A new artificial life formalization model: A worm with a Bayesian Brain*

F. Aznar, M. Pujol, R. Rizo, and P. Suau

Department of Computer Science and Artificial Intelligence
University of Alicante
{fidel,mar,rizo,pablo}@dccia.ua.es

Key words: Bayesian Programming, Artificial Life, Life Formalization Model

Abstract. This paper shows an application of Bayesian Programming to model a simple artificial life problem: that of a worm trying to live in a world full of poison. Any model of a real phenomenon is incomplete because there will always exist unknown, hidden variables that influence the phenomenon. To solve this problem we apply a new formalism, Bayesian programming, which has previously been used in autonomous robot programming. The proposed worm model has been used to train a population of worms using genetic algorithms. We will see the advantages of our method compared with a classical approach. Finally, we discuss the emergent behaviour patterns we observed in some of the worms and conclude by explaining the advantages of the applied method.

1 Introduction

Articial Life is a relatively recent discipline whose primary goal was to study the recreation of biological phenomena using artificial methods.

Nevertheless, applications in this field have quickly exceeded purely biological applications: the methods used can be useful in the study, simulation and behaviour prediction of a wide set of complex systems, not only biological.

The immediate applications of Artificial Life are in the simulation of complex processes, chemical synthesis, multivariate phenomena, etc.

Very complex global behaviour patterns can be observed, initiated by simple local behaviour. It is this characteristic (sometimes called emergent behaviour) which makes Artificial Life particularly appropriate for the study and simulation of complex systems for which detailed analysis, using traditional methods, is practically non-viable.

Nevertheless, it is necessary to bear in mind that any model of a real phenomenon will always be incomplete due to the permanent existence of unknown, hidden variables that will influence the phenomenon. The effect of these variables is malicious since they will cause the model and the phenomenon to have

* This work has been financed by Spanish Comisión Interministerial de Ciencia y Tecnología (CICYT) project number TIC2001-0245-C02-02 and by the Generalitat Valenciana project GV04B685

different behavioural patterns. In this way both artificial systems and natural systems have to solve a common problem: how each individual within the system uses an incomplete model of the environment to perceive, infer, decide and act in an efficient way.

Reasoning with incomplete information continues to be a challenge for artificial systems. Probabilistic inference and learning try to solve this problem using a formal base. A new formalism, the Bayesian programming (BP) [1], based on the principle of the Bayesian theory of probability, has been successfully used in autonomous robot programming. Bayesian programming is proposed as a solution when dealing with problems relating to uncertainty or incompleteness.

Certain parallelisms exist between this kind of programming and the structure of living organisms. As shown in a theoretical way in [2], we can suppose that if a natural process correspond to all the steps in a Bayesian program, then live organisms also use Bayesian inference and learning. In this way, natural evolution provided living beings with both the pertinent variables, and the adequate decomposition and parametric forms. The pertinent variables may have been obtained by selecting the sensors and actuators in order to supply vital information. The decomposition would correspond to the structure of the nervous system, which basically expresses dependencies and conditional independencies between variables. The parametric forms can be seen as the information processing units, implemented by neurons and assemblies of neurons. Given this apparatus, corresponding to preliminary knowledge, each individual in his lifetime could answer the first question by experimenting and learning the values of the free parameters of his nervous system.

In this paper we will see a simple example of how to apply BP formalism to a specific artificial life problem. We will define a virtual world, divided into cells, some of which contain poison. In this world lives a worm with only one purpose, to grow indefinitely. In order to grow the worm must move through a certain number of non poisonous cells in its world. If the worm moves into a poisonous cell then it will die. The worm has a limited vision of the world, provided by its sensorial organs, found in its head. These sensors allow the worm to see no further than the adjacent cell.

We believe that this is one of the first approaches that uses Bayesian programming for the formalization of an artificial life problem as we haven't found any evidence of it's application in this field. The main area where BP has been and continues to be applied is robotics [2], [1], [3], [4], [5].

This paper is divided into five parts. The first is a short formal introduction to Bayesian Programming, the second describes the problem and it's description in terms of BP, the third shows the way we create the environment to evolve the worms, the fourth is about the experimentation and the emergent behaviours we witness in the worm population and finally, the fifth, draws conclusions and future lines of investigation to be followed.

2 Bayesian Programming

Before specifying our problem we will introduce the reader to the principles and basics of Bayesian programming showing the basic concepts, postulates, definitions, notations and rules that are necessary to define a Bayesian Program.

2.1 Basic concepts

Definition and notation

Proposition We will use logical propositions denotated by lowercase names. Propositions may be composed to obtain new propositions using the usual logical operators: $(\wedge, \vee, \neg)$ denoting the conjunction, disjunction and the negation respectively.

Discrete variable Discrete variables will be denoted by names starting with one uppercase letter. By definition, a discrete variable X is a set of logical propositions x_i such that these propositions are mutually exclusive ($\forall_{i,j}, i \neq j, x_i \wedge y_i = false$) and mutually exhaustive (at least one of the propositions x_i is true). x_i stands for X takes its i^{th} value. $\lfloor X \rfloor$ denotes the cardinal of the set X (the number of the propositions x_i).

The conjunction of two variables X and Y, denoted by $X \otimes Y$, is defined as a set of $\lfloor X \rfloor \otimes \lfloor Y \rfloor$ propositions $x_i \wedge y_i$. $X \otimes Y$ is a set of mutually exclusive and exhaustive logical propositions (consequently it is a new variable). Of course, the conjunction of n variables is also a variable. The disjunction of two variables, defined as the set of propositions $x_i \vee y_i$ is not a variable because these propositions are not mutually exclusive.

Probability To be able to deal with uncertainty, we will attach probabilities to propositions. We consider that, to assign a probability to a proposition a, it is necessary to have at least some preliminary knowledge, summed up by a proposition π. Consequently, the probability of a proposition a is always conditioned, at least, by π. For each different $\pi, P(\cdot|\pi)$ is an application assigning a unique real value $P(a|\pi)$ in the interval $[0, 1]$,to each proposition a.

Of course, we will be interested in reasoning on the probabilities of the conjunctions, disjunctions and negations of propositions, denoted, respectively, by $P(a \wedge b|\pi), P(a \vee b|\pi), P(\neg a|\pi)$. We will also be interested in the probability of a proposition a conditioned by previous knowledge π and some other proposition b. This will denotated $P(a|b \wedge \pi)$.

For simplicity and clarity, we will also use probabilistic formula with variables appearing instead of propositions each time a variable X appears in a probabilistic formula $\Phi(X)$ it should be understood as $\forall x_i \in X, \Phi(x_i)$.

Inference postulates and rules This section presents the inference postulates and rules used to carry out probabilistic reasoning

Conjunction and normalization postulates for propositions In probabilistic reasoning only two basic rules are defined and used to derive the rest. These two rules, if we use discrete probabilities, are sufficient to solve any inference problem.

- Conjunction Rule. Gives the probability of a conjunction of propositions.

$$\begin{aligned} P(\,a \wedge b|\,\pi) &= P(a|\pi) \times P(b|a \wedge \pi) \\ &= P(b|\pi) \times P(a|b \wedge \pi) \end{aligned} \tag{1}$$

- Normalization Rule. States that the sum of the probabilities of a and $\neg a$ is one.

$$P(\,a|\,\pi) + P(\,\neg a|\,\pi) = 1 \tag{2}$$

Using the last two we derive the next rules for propositions and for variables:

- Disjunction rule form propositions.

$$P(a \vee b|\pi) = P(a|\pi) + P(b|\pi) - P(a \wedge b|\pi) \tag{3}$$

- Conjunction rule for variables.

$$\begin{aligned} P(\,X \otimes Y|\,\pi) &= P(X|\pi) \times P(Y|X \wedge \pi) \\ &= P(Y|\pi) \times P(X|Y \wedge \pi) \end{aligned} \tag{4}$$

- Normalization rule for variables.

$$\sum_X P(X|\pi) = 1 \tag{5}$$

- Marginalization rule for variables.

$$\sum_X P(X \otimes Y|\pi) = P(Y|\pi) \tag{6}$$

2.2 Bayesian program definition

A Bayesian program is defined as a means of specifying a family of probability distribution. The constituent elements of a BP are presented in figure 1.

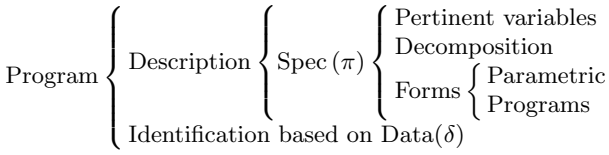

Fig. 1. Structure of a Bayesian program

Description The purpose of a description is to specify an effective method of computing a joint distribution on a set of variables $\left\{X^1, X^2, ..., X^n\right\}$ given a set of experimental data δ and preliminary knowledge π. This joint distribution is denoted as $P(X^1 \otimes X^2 \otimes ... \otimes X^n | \delta \otimes \pi)$

Preliminary Knowledge To specify preliminary knowledge the programmer must undertake the following:

- Define the set of relevant variables $\left\{X^1, X^2, ..., X^n\right\}$ on which the joint distribution is defined.
- Decompose the joint distribution. Given a partition of $\left\{X^1, X^2, ..., X^n\right\}$ into k subsets we define k variables $L^1, ..., L^k$ each corresponding to one of these subsets.
  Each variable L^i is obtained as the conjunction of the variables $\left\{X^{i_1}, X^{i_2}, ...\right\}$ belonging to the subset i. This way the conjunction rules leads to:
  $$\begin{aligned} P(X^1 \otimes X^2 \otimes ... \otimes X^n | \delta \otimes \pi) = P(L^1 | \delta \otimes \pi) \times P(L^2 | L^1 \otimes \delta \otimes \pi) \times ... \times \\ \times P(L^k | L^{k-1} \otimes L^{k-2} \otimes ... \otimes L^1 \otimes \delta \otimes \pi) \end{aligned} \tag{7}$$
  Conditional independence hypotheses then allow further simplifications. A conditional independence hypothesis for variable L^i is defined by picking some variables X^i among the variables appearing in conjunction $L^{i-1} \otimes L^{i-2} \otimes ... \otimes L^1$, calling R^i the conjunction of these chosen variables and setting:
  $$P(L^i | L^{i-1} \otimes L^{i-2} \otimes ... \otimes L^1 \otimes \delta \otimes \pi) = P(L^i | R^i \otimes \delta \otimes \pi) \tag{8}$$
  We obtain:
  $$\begin{aligned} P(X^1 \otimes X^2 \otimes ... \otimes X^n | \delta \otimes \pi) = P(L^1 | R^1 \otimes \delta \otimes \pi) \times \\ \times P(L^2 | R^2 \otimes \delta \otimes \pi) \times ... \times P(L^k | R^k \otimes \delta \otimes \pi) \end{aligned} \tag{9}$$
  Such a simplification of the joint distribution as a product of simpler distributions is called a decomposition.
- Define the form. Each distribution $P(L^i | R^i \otimes \delta \otimes \pi)$ appearing in the product (9) is then associated with either a parametric form (i.e., a function $f_\mu(L^i)$) or another Bayesian program. In general μ is a vector of parameters that may depend on R^i or δ or both. Learning takes place when some of these parameters are computed using the data set δ.

Questions Given a description (i.e., $P(X^1 \otimes X^2 \otimes ... \otimes X^n | \delta \otimes \pi)$), a question is obtained by partitioning $\left\{X^1, X^2, ..., X^n\right\}$ into three sets: the searched variables, the known variables and the unknown variables.

The variables $Searched$, $Known$ and $Unknow$ are defined as the conjunction of the variables belonging to these sets. We define a question as the distribution:

$$P(Searched | Known \otimes \delta \otimes \pi) \tag{10}$$

2.3 Running a Bayesian program

Running a Bayesian program supposes two basic capabilities: Bayesian inference and decision-making.

Bayesian Inference Given the joint distribution $P(X^1 \otimes X^2 \otimes ... \otimes X^n | \delta \otimes \pi))$, it is always possible to compute any possible question, using the following general inference:

$$\begin{aligned} P(Searched|Known \otimes \delta \otimes \pi) &= \sum_{Unknown} P\left(Searched \otimes Unknown | Known \otimes \delta \otimes \pi\right) \\ &= \frac{\sum_{Unknown} P(Searched \otimes Unknown \otimes Known | \delta \otimes \pi)}{P(Known|\delta \otimes \pi)} \\ &= \frac{\sum_{Unknown} P(Searched \otimes Unknown \otimes Known | \delta \otimes \pi)}{\sum_{Unknown} \sum_{Searched} P(Searched \otimes Unknown \otimes Known | \delta \otimes \pi)} \\ &= \frac{1}{\sum} \times \sum_{Unknown} P\left(Searched \otimes Unknown \otimes Known | \delta \otimes \pi\right) \end{aligned} \tag{11}$$

In the third equation (11) the denominator appears to be a normalization term. Consequently, by convention it is replaced by $\frac{1}{\sum}$.

The general Bayesian inference is a very difficult problem. The problem of exact inference has been proved to be NP-hard and the general problem of approximate inference too. In this paper we assume all inference problems to be solved and implemented using an efficient inference machine.

Decision-making For a given distribution, different decision policies are possible. We can search for the best (highest probability) values or we can draw at random according to the distribution. We will use the second policy calling it $Draw(P(Searched|Known \otimes \delta \otimes \pi))$.

3 Specifying the problem using Bayesian Programming

We commented above on the existence of a world, composed of $n \times m$ cells, where each cell C_{ij} could be in any one of four different states: empty, containing poison, part of the wall which surrounds this artificial world or it could be hidden from view beneath the worm's tail. In this way a cell $C_{ij} = \{\emptyset, V, M, L\}$. The wall configuration is uniform for each generated world, however, in contrast, the distribution of poison is random and varies from world to world. Initially, we asume the amount of poisonous cells to be between 5%-10% of the total.

Within each world lives only a single worm which only objective is to move and to grow. A worm grows and increases its length by one unit every time it moves through d cells inside its world. If the worm moves to a cell that is not

empty then it will die. The only information about the world available to the worm is provided by its sensors, located in its head (see figure 2). A sensor is only able to see the state of cells adjacent to and in front of the worm's head, no further.

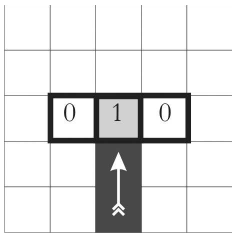

Fig. 2. Worm's vision relating to its head and the direction it has in the world.

We assume that each worm has a certain knowledge represented as states. In this way each worm can stay in one state E_t given a reading and a previous state. Furthermore, a worm could obtain a reading of the world L_t represented as a binary triplet which specifies if the cell in the position of its components is occupied '1' or not '0'. Finally, a worm could execute three actions. Go straight ahead, turn left or turn right $A_t = \{u, l, r\}$ the actions will be guided only by a reading and the actual state of the worm. Once the action A_t has been executed the worm can change to a new state E_{t+1}

3.1 Variables description

The first part of a Bayesian program is to define the pertinent variables of the problem.

To develop a movement in the world, the worm only needs to know the reading L_t of it's sensor and the actual state E_t, in addition to the set of actions A it could develop in the world. As we commented previously, an action A_t must be followed by an instant change in state $t + 1$.

In this way we define the following variables for each instant t:

$$\begin{aligned} L_t &= \{000, 001, 010, ..., 111\}, \lfloor L_t \rfloor = 8 \\ E_t &= \{0, 1, 2, ..., k\}, \lfloor E_t \rfloor = k + 1 \\ A_t &= \{u, l, r\}, \lfloor A \rfloor = 3 \end{aligned} \tag{12}$$

3.2 Decomposition

We define a decomposition of the joint probability distribution $P(L_t \otimes E_{t-1} \otimes E_t \otimes A | \pi_W)$ as a product of simpler terms. This distribution is conditioned by the previous knowledge π_w we are defining.

$$
\begin{aligned}
P\left(L_t \otimes E_{t-1} \otimes E_t \otimes A_t | \, \pi_W\right) = P(L_t|\pi_W) \times P(E_{t-1}|L_t \otimes \pi_W) \times \\
\times P(E_t|E_{t-1} \otimes L_t \otimes \pi_W) \times P(A_t|E_t \otimes E_{t-1} \otimes L_t \otimes \pi_W) = \\
= P(L_t|\pi_W) \times P(E_{t-1}|L_t \otimes \pi_W) \times \\
\times P(E_t|E_{t-1} \otimes L_t \otimes \pi_W) \times P(A_t|E_t \otimes L_t \otimes \pi_W)
\end{aligned}
\tag{13}
$$

The second equality is deduced from the fact that an action only depends on the actual state and the reading taken.

3.3 Parametrical forms

In order to be able to solve the joint distribution we need to assign parametrical forms to each term appearing in the decomposition:

$$
\begin{aligned}
P(L_t|\pi_W) &\equiv Uniform \\
P(E_{t-1}|L_t \otimes \pi_W) &\equiv Uniform \\
P(E_t|E_{t-1} \otimes L_t \otimes \pi_W) &\equiv G\left(\mu(E_{t-1}, L_t), \sigma(E_{t-1}, L_t)\right) \\
P(A_t|E_t \otimes L_t \otimes \pi_W) &\equiv G\left(\mu(E_t, L_t), \sigma(E_t, L_t)\right)
\end{aligned}
\tag{14}
$$

We assume that the probability of a reading is uniform because we have no prior information about the distribution of the world. In the same way we consider that all possible worm states can be reached with the same probability.

Give a state E_{t-1} and a lecture L_t we believe that only one state E_t would be preferred. In this way the distribution $P(E_t|E_{t-1} \otimes L_t \otimes \pi_W)$ is unimodal. However, depending on the situation, the decision to be made may be more or less certain. This behaviour is resumed by assigning a Gaussian parametrical form to $P(E_t|E_{t-1} \otimes L_t \otimes \pi_W)$.

In the same way, given a state and a reading we suppose that an action with more or less intensity would be prepared . We assign a Gaussian parametrical form to $P(A_t|E_t \otimes L_t \otimes \pi_W)$.

3.4 Identification

We show a set of free parameters which define the way the worm moves. These free parameters, derived from the parametrical form (means and standard deviations of all the Gaussians $\lfloor E_{t-1} \rfloor \times \lfloor L_t \rfloor$ and $\lfloor E_t \rfloor \times \lfloor L_t \rfloor$), would be the ones to be learned.

3.5 Utilization

The movement of a worm involves the following steps

- To obtain a reading L_t from the worm's sensors.
- To answer the question $Draw(P(A_t|E_t \otimes L_t \otimes \pi_W))$
- The worm will execute the movement command A
- To answer the question $Draw(P(E_{t+1}|E_t \otimes L_T \otimes \pi_W))$
- The worm will change to the state E_{t+1}

4 Genetic Algorithms

Genetic algorithms (GA) are a global search technique which mimic aspects of biological evolution, namely the process of natural selection and the principle of *survival of the fittest*. They use an adaptive search procedure based on a population of candidate solutions or *chromosomes*. Each iteration or *generation* involves a competitive selection procedure that favours fitter solutions and rejects poorer solutions. The successful candidates are then recombined with other solutions by swapping components with one another; they can also be mutated by making a small change to a single component. The procedure is repeated for many generations, producing new solutions that are biased towards regions of the search space in which good solutions have already been found.

We initially assume that the worm's parameters are generated randomly. The worm only has previous knowledge provided by its knowledge decomposition. The learning process would be produced generation after generation, where the longest living worms in the world would be those most enabled and adapted to reproduce and to maintain their *intelligence*.

4.1 Chromosome codification

A chromosome is represented using two tables. The first one is formed by $2 \cdot k \cdot 8$ components specifying the Gaussians $\lfloor E_{t-1} \rfloor \times \lfloor L_t \rfloor$ which represent $P(E_t | E_{t-1} \otimes L_t \otimes \pi_W)$.

The second table is formed by the same component numbers specifying the Gaussians $\lfloor E_t \rfloor \times \lfloor L_t \rfloor$ which represent $P(A_t | E_t \otimes L_t \otimes \pi_W)$. In this way, each chromosome contains $32 \cdot k$ gens.

In the described experiments, the initial chromosome population is obtained by randomly initializing the Gaussian parameters in the following range:

$$G_1(\mu, \sigma) : \begin{cases} \mu = [0, ..., k] \\ \sigma = [0, 0.25, 0.50, 0.75, 1] \end{cases}$$

for the probability $P(E_t | E_{t-1} \otimes L_t \otimes \pi_W)$ and

$$G_2(\mu, \sigma) : \begin{cases} \mu = [0, 1, 2] \\ \sigma = [0, 0.25, 0.50] \end{cases}$$

for $P(A_t | E_t \otimes L_t \otimes \pi_W)$.

4.2 Fitness function

The evaluation function contains specific knowledge that is used to assess the quality of the solutions.

In our case we want to reward the worms that live the longest time in the world. In this way we describe the fitness function as the number of iterations that a worm lives in a randomized generated world. In order to avoid the situation

where a simple world produces an overvalued worm, we generate w random worlds to evaluate each worm's fitness.

As we commented previously each world is composed of cells (empty or full of poison) contained within a wall. All worlds are the same size and have the same wall disposition, only the quantity and position of poisonous cells varies, being selected randomly and comprising between 5% and 10% of the total cells.

4.3 Selection, crossover an mutation operators

The selection phase in a genetic algorithm involves creating a *mating pool* by selecting individual solutions that are fitter with a higher probability. The selected individuals in the mating pool are then combined to create a new population using the crossover operator, with occasional small random changes due to the mutation operator. We are going to show the operators that have provided the best results in the experimentation test.

Selection operator. We used a stochastic remainder sampling selector (SRS) with a two-staged selection procedure. In the first stage, each individual's expected representation is calculated. A temporary population is filled using the individuals with the highest expected numbers. Any fractional expected representations are used to give the individual more likelihood of filling a space. The second stage of selection is uniform random selection from the temporary population. In addition we use elitism (the best individual from each generation is carried over to the next generation).

Crossover operator. We use an asexual two-point crossover operator. In this way the mother genes will be selected until the crossover point where the father genes will be copied. This process will be done for the two tables (see figure 3) that describe the chromosome.

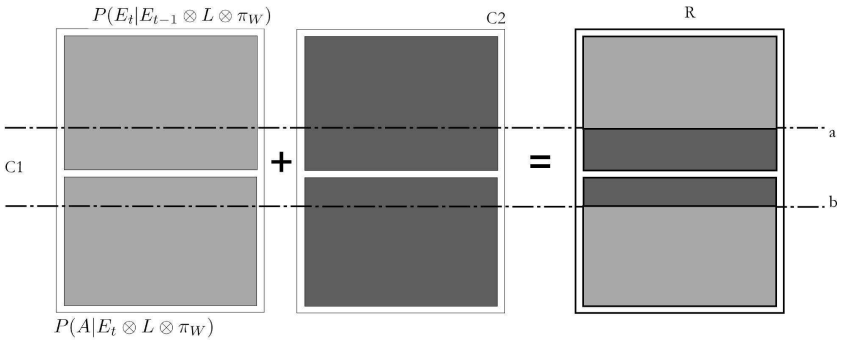

Fig. 3. Asexual two point crossover operator for the worm's chromosome.

Mutation operator We define an incremental mutation operator for states, in this way given a gene x we define a mutation as: $x \in [0, k], mut(x) = x + 1\, MOD\, k$. Suppose we have four states, and that $q_2 = 3$, if we mutate this element we will obtain $q_2 = 3 + 1\, MOD\, 4 = 0$. A random mutation scheme is used to choose the directions for the worm to take. A new direction is generated randomly and then substitutes the original gene.

4.4 Used parameters

Using the operators presented in the previous section we obtain an evolutive learning process for a worm. Developing empirical tests we arrive at the conclusion that a number of states (k) greater than five complicates the learning process of the worm and does not improve the movements made by the worm. For this reason, in the rest of the experiments, we use a fixed number of states equal to five (see figure 4).

In addition, for the remainder of tests we use $d = 5$ (for each five cells the worm moves through it will increase it's size by one unit) and $w = 6$ (six random worlds will be generated in order to evaluate each worm).

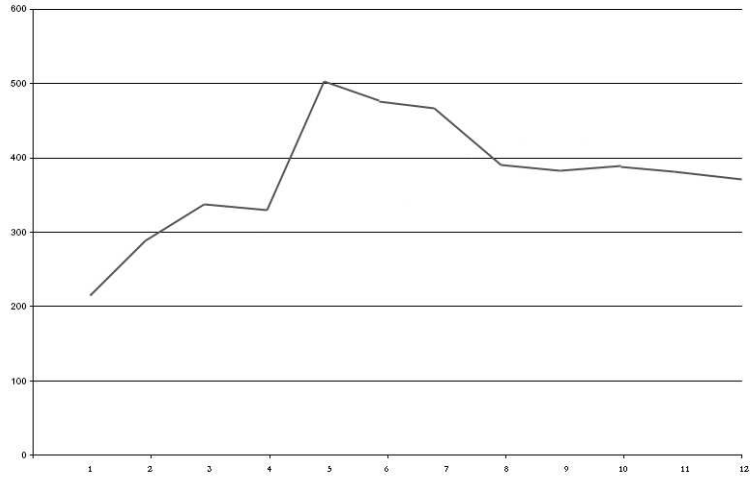

Fig. 4. Maximum evaluated individual for each number of states. The y axis represents the worm's fitness and the x axis the number of states used (k). We utilize 100 executions for each state with a population of 250 individuals and 500 generations using the operators specified in the previous section.

4.5 Worms evolution

In order to obtain the best individual we use 100 executions for each state with a population of 250 individuals and 500 generations using the operators specified in the previous section. In figure 5 an example is shown of the executions showing the fitness of the worst and best performing individual as well as the average results obtained, illustrating the algorithm convergence. For each algorithm execution the evaluation took about 2 minutes using a Pentium IV running at 2Ghz.

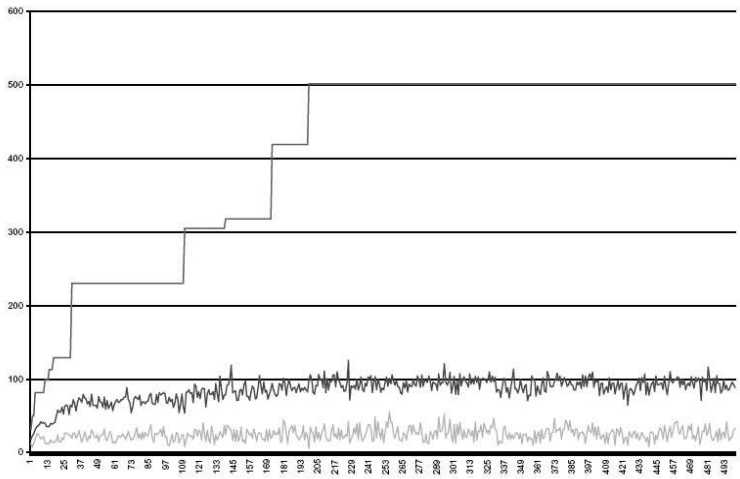

Fig. 5. We show the evolution of the worst (bottom), the average (in the middle) and the best performing individual (top). The y axis represents the worm's fitness and the x axis the actual generation. Until the first 50 iterations an improvement is produced in the medium and the worst individual. Then the graph tends to oscillate although a slight increase is produced (because the best case increases and maintains it's level through elitism).

5 Survival behaviours and experimentation

In this section we will analyze some characteristics and emergent behaviours that were observed in the worms. Readers of this paper are invited to test our simulator at the following address `http://www.dccia.ua.es/~fidel/worm.zip`.

Using Bayesian Programming the worm's previous knowledge is defined and mechanisms are given to provide new knowledge to the worm. This data is represented using two sets of discrete Gaussians which were learned using genetic algorithms. However, we should remember that to get the information of the learned distributions we use the *Draw* function which randomly extracts a value

for the distribution. In this way we obtain a non-deterministic behaviour, which is more adaptable to variations in complex worlds.

After training the worm population we simulate, in a graphical way, the best individual found. It is curious to see different behaviour patterns, which provide more survival opportunities. Some of these patterns even seem to imitate natural behaviour developed in some animals.

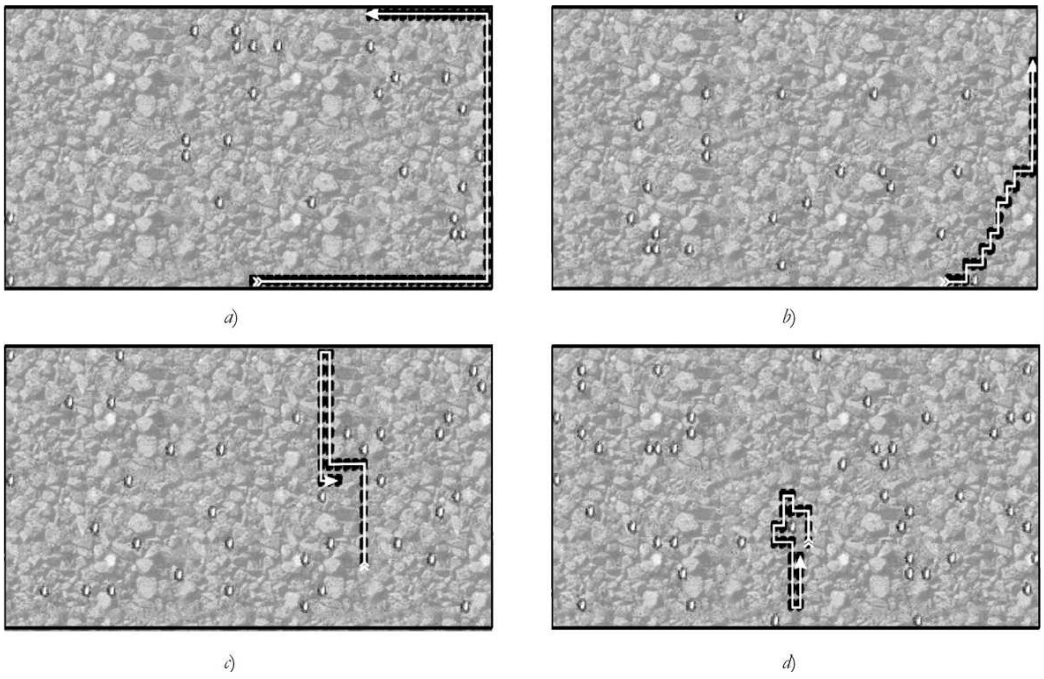

Fig. 6. Different behavior patterns. *a*) Follow the edge of the world. *b*) Zigzag movement. *c*) *Ping-pong* behaviour. *d*) In the movement the worm seems to follow its tail. The arrow points to the next worm displacement.

One of the most common patterns is to follow the edge of the world while no poison is found near it (see figure 6a). This is a good way to move if the proportion of poison is low near the edges and configurations don't exist that trap the worm between the perimeters and the poison. Another curious behaviour is the development of a zigzag movement emulating the way some snakes move (see figure 6b) so reducing the area that the worm occupies in the world. In addition it is quite common for the worm to move up and down like a ping-pong ball (see figure 6c). Finally, we underline the movement of some worms which seem to move as if trying to reach their tails, so forming a spiral (see figure 6d).

The behaviour described above (and some others) are repeated and combined with the obtained worms. These behaviour are not programmed implicitly, they

have been obtained using the proposed *brain* model and selected using an evolving process in a population.

5.1 Comparing our model with a clasical one

We can see some advantages of our method if we compare it to a more classical model approach, the finite states machine (FSM) [6]. This approach has various drawbacks. First, it assumes a perfect world model, which is false. It is necessary to know that any model of a real phenomenon is incomplete because there will always exist non-considered, hidden variables, that will influence the phenomenon. The effect of these variables is malicious since they will cause the model and the phenomenon to show different behavioural patterns

Second, a FSM develop a deterministic behaviour therefore in certain world configurations it will fail. On the other hand, a Bayesian model has not a deterministic behaviour and two different executions in the same world may have different results, which provide greater adaptability to changes in the environment configuration.

6 Conclusions

In this paper we have seen an application of Bayesian programming in an artificial life system. The formalism of the artificial life models is a continuous field of investigation because of the complexity of the systems we work with [7],[8]. In addition, we have the added difficulty of working with uncertainty and include it into the model we want to use. The Bayesian programming brings up a formalism where implicitly, using probabilities, we work with the uncertainly.

In a world with randomly distributed poison lives a worm, which main purpose is to grow up. We propose a formalization of the virtual worm using a decomposition in terms of a joint probability distribution of their knowledge. In this way, applying the Bayesian Programming we obtain a versatile behaviour adaptable to changes and what is more a mathematical description of the probabilistic environment model.

We have seen some advantages of our method comparing it to a more classical model approach, the finite states machine (FSM [6]) (see section 5.1).

The learning process, given a worm population, has been developed with evolving techniques, using genetic algorithms. The principal reason for using GA was because they are a global search technique which mimic aspects of biological evolution even though other search techniques could be picked to select the worms. Each used chromosome is the codification of the two distributions obtained with the previous Bayesian formalism (see figure 3).

Satisfactory results were obtained that prove the validity of the proposed model. Relatively complex and elaborate behavioural patterns were observed in the movements of the most highly adapted worms. These behaviour patterns were not implicitly programmed but were obtained in an emergent way using the proposed model.

Bayesian programming is, therefore, a promising way to formalize both artificial and natural system models. In this example, we have seen how this paradigm can be adapted to a simple, artificial life problem. Future studies will try to model different artificial life systems using this new formalism.

References

1. Lebeltel, O., Bessière, P., Diard, J., Mazer, E.: Bayesian robots programming. Autonomous Robots **16** (2004) 49–79
2. Bessière, P., Group, I.R.: Survei:probabilistic methodology and tecniques for artefact conception and development. INRIA (2003)
3. Koike, C., Pradalier, C., Bessiere, P., Mazer, E.: Proscriptive bayesian programming application for collision avoidance. Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems (IROS); Las Vegas, USA (2003)
4. C. Coué, Th. Fraichard, P.B., Mazer, E.: Using bayesian programming for multisensor data fusion in automotive applications. IEEE Intelligent Vehicle Symposium (2002)
5. Bellot, D., Siegwart, R., Bessière, P., Coué, C., Tapus, A., Diard, J.: Bayesian reasoning for real world robotics: Basics, scaling and examples. Book Chapter in LNCS/LNAI, `http://128.32.135.2/users/bellot/files/David_Bellot_LNCS_LNAI.pdf` (2004)
6. Dysband, E.: Game Programming Gems. A finite-state machine class (237-248). Charles River Media (2000)
7. Agre, P., Horswill, I.: Lifeworld analysis. Journal of Artificial Intelligence Research **6** (1997) 111–145
8. Rasmussen, S., L. Barrett, C.: Elements of a theory of simulation. ECAL1995 (Advances in Artificial Life, Third European Conference on Artificial Life, Granada, Spain). (1995) 515–529