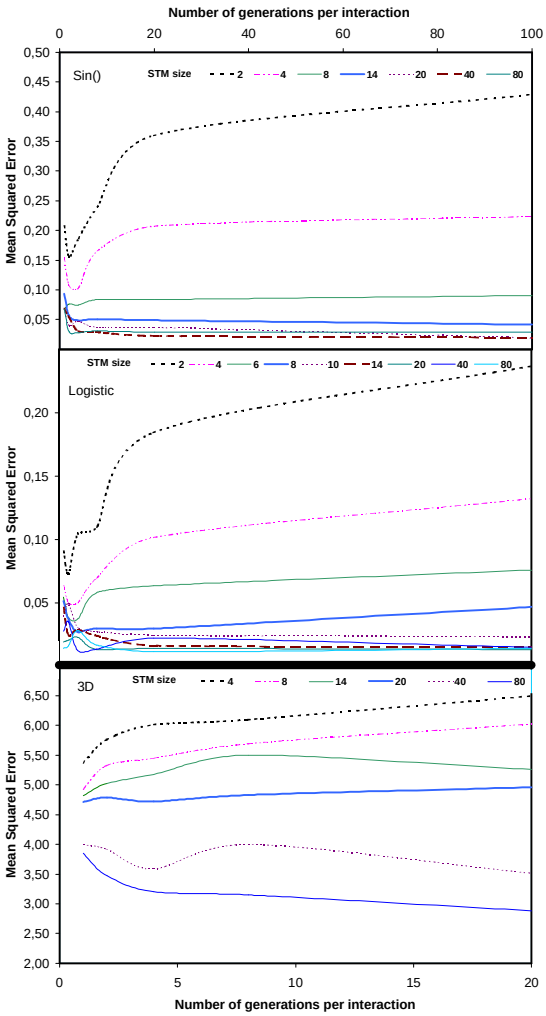
Number of generations per interaction
Sin()
STM size
2
4
8
14
20
40
80
Mean Squared Error
Logistic
6
10
3D
Number of generations per interaction

Figure 3

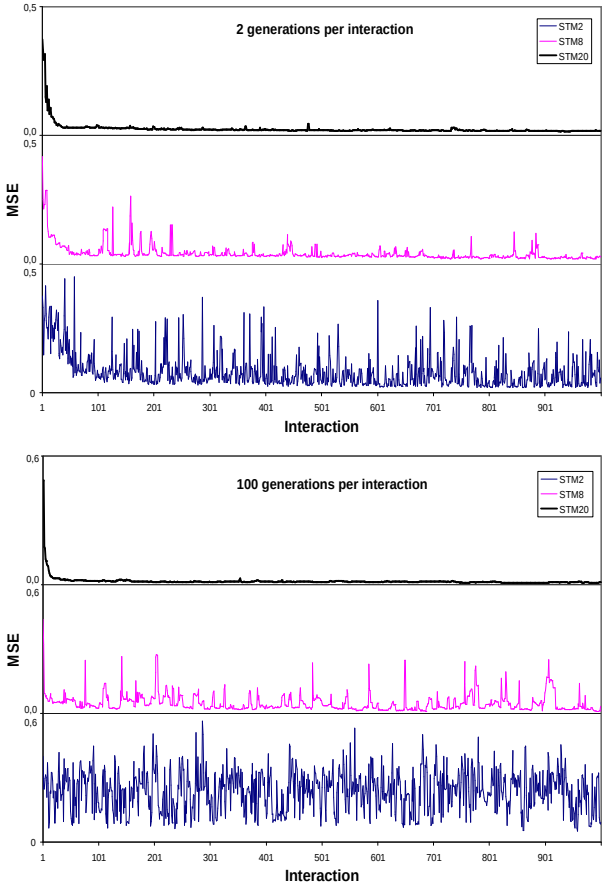
2 generations per interaction
STM2
STM8
STM20
MSE
0,5
0,0
0,5
0,0
0,5
0
1
101
201
301
401
501
601
701
801
901
Interaction
100 generations per interaction
STM2
STM8
STM20
MSE
0,6
0,0
0,6
0,0
0,6
0
1
101
201
301
401
501
601
701
801
901
Interaction

Figure 4

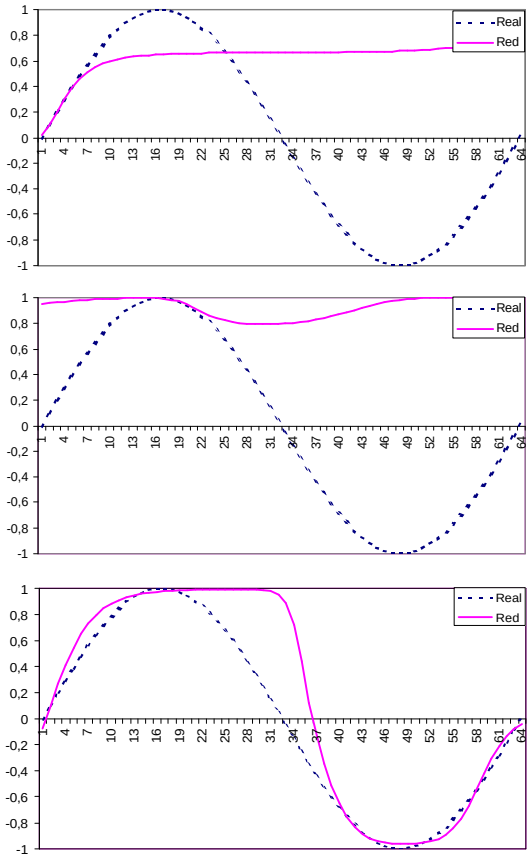
Real
Red
1
0,8
0,6
0,4
0,2
0
-0,2
-0,4
-0,6
-0,8
-1
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64

Figure 5

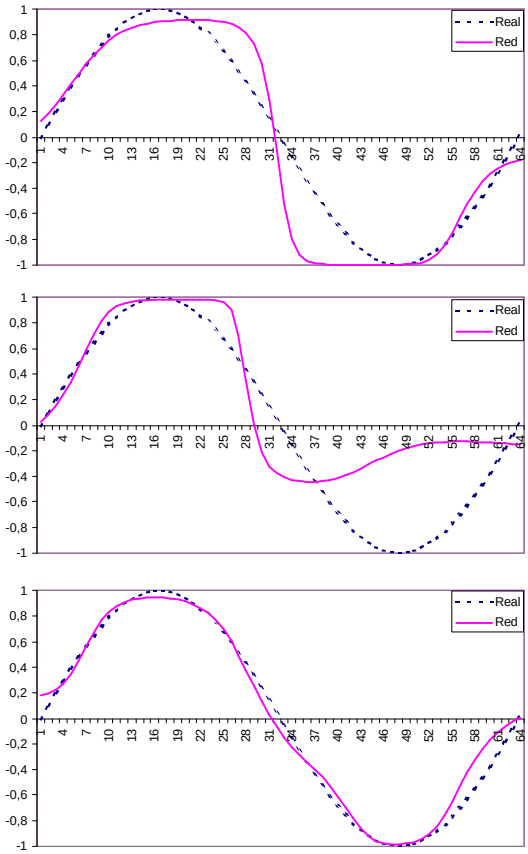
Real
Red
1
0,8
0,6
0,4
0,2
0
-0,2
-0,4
-0,6
-0,8
-1
1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64

Figure 6

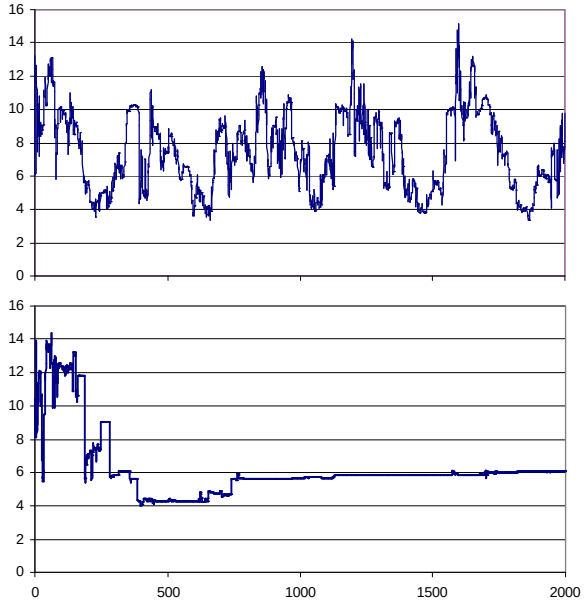

Figure 7

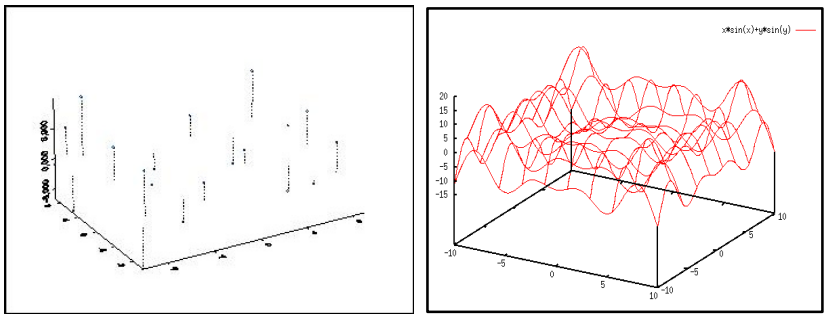

Figure 8

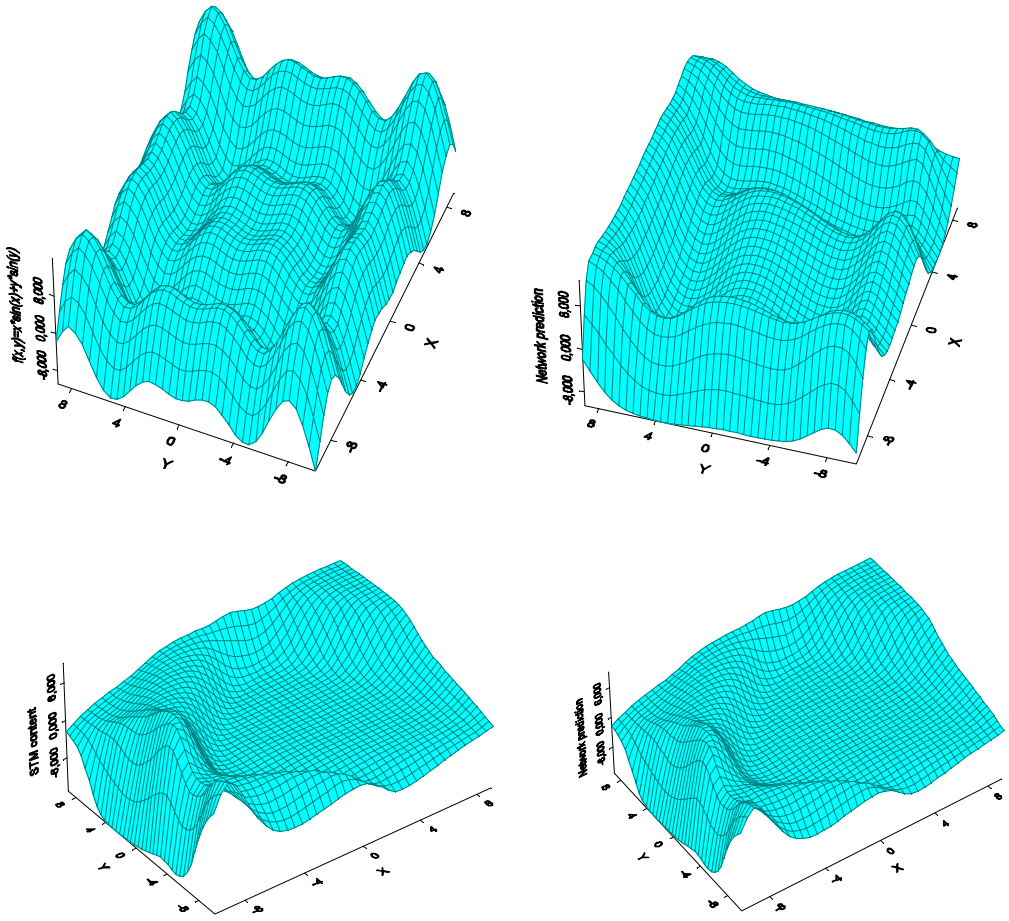
f(x,y)=x*tanh(x)+y*sin(y)
-8,000 0,000 8,000
Network prediction
-8,000 0,000 8,000
STM content
-8,000 0,000 8,000
Network prediction
-8,000 0,000 8,000
X
Y

Figure 9