An evolutionary behavioral model for decision making

Oscar Javier Romero López

Adaptive Behavior 2011 19: 451
DOI: 10.1177/1059712311419680

Original Paper

Adaptive Behavior

Adaptive Behavior
19(6) 451–475
© The Author(s) 2011
Reprints and permissions:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/1059712311419680
adb.sagepub.com
SAGE

An evolutionary behavioral model for decision making

Oscar Javier Romero López

Abstract

For autonomous agents the problem of deciding what to do next becomes increasingly complex when acting in unpredictable and dynamic environments while pursuing multiple and possibly conflicting goals. One of the most relevant behavior-based models that tries to deal with this problem is the *behavior network* model proposed by Maes. This model proposes a set of behaviors as purposive perception–action units that are linked in a nonhierarchical network, and whose behavior selection process is orchestrated by spreading activation dynamics. In spite of being an adaptive model (in the sense of self-regulating its own behavior selection process), and despite the fact that several extensions have been proposed in order to improve the original model adaptability, there is not yet a robust model that can self-modify adaptively both the topological structure and the functional purpose of the network as a result of the interaction between the agent and its environment. Thus, this work proposes an innovative hybrid model driven by gene expression programming, which makes two main contributions: (1) given an initial set of meaningless and unconnected units, the evolutionary mechanism is able to build well-defined and robust behavior networks that are adapted and specialized to concrete internal agent's needs and goals; and (2) the same evolutionary mechanism is able to assemble quite complex structures such as deliberative plans (which operate in the long-term) and problem-solving strategies. As a result, several properties of self-organization and adaptability emerged when the proposed model was tested in a robotic environment using a multi-agent platform.

Keywords

Intelligent and autonomous agents, evolutionary computation, gene expression programming, behavior networks, adaptive behavior, automated planning

1 Introduction

An autonomous agent is a self-contained program that is able to control its own decision-making process, sensing and acting autonomously in its environment, and by doing so realize a set of goals or tasks for which it is designed. Usually these goals and tasks change dynamically through time as a consequence of internal and external (environmental) perturbations and, ideally, the agent should adapt its own behavior to these perturbations. Maes (1989, 1990, 1992) proposed a model for building such an autonomous agent that includes a mechanism for action selection (MASM) in dynamic and unpredictable domains, based on so-called behavior networks. This model specifies how the overall problem can be decomposed into subproblems, that is, how the construction of the agent can be decomposed into the construction of a set of component modules (behaviors) and how these modules should be made to interact. The total set of modules and their interactions provide an answer to the question of how the sensor data and the current internal state of the agent determine the actions (effector outputs) and future internal state of the agent through an activation spreading mechanism that determines the best behavior to be activated in each situation. In addition, this model combines characteristics of both traditional planners and reactive systems: it produces fast and robust activity in a tight interaction loop with the environment, while at the same time allowing for some prediction and planning to take place.

Although original Maes networks do work in continuous domains, they do not exploit the additional information provided by continuous states. Similarly, though there are mechanisms to distinguish different types of goals in MASM, there are no means to support goals with a continuous truth state (such as "have

Fundación Universitaria Konrad Lorenz, Bogotá, Colombia

Corresponding author:
Oscar J Romero López, Fundación Universitaria Konrad Lorenz, Carrera 9 Bis No. 62 - 43, Bogotá, Colombia
Email: ojrlopez@hotmail.com

stamina") to become increasingly demanding the less they are satisfied. In regard to this, some extensions to the original behavior network model have been proposed (Dorer, 1999, 2004).

One of the most relevant weakness of the original Maes model is that the whole network model is fixed. It therefore requires both the network structure (e.g., spreading activation links) and global parameters of the network that define the characteristics of a particular application (e.g., goal-orientedness vs. situation-orientedness, etc.) to be pre-programmed, and hence the agent has no complete autonomy over its own decision-making process. In order to resolve this problem, Maes (1991) proposed two mechanisms depending on real-world observations: a learning mechanism for adding/deleting links in the network, and an introspection mechanism for tuning global parameters. The main problem with the former is that it does not use a real machine learning algorithm, but rather a simple statistical process based on observations, so many hand-coded instructions are still required. With respect to the latter, it proposes a meta-network (another behavior network) that controls the global parameter variation of the first network through time, but the problem still remains: who is in charge of dynamically adapting the global parameters of the meta-network? It seems to be similar to the well-known homunculus problem of cognitive psychology; or, in colloquial terms, the Russian nested dolls (matryoska dolls) effect.

This work proposes a novel model based on gene expression programming (GEP; Ferreira, 2001) that, on the one hand, allows the agent to self-configure both the topological structure and the functional characterization of each behavior network without losing the required expressiveness level; and, on the other hand, allows the agent to build more complex decision-making structures (e.g., deliberative plans and problem solving strategies) from the assembly of different kinds of evolved behavior networks. In contrast to the behavior network extensions mentioned above, this approach confers a high level of adaptability and flexibility, always producing, as a consequence, syntactically and semantically valid behavior networks.

The remainder of this article is organized as follows. Section 2 outlines the operation of the behavior network model. Section 3 explains in detail how the behavior network model is extended using GEP. Section 4 illustrates the plan-extracting process using GEP. Section 5 outlines and discusses the results of the experiments. The concluding remarks are given in Section 6.

2 Behavior networks model

In the following, we describe the behavior network formalism. Since we do not need the full details for our purposes, the description will be sketchy and informal at some points.

A behavior network (BN) is a mechanism proposed by Maes (1989) as a collection of competence modules that work in a continuous domain. Action selection is modeled as an emergent property of an activation / inhibition dynamics among these modules. A behavior i can be described by a tuple $\langle c_i, a_i, d_i, \alpha_i \rangle$. c_i is a list of preconditions which have to be fulfilled before the behavior can become active and $e = \tau(c_i, s)$ is the executability of the behavior in situation s where $\tau(c_i, s)$ is the (fuzzy) truth value of the precondition in situation s. a_i and d_i represent the expected (positive and negative) effects of the behavior's action in terms of an *add list* and a *delete list*. Additionally, each behavior has a level of activation α_i. If the proposition X about environment is true and X is in the precondition list of the behavior A, there is an active link from state X to action A. If goal Y has an activation greater than zero and Y is in the add list of behavior A, there is an active link from goal Y to action A.

Internal links include predecessor links, successor links, and conflicter links. There is a successor link from behavior A to behavior B (A has B as successor) for every proposition p that is a member of the add list of A and also a member of the precondition list of B (so more than one successor link between two competence modules may exist). A predecessor link from module B to module A (B has A as predecessor) exists for every successor link from A to B. There is a conflicter link from module A to module B (B conflicts with A) for every proposition p that is a member of the delete list of B and a member of the precondition list of A. The following is the procedure to select an action to be executed at each step:

1. Calculate the excitation coming in from the environment and the goals.
2. Spread excitation along the predecessor, successor, and conflicter links, and normalize the behavior activations so that the average activation becomes equal to the constant π.
3. Check any executable behaviors, choose the one with the highest activation, execute it, and finish. A behavior is executable if all the preconditions are true and if its activation is greater than the global threshold. If no behavior is executable, reduce the global threshold and repeat the cycle.

Additionally, the model defines five global parameters that can be used to "tune" the spreading activation dynamics of the BN and thereby affect the operation of the behavior network:

1. π: the mean level of activation.
2. θ: the threshold for becoming active. θ is lowered by 10% each time none of the modules could be

selected. It is reset to its initial value when a module could be selected.
3. ϕ: the amount of activation energy a proposition that is observed to be true injects into the network.
4. γ: the amount of activation energy a goal injects into the network.
5. δ: the amount of activation energy a protected goal takes away from the network.

In the following section, we describe how the BN topology can be evolved in order to adapt to continuously changing goals and states of the environment. An approach to how complex decision-making structures (e.g., deliberative plans) can emerge from the interaction of multiple evolved BNs, is also presented.

3 Evolutionary behavior networks

We propose an extended version of Maes' model, described above, that incorporates more sophisticated, rational, and complex processing modules than the simple state machines proposed by Maes.[1] In addition, it incorporates an evolutionary mechanism addressed by gene expression programming (GEP; Ferreira, 2001) in charge of evolving the BN topology, namely, the activation / inhibition links among behaviors, the preconditions of each behavior, and the algorithm's global parameters. This section explains how the chromosomes of GEP can be modified so that a complete BN—including the architecture, the activation/inhibition links, and the global parameters—can be totally encoded by a linear chromosome, even though it may be expressed as a nonlinear structure such as an expression tree. It is also shown how this chromosomal organization allows the adaptation of the network using the evolutionary mechanisms of selection and modification, thus providing an approach to the automatic design of BNs.

The main reason we have used GEP, instead of typical genetic programming (GP) or other kinds of evolutionary algorithms (e.g., genetic algorithms, GA), as a behavior–network evolutionary mechanism is due to its powerful and robust capability of always creating complex and meaningful structures. The fundamental difference between the three algorithms resides in the nature of the individuals: in GA the individuals are symbolic strings of fixed length (chromosomes); in GP the individuals are nonlinear entities of different sizes and shapes (parse trees); and in GEP the individuals are encoded as symbolic strings of fixed length (chromosomes), which are then expressed as nonlinear entities of different sizes and shapes (expression trees). Thus, the structural and functional organization of GEP genes always guarantees the production of valid solutions, no matter how much or how profoundly the chromosomes are modified.

3.1 Genetic encoding of behavior networks

The network architecture is encoded in the familiar structure of a head and tail (Poli, Langdon, & McPhee, 2008). The head contains special *functions* that activate the units, and *terminals* that represent the input units. The tail contains only *terminals*. Let us now analyze an example of how the BN is encoded into a GEP chromosome.

In Figure 1a, a linear multigenic chromosome is initially generated in a random way and then modified by genetic operators. Each multigenic chromosome defines several *behavioral genes* and just one *functional gene*. Each behavioral gene encodes a different behavior's structure, whereas the functional gene encodes the global parameters of the BN. We propose a multigenic chromosomal structure, which is more appropriate for evolving good solutions to complex problems because it permits the modular construction of complex, hierarchical structures, where each gene encodes a smaller and simpler building block (a behavior). These building blocks are physically separated from one another and thus can evolve independently. Not surprisingly, these multigenic systems are much more efficient than unigenic ones (Ferreira, 2000). The details of the encoding process will be explained later.

The multigenic chromosome can then be translated into the whole expression tree shown in Figure 1b through the conversion process described by Poli et al. (2008) and Ferreira (2006). Here, it is possible to identify three kinds of functions: **B**, **D**, and **T**. The **B** function is used for representing each behavior of the net, and it has an arity of three: the first branch is a set of preconditions, the second is a set of activation links that connects to other behaviors, and the third is a set of inhibition links that connects to other behaviors (dashed arrows). For example, behavior B_3 is activated when behavior B_2 makes true precondition P_4. After that, it spreads activation energy to behavior B_1 through precondition P_5, and spreads inhibitory energy to behavior B_2 through precondition P_3. The **D** and **T** functions are connectivity functions that join two or three elements, respectively, of the same nature (e.g., behaviors, preconditions, goals, etc.). Note that the expression tree is composed of several subexpression trees (sub-ETs), each one representing the structure of a unique behavior in the net; hence each sub-ET has a particular organization that is encoded into one separate behavioral gene, and the whole expression tree (ET) models the entire behavior network.

Figure 1c depicts a basic BN with three behaviors (*B1*, *B2*, and *B3*), where the solid arrows denote excitatory activation connections and the dashed arrows denote inhibition connections between behaviors. P_1, P_2, P_3, P_4, and P_5 denote the preconditions for behaviors. In order to simplify the picture, each behavior only defines a few preconditions. However, in the real

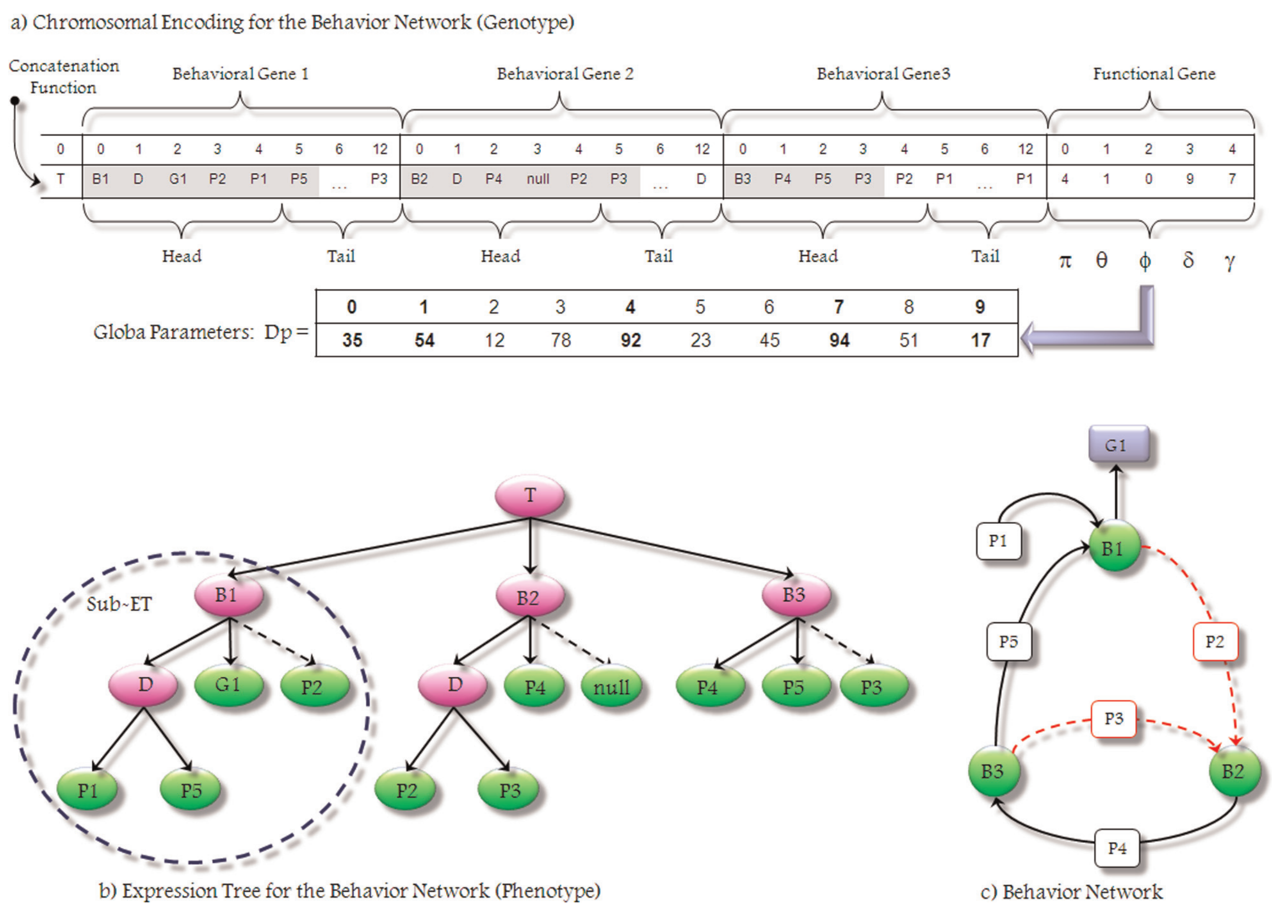

a) Chromosomal Encoding for the Behavior Network (Genotype)
Concatenation Function
Behavioral Gene 1
Behavioral Gene 2
Behavioral Gene3
Functional Gene
0 T
0 1 2 3 4 5 6 12 | B1 D G1 P2 P1 P5 … P3
0 1 2 3 4 5 6 12 | B2 D P4 null P2 P3 … D
0 1 2 3 4 5 6 12 | B3 P4 P5 P3 P2 P1 … P1
0 1 2 3 4 | 4 1 0 9 7
Head
Tail
π θ ϕ δ γ
Globa Parameters: Dp =
0 1 2 3 4 5 6 7 8 9
35 54 12 78 92 23 45 94 51 17
Sub-ET
T
B1 B2 B3
D G1 P2 D P4 null P4 P5 P3
P1 P5 P2 P3
b) Expression Tree for the Behavior Network (Phenotype)
G1 P1 B1 P5 P2 P3 B3 B2 P4
c) Behavior Network

Figure 1. GEP translation of a behavior network.

implementation the preconditions set for each behavior might be composed of a subset of sensory inputs (internal and external), a subset of active working memory elements, a subset of current subgoals, and a subset of motivational states (drives, moods, and emotions). G_1 is an agent's global goal pursued by behavior B_1.

In the example in Figure 1a, for each behavioral gene, positions from 0 to 3 encode the head domain (so both functions and terminals are allowed), and positions from 4 to 12 encode the tail domain (where only terminals are allowed). Because each behavior defines variable sets of preconditions, activation links, and inhibition links, the corresponding genetic encoding spans along regions of different sizes into the behavioral gene. Those regions are called *open reading frames* (ORF; Ferreira, 2001). In GEP, what changes is not the length of genes, but rather the length of the ORF. Indeed, the length of an ORF may be equal to or less than the length of the gene. The noncoding regions are the essence of GEP and evolvability, because they allow the modification of the genome using all kinds of genetic operators without any kind of restriction. In fact, genetic operators can work on both regions (ORF and noncoding regions), always producing syntactically correct behavior networks.

Each sub-ET can be generated straightforwardly from chromosomal representation as follows: first, the start of a gene corresponds to the root of the sub-ET, forming this node in the first line; second, depending on the number of arguments to each element (functions may have a different number of arguments, whereas terminals have an arity of zero), in the next line are placed as many nodes as there are arguments to the functions in the previous line; third, from left to right, the nodes are filled, in the same order, with the elements of the gene; and fourth, the process is repeated until a line containing only terminals is formed.

Because the process is bidirectional, inversely each behavioral gene can be easily inferred from the corresponding sub-ET as follows: the behavior function (B) of the sub-ET is encoded, and the algorithm then makes a straightforward reading of the sub-ET from left to right and from top to bottom (exactly as one reads a page of text). For instance, the sub-ET for behavior B_1 is encoded as: **B1-D-G1-P2-P1-P5**, and this is the ORF of its corresponding behavioral gene (i.e., the shadowy region for this gene in Figure 1).

The functional gene encodes an additional domain called D_p, which represents the global parameters of the BN. For the functional gene, position 0 encodes π (the mean level of activation), position 1 encodes θ (the threshold for becoming active), position 2 encodes ϕ (the amount of energy for preconditions), position 3 encodes δ (the amount of energy for protected goals), and position 4 encodes γ (the amount of energy for goals). The values of global parameters are kept in an array and are retrieved as necessary. The number represented by each position in the parameters domain indicates the order in the array D_p. For example, position 0 in the functional gene (π) encapsulates the index “4” which corresponds to the value 92 in the D_p array (in bold), and so on. For simplicity, Figure 1 only shows an array of 10 elements for parameter domain D_p, but in the implementation we use an array of 100 elements, where each position encodes one numeric value between 0 and 100. Genetic operators guarantee that global parameters are always generated inside the domain of the D_p array.

3.2 Special genetic operators

The evolution of such complex entities composed of different domains and different alphabets requires a special set of genetic operators so that each domain remains intact. The operators of the basic gene expression algorithm (Ferreira, 2001) are easily transposed to behavior-net encoding chromosomes, and all of them can be used provided the boundaries of each domain are maintained so that alphabets are not mixed up. Mutation was extended to all the domains so that every different gene (behavioral or functional) was modified following its respective domain constraints (e.g., not replacing terminal nodes by function nodes in the tail region, etc.). Insertion sequence (IS) and root insertion sequence (RIS) transposition were also implemented in behavioral genes and their action is obviously restricted to heads and tails. In the functional gene we define only an IS operator (because the RIS operator is not applicable here) that works within the D_p domain, ensuring the efficient circulation of global parameters in the population. Another special operator, parameters’ mutation, was also defined in order to directly introduce variation in the functional gene (i.e., global parameters region), selecting random values from the D_p array.

The extension of recombination and gene transposition to GEP-nets is straightforward, as their actions never result in mixed domains or alphabets. However, for them to work efficiently (i.e., allow an efficient learning and adaptation), we must be careful in determining which behavior’s structure elements and/or global parameters go to which region after the splitting of the chromosomes, otherwise the system is incapable of evolving efficiently. In the case of gene recombination and gene transposition, keeping track of behavioral and functional genes is not a difficult task, and these operators work very well in GEP-nets. But in one-point and two-point recombination where chromosomes can be split anywhere, it is impossible to keep track of the behavior’s structure elements and global parameters. In fact, if applied straightforwardly, these operators would produce such large evolutionary structures that they would be of little use in multigenic chromosomes (Ferreira, 2006). Therefore, for our multigenic system, a special intragenic two-point recombination was used so that the recombination was restricted to a particular gene (instead of interchanging genetic material with other kinds of genes in the chromosome).

In summary, in order to guarantee the generation of valid BNs, all genetic operators have to comply with the following constraints:

- In the first position of behavioral genes, only a B (behavior) node can be inserted.
- For the head region in behavioral genes:
  - Mutation only by connectivity functions (D and T), and by terminals such as preconditions (P_n) and goals (G_n).
  - Transposition (IS and RIS) and one-point and two-point recombination operators must follow the same syntactic validations as the mutation operator.
- For the tail region in behavioral genes:
  - Terminals can only be mutated, transposed, and recombined using elements from the tail domain, such as preconditions (P_n) and goals (G_n). No syntactic validations are required.
- For global parameters in the functional gene:
  - Terminals can only be mutated, transposed, and recombined using numeric values from parameters domain D_p, that is, numeric values between 0 and 100. No additional syntactic validations are required.

Finally, for each behavioral gene, the length of the head h is chosen depending on the problem domain (e.g., for the experiments we used $h = 10$). This parameter allows at least the encoding of all behaviors set. On the other hand, the length of the tail t is a function of both h and the number of arguments n of the function with more arguments (also called maximum arity), and is evaluated by Equation 1:

$$t = h(n-1) + 1 \qquad (1)$$

In our case, $n = 3$ because this is the maximum arity for functions B and T. If we define $h = 10$, then $t = 20$, so the maximum fixed length for each behavioral gene is 30, and for the functional gene it is 5 (the maximum number of global parameters).

3.3 Fitness functions for BN chromosomes

In this section we describe how behavior-network chromosomes are evaluated so that they have higher or lower probability of being replicated in the next generation of the evolutionary process. For the fitness evaluation we have taken into account the theorems proposed by Nebel and Babovich-Lierler (2004) and, additionally, we have identified a set of necessary and sufficient conditions that make behavior networks goal converging. Note that all reinforcement parameters used in the next fitness functions are self-generated by the system from changes observed in the agent's internal states, so that they do not require a priori or manual adjustment made by a designer.

We first define two fitness functions: one evaluates how well defined the behavior-network structure is, and the other evaluates the efficiency and functionality of the behavior network. First, the fitness function for evaluating the behavior-network structure is

$$FFS_i = A_i + B_i + C_i + D_i + E_i + F_i, \tag{2}$$

where i is a chromosome encoding a specific BN, and each term is defined as follows.

A_i: Is there at least one behavior of the net accomplishing a goal? such that

$$A = \begin{cases} a1, & \text{if } \exists\, beh \in i \mid a_{beh} \cap G(t) \neq \emptyset \\ a2, & \text{otherwise} \end{cases} \tag{3}$$

where beh is any behavior of the network, a_{beh} is the add list of beh, $G(t)$ is a set of global goals, $a1$ is a positive reinforcement (by default 100), and $a2$ is a negative reinforcement (by default -100).

B_i: Are all behaviors of the net well connected? such that

$$B = n_{cp} \cdot b1 + n_{up} \cdot b2, \tag{4}$$

where n_{cp} is the number of behaviors correctly connected to others through successor and predecessor links (self-inhibitory connections are incorrect), n_{up} is the number of unconnected behaviors (no propositions at either *add list* or *delete list*), $b1$ is a positive reinforcement (by default $+10$), and $b2$ is a negative reinforcement (by default -20).

C_i: Are there any deadlock loops defined by the BN? such that

$$C = \begin{cases} (n_p * c1) + (n_{np} * c2), & \text{if the BN} \\ \quad \text{has associated a global goal} & \\ c3, & \text{otherwise} \end{cases} \tag{5}$$

where n_p is the number of behaviors that define at least one path connecting to the global goal, $c1$ is a positive reinforcement (by default $+20$), n_{np} is the number of behaviors without a path between them and the global goal, $c2$ is a negative reinforcement (by default -10), and $c3$ is another negative reinforcement (by default -50).

D_i: Are all propositions (preconditions, add list, and delete list) of each behavior unambiguous? (e.g., the precondition set is ambiguous if it has propositions p and $-p$ at the same time) such that

$$D = \sum_{i=0}^{k} (n_{na} * d1) + (n_a * d2), \tag{6}$$

where k is the total number of behaviors (behavioral genes) of the BN, n_{na} is the number of propositions that are not ambiguous, $d1$ is a positive reinforcement (by default $+10$), n_a is the number of ambiguous propositions, and $d2$ is a negative reinforcement (by default -20).

E_i: Are all add-list propositions non-conflicting? (e.g., a proposition that appears both in the add list and in the delete list—for the same behavior—is a conflicting proposition), such that

$$E = \sum_{i=0}^{k} (n_{nca} * e1) + (n_{ca} * e2), \tag{7}$$

where k is the total number of behaviors of the BN, n_{nca} is the number of non-conflicting add-list propositions, $e1$ is a positive reinforcement (by default $+10$), n_{ca} is the number of conflicting add-list propositions, and $e2$ is a negative reinforcement (by default -20).

F_i: Are all delete-list propositions non-conflicting? (e.g., a proposition that appears both in the preconditions set and in the delete list—for the same behavior—is a conflicting proposition), such that:

$$F = \sum_{i=0}^{k} (n_{ncd} * f1) + (n_{cd} * f2), \tag{8}$$

where k is the total number of behaviors of the BN, n_{ncd} is the number of non-conflicting delete-list propositions, $f1$ is a positive reinforcement (by default $+10$), n_{cd} is the number of conflicting delete-list propositions, and $f2$ is a negative reinforcement (by default -20).

Second, the fitness function for evaluating network functionality is:

$$FFE_i = G_i + H_i + I_i + J_i + L_i + M_i + N_i, \tag{9}$$

where i is a chromosome encoding a specific BN, and each term is defined as follows.

G_i: This term determines if γ (the amount of energy for goals) is a well-defined parameter. Because the parameter γ must reflect the "goal-orientedness" feature of the BN,

$$G = \begin{cases} 100 - g1, & \text{if} (R_{freqG} > 0 \wedge R_\gamma > 0) \vee \\ & (R_{freqG} < 0 \wedge R_\gamma < 0) \\ g2, & \text{otherwise} \end{cases} \tag{10}$$

where R_{freqG} is the absolute variation rate which determines how often a goal is activated or reactivated by the internal agent's motivational subsystem. Therefore

$$R_{freqG} = \frac{freqG_{cur} - freqG_{pri}}{freqG_{cur}}, \tag{11}$$

where $freqG_{cur}$ is a frequency indicating how many goals are activated in the current state, and $freqG_{pri}$ is a frequency indicating how many goals were activated in a prior state. R_γ is the absolute variation rate for parameter γ:

$$R_\gamma = \frac{\gamma_{cur} - \gamma_{pri}}{\gamma_{cur}}, \tag{12}$$

where γ_{cur} is the value for γ in the current state, and γ_{pri} is the value of γ in the prior state. Finally, $g1$ is the absolute difference among the variation rates: $g1 = | R_{freqG} - R_\gamma |$; and $g2$ is a negative reinforcement (by default -100). Intuitively, when the frequency of activated goals increases over time, the global parameter γ should increase proportionally too.

H_i: This term determines if ϕ (the amount of energy for preconditions) is a well-defined parameter. Because the parameter ϕ must reflect the "situation relevance" and "adaptivity" features of the BN,

$$H = \begin{cases} 100 - h1, & \text{if}(R_{freqC}>0 \wedge R_\phi>0) \vee (R_{freqC}<0 \wedge R_\phi<0) \\ h2, & \text{otherwise} \end{cases} \tag{13}$$

where $h1$ is the absolute difference among the absolute variation rates; $h1 = | R_{freqC} - R_\phi |$. R_{freqC} is a variation rate (between current and prior states) that determines how often the environmental perturbations are perceived by the agent. R_ϕ denotes the absolute variation rate for parameter ϕ and $h2$ is a negative reinforcement (by default -100). Note that the absolute variation rates are treated similarly as in the term G.

I_i: This term determines if π (the mean level of activation) is a well-defined parameter. Because the parameter π must reflect the "adaptivity" and "bias to ongoing plans" features of the BN,

$$I = \begin{cases} 100 - i1, & \text{if}(R_{freqSG}>0 \wedge R_\pi>0) \vee \\ & (R_{freqSG}<0 \wedge R_\pi<0) \\ i2, & \text{otherwise} \end{cases} \tag{14}$$

where $i1$ is the absolute difference among the absolute variation rates; $i1 = | R_{freqSG} - R_\pi |$. R_{freqSG} is a variation rate (between a current and prior states) that determines the activation frequency of the subgoals set that are associated to a current global goal. R_π denotes the absolute variation rate for parameter π and $i2$ is a negative reinforcement (by default -100). Absolute variation rates are treated similarly as in the term G. Intuitively, if the environment requires the agent to address its actuation to the achievement of a hierarchical set of goals, the BN must increase the value of π through time; otherwise, if the environment is quite dynamic and an adaptive behavior is required, the parameter π should decrease.

J_i: This term determines if δ (the amount of energy for protected goals) is a well-defined parameter. Because the parameter δ must reflect the "avoiding goal conflicts" feature of the BN,

$$J = \begin{cases} 100 - j1, & \text{if}(R_{auto}>0 \wedge R_\delta<0) \vee (R_{auto}<0 \wedge R_\delta>0) \\ j2, & \text{otherwise} \end{cases} \tag{15}$$

where $j1$ is the absolute difference between the absolute variation rates; $j1 = | R_{auto} - R_\delta |$. R_{auto} is the absolute variation rate for the number of self-referenced loops identified by the agent between current and prior states (e.g., when the system identifies a circular reference of behavior activation such as: $a \rightarrow b$, $b \rightarrow c$, $c \rightarrow a$). R_δ denotes the absolute variation rate for parameter δ and $j2$ is a negative reinforcement (by default -100). Intuitively, if R_{auto} increases, then R_δ should decrease proportionally, and vice versa.

L_i: This term determines if θ (the threshold for becoming active) is a well-defined parameter. Because the parameter θ must reflect the "bias to ongoing plans," "deliberation," and "reactivity" features of the BN,

$$L = \begin{cases} 100 - l1, & \text{if}(R_{freqCE}>0 \wedge R_\theta<0) \vee (R_{freqCE}<0 \wedge R_\theta>0) \\ l2, & \text{otherwise} \end{cases} \tag{16}$$

where $l1$ is the absolute difference among the absolute variation rates; $l1 = | R_{freqCE} - R_\theta |$. R_{freqCE} is the absolute variation rate for the number of changing environmental elements between current and prior states (e.g., novel objects coming into the perception field, or perceived objects that change physically, etc.). R_θ denotes the absolute variation rate for parameter θ and $l2$ is a negative reinforcement (by default -100). Intuitively, if R_{freqCE} increases (i.e., the environment is more dynamic), then R_θ should decrease proportionally (making the BN more reactive); but if R_{freqCE} decreases, then R_θ should increase in order to make the BN more deliberative.

M_i: This term validates the add-list efficiency of each behavior. If the current state includes a percept that corresponds to any add-list's proposition of any behavior, the behavior will receive a positive reinforcement (remember that the add list represents the expected effects which are produced during behavior execution).

$$M = m1 \cdot \sum_{i=0}^{k} eev, \quad (17)$$

where $m1$ is a positive reinforcement (by default $+100$) and k is the number of propositions defined by the add-list of the activated behavior (a_{beh}). *eev* is a function that determines if the expected effect is included in the current state ($S(t)$), in other words, it validates if the condition $\exists\, p \in S(t) \mid p\ \cap\ a_{beh}\ \neq\ \emptyset$ is true.

N_i: This term validates the delete-list efficiency of each behavior. If the current state includes a percept that corresponds to any delete-list's proposition of any behavior, the behavior will receive a negative reinforcement (remember that the delete list represents the unexpected effects which should not be present during behavior execution).

$$N = n1 \cdot \sum_{i=0}^{k} een \quad (18)$$

where $n1$ is a negative reinforcement (by default -200) and k is the number of propositions defined by the delete-list of the activated behavior (d_{beh}). *een* is a function that determines if the unexpected effect is not included in the current state ($S(t)$), in other words, it validates if the condition $\exists\, p \in S(t) \mid p\ \cap\ d_{beh}\ =\ \emptyset$ is true.

Finally, the whole fitness for each BN is calculated as

$$FFT_i = FFS_i + FFE_i \quad (19)$$

All the elements of the function exert different (and in some cases, opposing) evolutionary and selective pressures. On the one hand, we have defined a function element for each of the most typical structural problems identified in behavior networks (such as terminating and dead-end networks, monotone networks, nonconverging acyclic networks, ambiguous and conflicting links, etc.). On the other hand, we have defined a function element for each kind of functional characterization of the behavior network (such as goal orientedness vs. situation orientedness, bias towards ongoing plans vs. adaptivity, deliberation vs. reactivity, and sensitivity to goal conflicts). So the whole fitness function tries to model a multi-objective problem where the most suitable solution will probably be found at an intermediate point.

4 Plan extraction

Another main contribution that our evolutionary approach makes is the capability to extract plans as HTN-like structures (HTN, hierarchical task networks; Sacerdoti, 1977) as a result of the interaction among behavior networks. In HTN planning, high-level tasks are decomposed into simpler tasks until a sequence of primitive actions solving the high-level tasks is generated. If the decomposition is not possible (e.g., because of colliding restrictions), the planner backtracks and creates a different decomposition. However, in our proposal we don't use a backtracking mechanism to infer a plan (which may be very expensive computationally when the plan grows in size), but we propose a mechanism that discovers a plan as an incremental composition of elements, from low-order building blocks to high-order tasks: *building blocks (symbolic and subsymbolic rules)* $\rightarrow$ *behaviors* $\rightarrow$ *behavior networks* $\rightarrow$ *plans*.[2] Thus, the main contribution we make in comparison with classic planners and methodologies (such as HTN planning) is that our approach does not need a priori definition of axioms, premises, and operators to generate a complex hierarchical plan.

The hierarchical plan is encoded as a linear structure of subtasks, which is evolved by GEP. Thus, a new plan is built and executed as follows:

1. A new plan is generated randomly as a genetic linear structure (this will be explained later).
2. The genetic structure is translated into an expression tree as in Section 3.
3. The expression tree is translated into a HTN-like structure.
4. The HTN-like structure is translated into an ordered tasks sequence as a result of applying the tree-traversal process in postorder.
5. Each subtask of the sequence is executed by a corresponding BN. The activated BN starts the spreading activation dynamics and selects an executable behavior. Each behavior is activated and the corresponding action is executed. The process continues until the associated subgoal is achieved by the BN.
6. After the BN achieves the subgoal, the next subtask is selected as indicated by the linear order of the sequence (plan), and the process restarts in the previous step. The plan execution finalizes when there are no more subtasks to execute.

In order to extract and encode plans, we need to introduce briefly three concepts: *automatically defined functions*, *homeotic genes*, and *cells*. Automatically defined functions (ADFs) were introduced by Koza (1992) as a way of reusing code in genetic programming. An ADF is an independent function or procedure that is invoked by a main program as many times as necessary. The way these ADFs interact with one another and how often they are called upon is encoded in special genes—homeotic genes—thus called because they are the ones controlling the overall development of the individual. And, continuing with the biological analogy, the product of expression of such genes is called a *cell*. Thus, homeotic genes determine which genes are expressed in which cell and how they interact with one another. Stated another way, homeotic genes

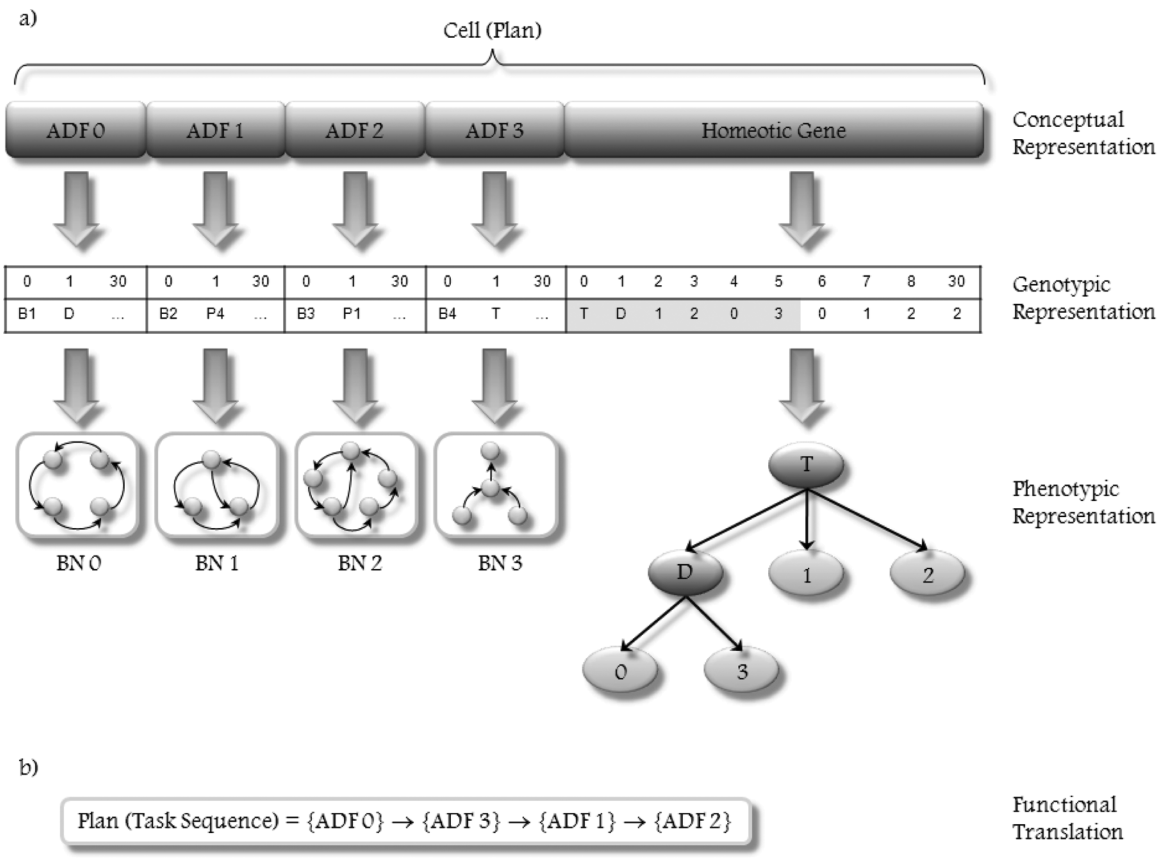

a)
Cell (Plan)
ADF 0
ADF 1
ADF 2
ADF 3
Homeotic Gene
Conceptual Representation
0 1 30 | 0 1 30 | 0 1 30 | 0 1 30 | 0 1 2 3 4 5 6 7 8 30
B1 D ... | B2 P4 ... | B3 P1 ... | B4 T ... | T D 1 2 0 3 0 1 2 2
Genotypic Representation
Phenotypic Representation
BN 0
BN 1
BN 2
BN 3
T
D
1
2
0
3
b)
Plan (Task Sequence) = {ADF 0} → {ADF 3} → {ADF 1} → {ADF 2}
Functional Translation

Figure 2. Plan encoding: (a) GEP cell encoding plans as HTN structures; (b) plan extraction.

determine which ADFs are called upon in which main program and how they interact with one another.

In Figure 2 we present a schematic representation of a plan as a cell structure that is formed by multiple ADFs and one homeotic gene. Each ADF corresponds to one BN with distinctive topology and functionality; in other words, every ADF is a multigenic chromosome that defines a set of behaviors and the spreading activation dynamics between them in order to accomplish a subgoal through the continuous reorientation of the agent's attention process. Thus, every ADF promises to achieve a subtask of the plan.

The homeotic gene builds a hierarchical structure as a HTN-like plan. Therefore, it defines how ADFs (goal-oriented behavior networks), which are functionally independent from each other, interact through cooperative and competitive relationships with each other. In this way, the homeotic gene is able to control which BN is activated in every moment and which one is inhibited while the others cooperate in order to achieve a set of goals and subgoals. As shown in Figure 2, each BN (ADF) has a different topology, different set of goals, and different global parameters, and hence each hierarchical structure plan (cell) is distinctive from other plans (considering a multicellular system).

Homeotic genes have exactly the same kind of structure as conventional genes and are built using an identical process. They also contain a head and a tail domain. In this case, the heads contain linking functions (so called because they are used to link different ADFs) and a special class of terminals—genic terminals—representing conventional genes, which, in the cellular system, encode different ADFs; the tails contain only genic terminals.

It is worth pointing out that homeotic genes have their specific length and their specific set of functions and terminals. In our case, the homeotic gene uses the connectivity functions **D** and **T** (which were explained in Section 3.1) in order to link ADFs (terminals), and each terminal is a node that represents the index of the ADF in the corresponding cell. For instance, terminal "0" invokes "ADF 0," terminal "1" invokes "ADF 1," and so on. Figure 2a depicts a homeotic gene that encodes the following genetic sequence (just the ORF region): **T-D-1-2-0-3**. In Figure 2b, the plan is extracted as an ordered task sequence as a result of applying the tree-traversal process in postorder. Thus, the extracted plan indicates that initially ADF_0 (BN_0) is executed; when it finishes, ADF_3 (BN_3) is executed, then ADF_1 (BN_1), and finally ADF_2 (BN_2).

It is important to note that the adaptive mechanism of these complex and hierarchical plans has an implicit co-evolutionary process: the evolutionary development of each separate decision-making structure (such as behaviors and BNs) affects the building of high-order cognitive structures such as plans. Co-evolution differs from ordinary unimodal evolutionary algorithms in terms of fitness function usage, because the evaluation process is based on interactions between individuals, in our case among BNs. Each BN represents a distinct component of the problem which has to collaborate with the others in order to construct an effective composite solution. In other words, the fitness function is nonstationary, but it is also based on the quality of co-existing individuals representing different problem components (De Jong & Pollack, 2004). Because the fitness measure is specified relative to other BN (as individuals), the improvement of the quality of a partial population triggers further improvements in other populations. Intuitively, from Figure 2 it is also possible to infer that each agent has a set of cells (each one modulating a hierarchical composite plan) which compete with each other for the right to solve a specific problem.

As a result, after a certain number of evolutionary generations, valid and better adapted cells are generated inside the agent. A roulette-wheel method is used to choose the cells with the most likelihood of selection derived from their own fitnesses, and their components' fitnesses (ADFs' fitnesses). The cell's fitness represents how good the interaction with the environment was during agent's lifetime and is partially determined by feedback signals from the environment and the evaluation of a fitness function.

4.1 Fitness function and feedback signal for plans

The evaluation of how good every generated plan is, is driven by two processes: (1) the *internal simulation* of every plan (cell); and (2) the later evaluation of the *fitness function* for every simulated plan.

For the *internal simulation* process we use an *anticipatory system* that provides feedback about how optimal a plan would be if it were executed in the agent's world. In other words, each plan (cell) is tested internally through multiple simulated loops; at each step, an anticipatory system (driven by an *anticipatory classifier system*; Butz, Goldberg, & Stolzmann, 2002; Stolzmann, 1999) predicts the next state of the world that has the highest probability of occurring. In the anticipatory system, the structure of the stimulus–response rules (classifiers) is enhanced by an effect part representing an anticipation of the perceptive consequences of its action on an environment (the agent's world). This effect part, associated with a learning process called the anticipatory learning process (ALP), enables the system to learn latently a complete internal representation of the environment. The ALP represents the application of the anticipatory behavioral control into the system. Therefore, the anticipatory system combines the idea of learning by anticipation with that of the learning classifier systems framework (Booker, Goldberg, & Holland, 1989).

The *fitness function* used to evaluate the performance of each generated plan is based on a function of three terms:

$$FFP_i = \frac{Reinforcement_i - Cost_i}{Duration_i} \tag{20}$$

where FFP_i is the fitness function for the simulated plan i, and the other terms are described as follows.

Duration$_i$: is the total period of time the simulated plan would take if it was executed by the agent in the world. This term is estimated as the sum of all predicted time responses that the agent would take for every action.

$$Duration_i = \sum_{i=0}^{n} AL_i, \tag{21}$$

where n is the number of steps (concrete actions) necessary to achieve the target goal from the current state, and AL is the number of activation loops that the BN takes in every step to select an executable behavior (remember that spreading activation dynamics can execute several loops before choosing an executable behavior). As you can see, the duration function (D_i) is inverse to the reinforcement function (R_i); so, intuitively, the fewer activation loops are necessary to achieve behavior activation, the stronger is the positive feedback signal.

Reinforcement$_i$: is the estimated net amount of feedback that the agent would collect if the plan was executed:

$$Reinforcement_i = \sum_{i=0}^{n} R_{beh,i} + R_{BN,i}, \tag{22}$$

where $R_{beh,i}$ is the estimated net reinforcement that the current activated behavior (selected by the BN in every step i) receives after executing its action in the internal simulation. $R_{BN,i}$ is the estimated net reinforcement that every BN receives after executing the simulated plan.

$R_{beh,i}$ is a prediction that the anticipatory classifier system makes about how much feedback (positive or negative) would be received by the behavior after executing its action. On the other hand, the $R_{BN,i}$ reinforcement corresponds to the evaluation of fitness function for every BN (as discussed for Equation 19).

Cost$_i$: is the estimated cost if the plan was executed, and is defined in terms of the amount of resources and energy that the agent would require for every action.

$$Cost_i = \sum_{i=0}^{n}\sum_{j=0}^{r} En_{i,j}, \tag{23}$$

where n is the number of simulated actions, r is the number of resources (actuators) that the agent would use to carry out the action i, and $En_{i,j}$ is the amount of energy that would be required by the actuator j at action i. For example, if the agent is a robot that has to move boxes from one place to another, the costs required by the robot in every execution step would depend on the number of actuators (e.g., an actuator to control the robot speed, an actuator to control the turn rate, an actuator to control a gripper, etc.) involved in every executed action, and on the amount of energy that would be needed to activate each one of these actuators. Note that the cost function is an optional term for the fitness function and it has to be defined for each problem domain. Therefore, the way these two processes (internal simulation and fitness function) interact with each other in order to validate a new plan is described as follows:

```
1. A set of plans (such as cells formed by ADFs) are generated using GEP operators.
2. For Each plan, do:
   2.1. The agent perceives the current state.
   2.2. While the plan simulation has not achieved the final goal, do:
        2.2.1. The corresponding BN (determined by the sequential order of the plan) is activated.
        2.2.2. The winner behavior, which has been selected by the spreading activation dynamics of the activated BN, is executed.
        2.2.3. The action that proposed the activated behavior is executed into the (internal) simulated environment.
        2.2.4. The statistics described above (duration, cost, and reinforcement) are updated.
        2.2.5. A new prediction of the next state is generated by the anticipatory system.
        2.2.6. If it is possible to make a prediction of the world's next state, the external sensory input is replaced by the prediction and the loop continues with step 2.2.1, otherwise, the simulation stops.
   2.3. End While.
3. End For Each.
4. The overall fitness function for each plan ($FFP$ from Equation 20) is computed (taking into consideration all the feedback signals).
5. The plan with the highest fitness is selected:
      $bestPlan = max_j(FFP_{(j,i)})$        (24)
6. The selected plan is executed by the agent.
```

5 Experimentation

In order to evaluate the proposed evolutionary decision-making model, the following aspects were considered:

1. Convergence rate of evolutionary structural design of BNs.
2. Comparison of convergence rates of evolutionary functional design of BNs using multiple experimental cases.
3. Convergence rate of evolutionary design of plans.

A simulated robotic environment was proposed to test the experiments. In the simulated environment, the robot had to collect different kinds of objects and then deliver them to specific storage boxes. The robot had to coordinate different kinds of tasks such as object search, object recognition, route planning, obstacle avoidance, battery recharging, object piling up, and so forth.

The simulated robotic environment was designed using the Player/Stage platform.[3] Stage simulates a population of mobile robots moving within and sensing a two-dimensional bitmapped environment. Various sensor models are provided, including sonar, scanning laser rangefinder, pan-tilt-zoom camera with color blob detection, and odometry. In our simulation the robot (agent) was provided with four kinds of sensor interfaces: sonar sensors, gps sensor, laser sensors, and fiducial sensors; and two kinds of actuator interfaces: position interface and gripper interface.

The sonar interface provides access to a collection of fixed range sensors, such as a sonar array. The gps interface provides access to an absolute position system, such as GPS. The laser interface provides access to a single-origin scanning range sensor, such as a SICK laser range-finder. The fiducial interface provides access to devices that detect coded fiducials (markers) placed in the environment. These markers work as reference points (fixed points) within the environment to which other objects can be related or which objects can be measured against. The fiducial sensor uses the laser sensor to detect the markers. The position interface is used to control a planar mobile robot base (i.e., it defines the translational and rotational velocities). The gripper interface provides access to a robotic gripper (i.e., it allows the robot to grasp and release objects). Figure 3 shows some of the robot interfaces used.

5.1 Convergence rate of evolutionary structural design of BNs

This experiment shows how the topological structure of BNs can be evolved through the GEP algorithm producing a new set of refined and syntactically well-formed structures. In order to find the adaptation rate

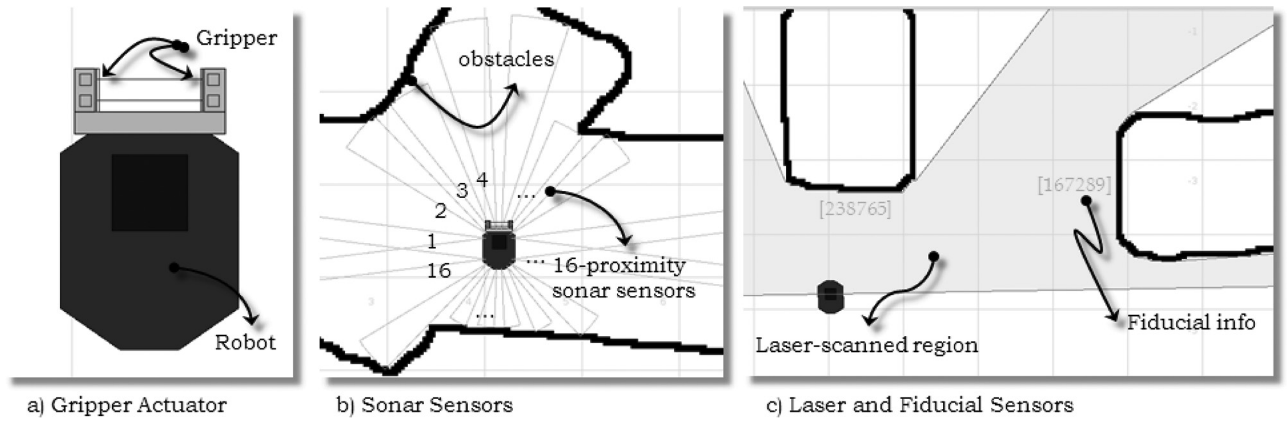


Figure 3. Robot interfaces.

Phenotype translation				
	Behavior	Preconditions	Add-list	Delete-list
Behavioral gene 0: B1-D-P1-P2-P8-P2-P1-P1-P4-P1	LOOK-FOR-STORAGE (B1)	No-Obstacles No-Identified-Storage	Identified-Storage	No-Identified-Storage
Behavioral gene 1: B2-T-P5-D-D-D-P1-P2-P6-P8-P12-P6-P9	LOOK-FOR-BOX (B2)	Identified-Storage No-Obstacles No-Full-Storage No-Identified-Box With-Battery	Identified-Box	No-Identified-Storage No-Identified-Box
Behavioral gene 2: B3-T-P11-D-P5-D-D-P14-P12-P8-P1-P9-P13	PICK-UP-BOX (B3)	Identified-Box No-Obstacles Identified-Storage With-Battery No-Stored-Box	Grasped-Box	No-Grasped-Box No-Stored-Box
Behavioral gene 3: B4-D-M1-P13-D-D-P9-P11-P1-P8	STORE-BOX (B4)	With-Battery Grasped-Box Identified-Storage No-Obstacles	Collect-All-Boxes	No-Stored-Box
Behavioral gene 4: B5-P7-P8-null	AVOID-OBSTACLES (B5)	Obstacles	No-Obstacles	null
Behavioral gene 5: B6-D-P3-P4-P4-P8	LOOK-FOR-BATTERY-CHARGER (B6)	No-Identified-Battery-Charger No-Obstacles	Identified-Battery-Charger	No-Identified-Battery-Charger
Behavioral gene 6: B7-D-D-D-P3-D-P9-M2-P4-P10-P8-P10	RECHARGE-BATTERY (B7)	Identified-Battery-Charger No-Obstacles No-Battery	With-Battery Maintain-Energy	No-Identified-Battery-Charger No-Battery
Functional gene 0: 18-42-87-91-46	π = 19, θ = 43, ϕ = 88, δ = 92, and γ = 47			

of evolved BNs, we proposed a target BN that achieves two global goals of the agent: *Collect-All-Boxes* and *Maintain-Energy*. The main idea of this experiment is that the robot has to collect all the boxes, which are scattered everywhere in the world, and put them down at the corresponding storage, while also avoiding surrounding obstacles and recharging the battery every time that it runs out of energy. The agent has to learn all the relationships between behaviors (activation and inhibition links) in the BN. Figure 4 depicts the world used in the experiment and the target BN that the robot had to discover through the evolutionary process.

We proposed a population of 100 BNs. Each individual BN was represented by a multigenic chromosome that was composed of seven behavioral genes (one for each behavior of the net), where each gene had a length of 30 elements (i.e., head length $=$ 10 and tail length $=$ 20), and one functional gene with a length of 5 elements (one for each global parameter). Thus, the whole chromosome has a length of 215 alleles (i.e., $30 * 7 + 5$). The GEP parameters were defined both through

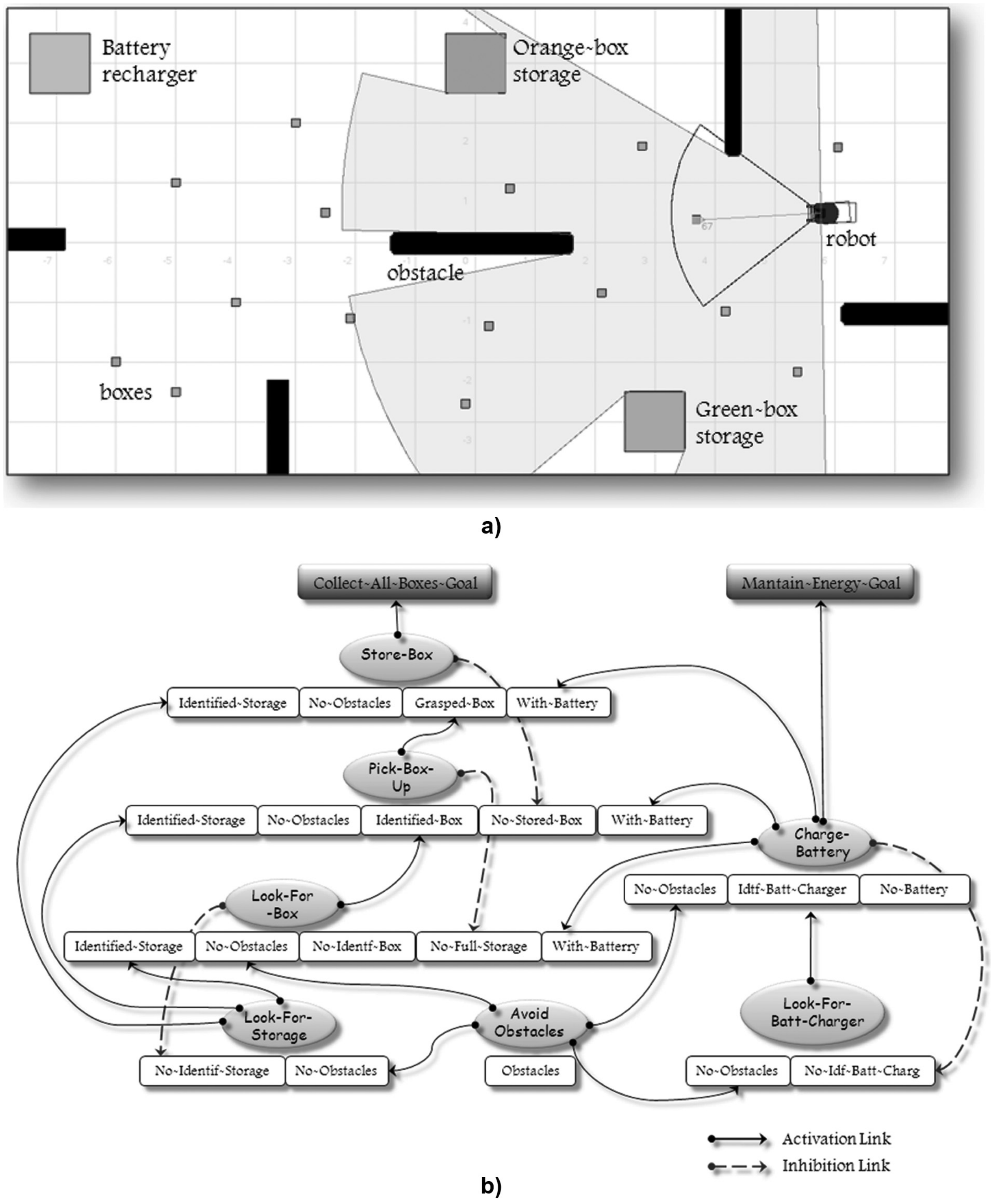


Figure 4. Contextual domain: (a) box-collecting world; (b) target BN for box-collecting task.

empirical adjustments obtained in previous experiments (Romero & de Antonio, 2008, 2009a, 2009b) and on the basis of theoretical ranges (Ferreira, 2006; Koza, 1992; Poli et al., 2008). Table 1 summarizes the GEP parameters used in the experiment.

Figure 5 shows the progression of the mean fitness of the population and the fitness of the best individual over 500 epochs. Note that the results of this experiment were obtained for a harmonic mean value for 100 runs. In other words, the experiment was run 100 times from identical starting data, and in every run it was executed for 500 evolutionary epochs. In every epoch, all the BNs of the population were evaluated using the fitness function in Equation

Table 1. GEP parameters for BN development (evolutionary process).

Parameter	Value	Parameter	Value
Number of runs	100	Mutation rate	0.05
Number of generations (epochs)	500	One-point recombination rate	0.2
Population size	100	Two-point recombination rate	0.4
Number of behavioral genes	7	IS transposition rate	0.2
Number of functional genes	1	RIS transposition rate	0.1
Head length of behavioral genes	10	Global parameter (GP) mutation rate	0.05
Head length of functional genes	5	GP one-point recombination rate	0.25
Chromosome length	215	GP two-point recombination rate	0.45

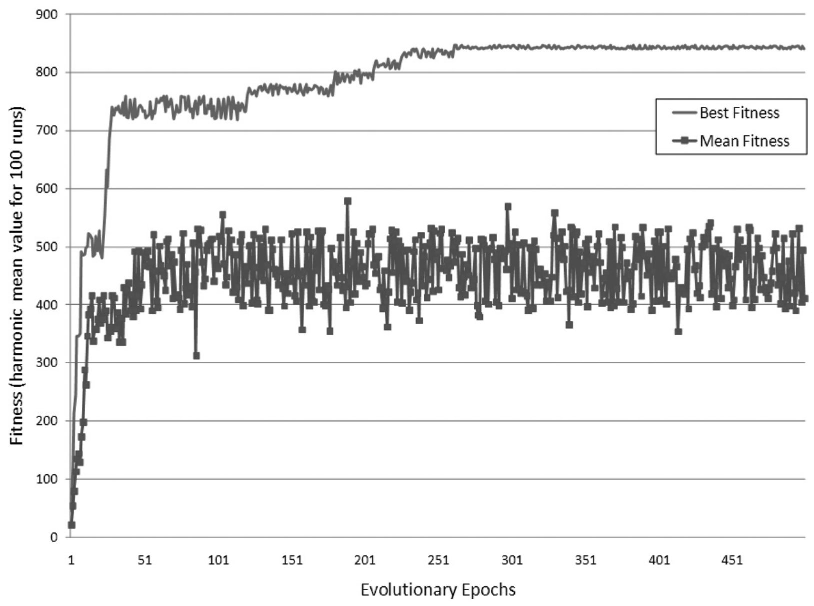


Figure 5. Behavior of the GEP population best and mean fitnesses during the evolutionary epochs.

Table 2. Statistics for BN structural evolution.

	Min. value	Max. value	Average	*SD*	Variance	*MSE*
Mean fitness	21	578	446.87	66.77	4459.14	85.73
Best fitness	45	847	790.28	97.13	9435.87	48.87

20; however, no plan execution was performed in this experiment.

It is worth noting that all the runs for the evolutionary process found a perfect solution (that is, a solution that represented a net topology as shown in Figure 4b) in a range between 259 and 283 epochs. After the system reached a global optimum, it maintained itself in a steady state for all the remaining epochs.

In Figure 5, the graph for mean fitness (i.e., the average fitness for the whole population of BNs) suggests different evolutionary dynamics for GEP populations. The oscillations of mean fitness, even after the discovery of a perfect solution, are unique to GEP. The oscillation is due in part to the small population size used to solve the problem presented in this work, and in part to the degree of diversity that injects both mutation and recombination genetic operators.

Table 2 summarizes the statistical data from this experiment. Mean square error (*MSE*) measures the average of the square of the "error." This error is the amount by which the estimator differs from the quantity to be estimated, and in our case, corresponds to the average number of topologically invalid BNs produced by every evolutionary epoch. From the results, notice that the *MSE* of the best fitness curve is almost 56% lower than that of the mean fitness curve $[1 - (MSE_{best}/MSE_{mean})]$.

The BN structure (chromosome) found by the best solution had the following genotype and phenotype (for simplicity, only the head segment is shown for every gene):

In spite of getting a good solution for the *box-collecting* task, the evolutionary process found some alternative solutions whose genes improved the target solution initially proposed. Nevertheless, these genes were found dispersed in chromosomes that were not totally optimized; therefore, the BN chromosome shown above remained the best solution for the problem because it allowed the system to be in a steady state for almost 250 epochs, meaning that the BN encoded by the best chromosome reached a state where no negative reinforcement was received by the robot.

An interesting example of these alternative solutions is a behavioral gene that included the "No-Battery" proposition in the add-list of STORE-BOX behavior, which means that the robot would probably run out of energy after leaving the grasped box at its corresponding storage. This simple variation may cause the robot to avoid interrupting the execution of STORE-BOX behavior the next time it finds a box (e.g., if the robot decides to release the grasped box and recharge battery instead of storing the box when it runs out of energy, the STORE-BOX behavior would be interrupted). Therefore, this new proposition guarantees that the robot checks the energy level after storing a box, and if it is low the RECHARGE-BATTERY behavior will receive more activation energy from the STORE-BOX behavior; otherwise, the LOOK-FOR-BOX behavior will receive more activation energy. Thus, the robot would always complete the *box-collecting* task successfully because it would never run out of energy between the execution of the PICK-UP-BOX and STORE-BOX behaviors. It is worth noting that this additional proposition, as well as other propositions defined by different genes, were not expected to appear in the target solution, so we can consider them as emergent properties produced by the evolutionary process.

5.2 Comparison of convergence rates of evolutionary functional design of BNs using multiple experimental cases

In order to measure the convergence rates of different aspects of the evolutionary functional design of BNs, we proposed various experimental cases where global parameters were continuously adapted to the situations.

- **Case 1**: this case measures the adaptation rates of the goal-orientedness versus situation-orientedness aspects of BNs. In this experiment, the robot senses one orange box and seven green boxes around it, where `<Collect-All-Orange-Boxes>` is the current goal. In spite of the fact that the robot receives more activation energy from the situation (i.e., the seven green boxes), it must learn to pursue current goals and avoid changes of attention focus (e.g., it must focus on collecting orange boxes instead of collecting green boxes). See Figure 6.
- **Case 2**: this case measures the adaptation rates of the deliberation versus reactivity aspects of BNs. Initially, the robot has to store an observed box in a specific storage and to achieve this it has to gradually accumulate activation energy and sequentially activate a set of behaviors which accomplish the `<Store-Box>` goal (deliberation). During the task execution, an unexpected situation is presented to the robot: some obstacles are dynamically moved around, so the robot has to react with a timely evasive action (reactivity) and retake the control after that. See Figure 7.
- **Case 3**: this case measures the adaptation rates of bias towards the ongoing plans versus adaptivity aspects of BNs. In this experiment the robot has to store a box in a storage situated at a specific point in the environment. The robot has to make a plan in advance in order to achieve the `<Store-Box>` goal. When the robot gets close to the storage, the latter is displaced to another location, so the robot is unable to store the box and has to start looking for the new storage location. The aim of this experiment is to validate the speed of the best evolved BN to replan a new problem-solving strategy in runtime. See Figure 8.
- **Case 4**: this case measures the adaptation rate of the sensitivity to goal conflicts aspect of BNs. In this experiment we take into account the anomalous example situation of the block world (Sussman, 1975). In this classical conflicting-goals example there are three blocks (`A, B, and C`) which must be piled up in a specific order. The initial state of the world is S(0) = `(<clear-B>, <clear-A>, <A-on-C>)` and the goals are `G(0) = (<A-on-B>, <B-on-C>)`. The robot should first achieve the goal `<B-on-C>` and then the goal `<A-on-B>`. It is tempted, however, to immediately stack A onto B which may bring it into a deadlock situation (not wanting to undo the goal already achieved). Some of the behaviors used for this experiment were: `<stack-A-on-B>`, `<stack-B-on-C>`, and `<take-A-from-C>`. See Figure 9.

Figure 10 shows the convergence curves for the experimental cases 1, 2, 3, and 4 using the harmonic mean value for 100 runs. It is important to note that the evolutionary process converged for all cases, although not always at the same speed. The convergence speed for case 4 was slower because the GEP algorithm had to adjust all the global parameters, whereas in the other cases it

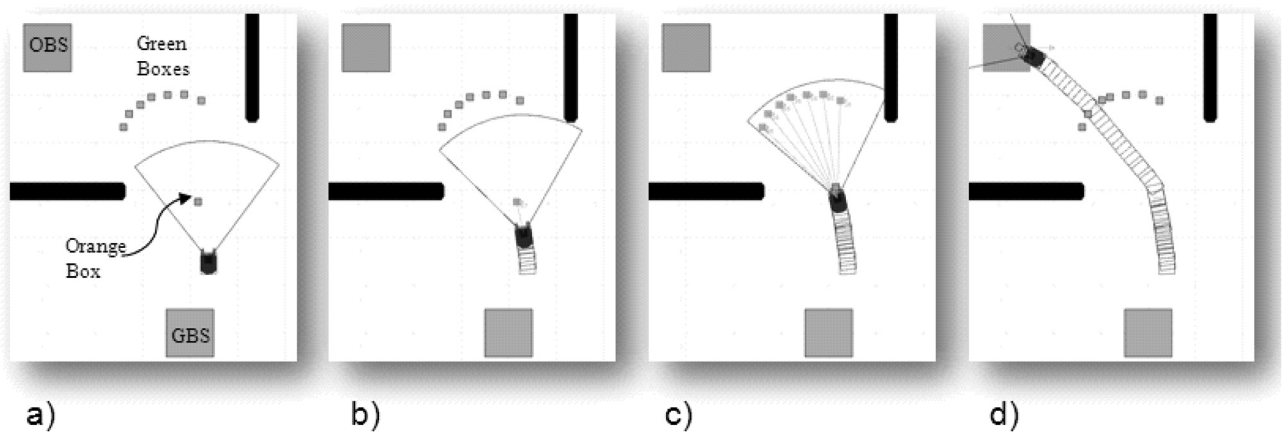


Figure 6. Case 1: (a) the robot senses an orange box; (b) the robot activates <Pick-up-orange-box> behavior; (c) the robot senses other seven “green” boxes; (d) the robot does not change the attention focus even though it receives more activation from the situation (the seven green boxes). GBS = green box storage, OBS = orange box storage.

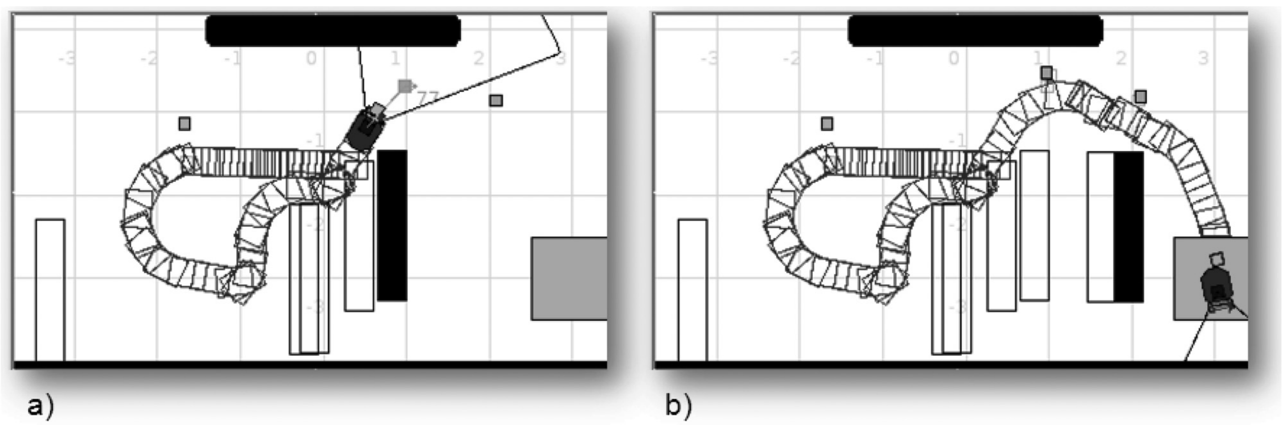


Figure 7. Case 2: (a) the robot reactively avoids a moving obstacle in front of it while it is carrying a box; (b) the robot finally stores the box in spite of multiple distracting obstacles.

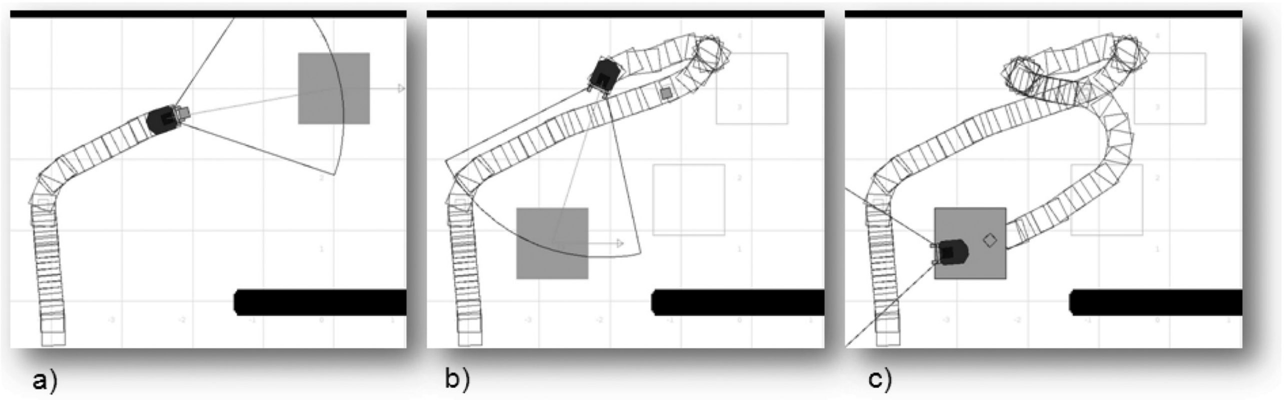


Figure 8. Case 3: (a) the robot is transporting a box to the storage; (b) the location of the storage is moved, so the robot changes its initial plan (dropping the grasped box and starting to look for the new location of the storage); (c) after the robot finds the new location of the storage it retakes the <LOOK-FOR-BOX> behavior and then stores the box.

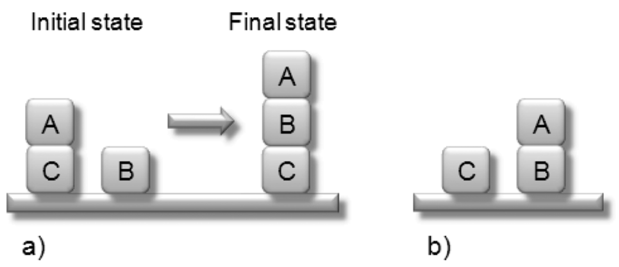


Figure 9. Case 4: (a) initial and final states of block world problem; (b) deadlock situation that avoids undoing the goal already achieved <A-on-B>.

only needed to adjust some parameters: π, γ, and ϕ for case 3; θ, γ, and ϕ for case 2; and γ and ϕ for case 1.

Table 3 shows the statistical data from the experiments. The CE (convergence epoch) column indicates the mean epoch when the algorithm converged in every experiment. The *MSE* is the mean square error for 100 runs of the corresponding experiment, where an error is considered as a wrong behavior activation produced in every execution step by the behavior network (in the MSE_{gep} column the BN with the best fitness generated by the GEP algorithm is used and in the MSE_{maes} column the original Maes BN model without an evolutionary mechanism is used). The *MSE* rate presents the performance relationship between MSE_{gep} and MSE_{maes}. It is important to note that, for the experiments executed, the proposed evolutionary BN model improves the performance results obtained by the original Maes BN model by between 58% and 69%. This improvement is due to the capability of the proposed evolutionary BN model to self-adjust the global parameters in runtime, whereas the original Maes BN model was always restricted to a fixed configuration of these parameters.

For all the experiments, the original Maes BN model was configured with the following fixed global parameters: $\delta = 90$, $\gamma = 50$, $\phi = 90$, $\pi = 90$, and $\theta = 100$. In contrast to these fixed values, Table 4 shows the global parameters discovered by the proposed evolutionary mechanism for each experimental case. From these results arise the following observations:

Case 1: the proposed evolutionary mechanism discovered that in order to keep the balance between the "goal-orientedness" and "situation-orientedness" aspects, γ must be approximately 29–34% greater than ϕ. For a very high value of γ or a very low value of ϕ, the agent (robot) could not adaptively redirect its attention focus towards more interesting goals when they were presented.

On the other hand, a value of ϕ greater than γ would prevent the robot achieving any of the goals because it would be continuously changing its attention focus.

Case 2: the proposed evolutionary mechanism found that in order to keep the balance between the "deliberation" versus "reactivity" aspects, the value of ϕ must be a little higher than γ (approximately 9–15%). With this configuration the agent was not only able to keep its attention focused on current goals, but was also able to react against unexpected or dangerous situations. Additionally, when reactive behavior was required, the proposed evolutionary mechanism discovered that a low value of θ allowed fast behavior activation because the BN took fewer activation loops and, as a consequence, the amount of deliberative processing was considerably decreased.

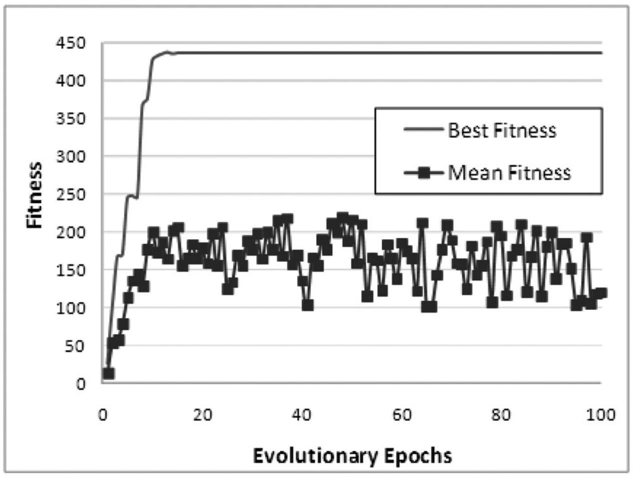
Case 3: the proposed evolutionary mechanism revealed that in order to keep the balance between the "bias towards ongoing plans" versus "adaptivity" aspects, the value of π must be approximately 46–52% greater than γ, and 72–77% less than ϕ. If these global parameters are kept between such ranges, the agent will not continually "jump" from goal to goal and in turn it will be able to adapt to changing situations. From the wrong solutions it is possible to infer that a value of π too much greater than γ and ϕ makes the BN more adaptive, although less biased towards ongoing plans, hence the agent will continually be changing the current goals without keeping the focus on any of them.

Case 4: the proposed evolutionary mechanism discovered that in order to preserve the "sensitivity to goal conflicts," the value of δ must be approximately 51–65% greater than γ; if it is less than 50% greater the BN does not take away enough activation energy from conflicting goals. A value of γ greater than δ causes the BN to go into deadlock due to the inability of the BN to undo goals already achieved. Furthermore, the evolutionary mechanism found that the value of ϕ must be approximately 62–73% greater than δ, otherwise the BN will not be able to activate the behavior sequence that resolves the goal conflict (i.e., <take-A-from-C>, then <stack-B-on-C>, and then <stack-A-on-B>) because it will not receive enough activation from the observed state. Finally, the evolutionary mechanism revealed that in conflicting goal situations the value of θ must be less than δ, otherwise the BN will be more deliberative and, therefore, it will execute more activation loops during which the behaviors that promise to directly achieve a goal (e.g., <stack-B-on-C> and <stack-A-on-B>) will accumulate more activation energy than those behaviors that solve conflictive goals through subgoaling (e.g., <take-A-from-C>).

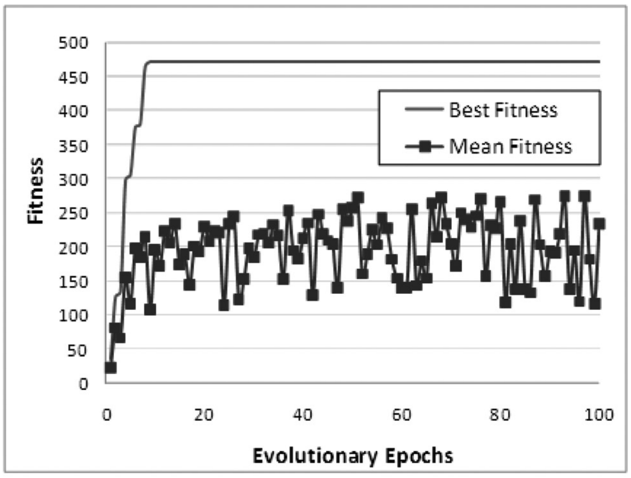
From the results obtained in the above experiments, it is evident that setting global parameters is a "multi-objective" problem, hence a single BN solution cannot define a proper setting for all the proposed experimental cases. So, the solution to this problem requires multiple BNs competing for survival in the population, where only the best BN will be activated according to the situation observed by the agent.

5.3 Convergence rate of evolutionary design of plans

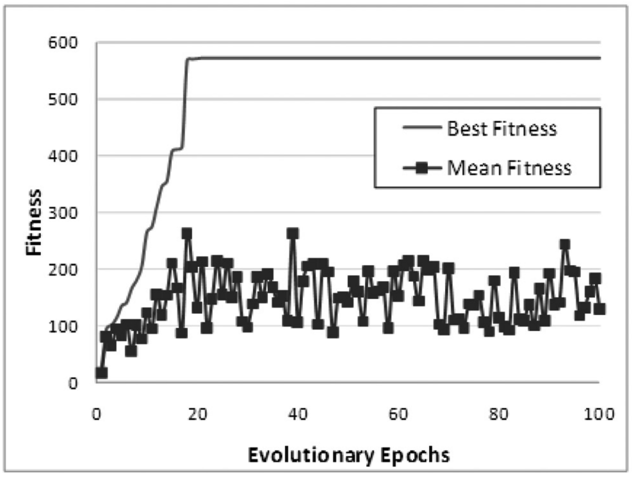
In this experiment we proposed two different tasks that the robot had to accomplish. The first was the



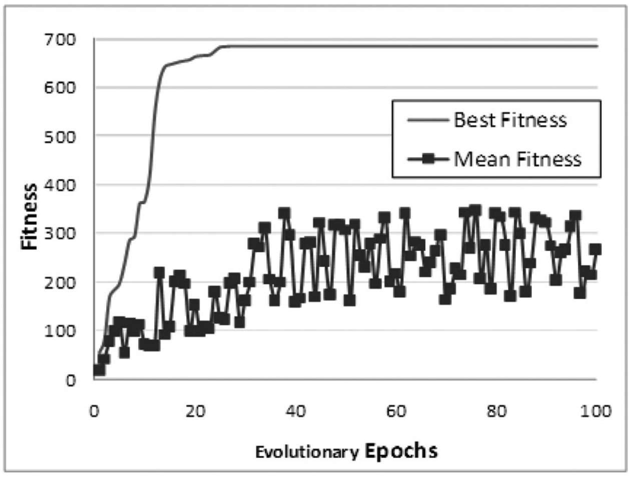
a) Experimental Case 1



b) Experimental Case 2



c) Experimental Case 3



d) Experimental Case 4

Figure 10. Behavior of the GEP population best and mean fitnesses during the evolutionary epochs for Cases 1, 2, 3, and 4.

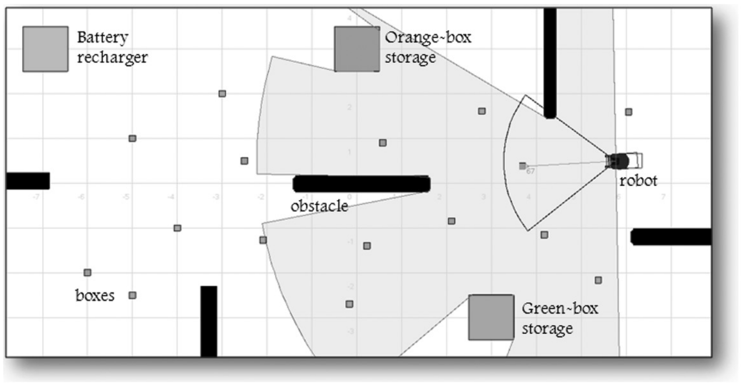
Table 3. Statistics for BN functional evolution. CE is the convergence epoch of the curve. MSE_{gep} is the mean square error of the case using the GEP algorithm. MSE_{maes} is the MSE using only the original Maes BN model. MSE rate is equivalent to $[1 - (MSE_{gep}/MSE_{maes})]$.

	Min.	Max.	Average	*SD*	CE	MSE_{gep}	MSE_{maes}	*MSE* rate
Case 1	27	437	416.10	70.90	15	45.78	118.03	61.21%
Case 2	35	472	455.44	69.10	9	41.39	98.36	57.91%
Case 3	28	573	515.18	137.73	21	56.15	145.87	61.50%
Case 4	53	683	629.01	143.98	27	73.02	233.28	68.69%

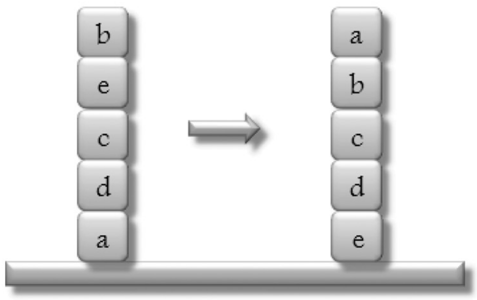
Table 4. Evolved global parameters.

	δ	γ	ϕ	π	θ
Case 1	85	68	23	92	95
Case 2	87	57	64	93	15
Case 3	88	37	61	18	93
Case 4	48	17	63	18	41

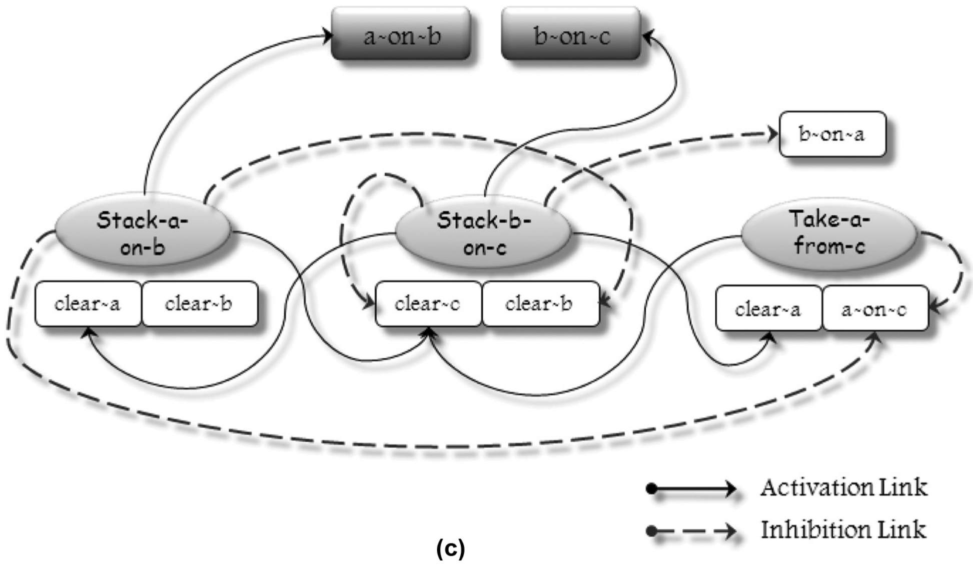
box-collecting task described in Section 5.1, and the second was a task in which the robot had to learn how to stack boxes on top of each other following specific order criteria (very similar to the Hanoi towers problem). The basic idea of the problem is as follows: the robot has to collect different kinds of boxes (orange and green boxes), which are scattered everywhere, and to store them in the corresponding storage. The boxes



(a)



(b)



(c)

Figure 11. Planning problem: (a) box-collecting world; (b) box-stacking sample; (c) partial BN for the box-stacking task.

are piled up one on top of another in the same order that the robot finds them. After all boxes are stored, the robot has to stack, in an ordered manner, all the collected boxes in every storage.

Figure 11a shows the environment used for the experiment. There were five green boxes (a, b, c, d, and e) and six orange boxes (f, g, h, i, j, and k). Figure 11b depicts an example where, given a set of stored boxes that were first stacked in a random order, the robot restacked them, this time following an ordered criteria (a-on-b, b-on-c, and so on).

Finally, Figure 11c depicts a partial segment of the BN for the *box-stacking* task. In our implementation we used variables for the *box-stacking* BN, so the robot was able to reuse the behaviors in different situations. So, for example, "Stack-*x*-on-*y*" behavior receives two variables (*x* and *y*) which can be any of the stored boxes.

The final aim of the experiment was that the robot had to generate a plan whose duration (in terms of execution steps) was optimized. Thus, the robot had to minimize both the total number of execution steps necessary to collect all green and orange boxes, and the number of execution steps to organize and stack (in an ordered manner) all the stored boxes.

It is important to note that both kinds of BNs used in this experiment were previously evolved by the robot with the purpose of focusing just on the planning problem and speeding up the solution convergence. Additionally, before this experiment, the robot performed a training phase where it got specific knowledge about where every box and storage in the world was situated. Table 5 summarizes the GEP parameters used in the evolutionary process for generating plans.

During the evolution of each ADF, the robot generated two BNs with different structural and functional features. That make sense because, on the one hand, the BN for the *box-collecting* task needs: (1) to be more oriented to situations than to goals, that is, the behavior selection must be driven by what is happening in the environment; (2) to be more adaptable rather than biased to ongoing plans, so the robot can exploit new opportunities, be more fault tolerant, and adapt to dynamically changing situations such as the sudden need to recharge the battery because it has run out of energy, or the need

Table 5. GEP parameters for plans development (evolutionary process).

Parameter	Value	Parameter	Value
Number of runs	50	Mutation rate	0.1
Number of generations (epochs)	500	One-point recombination rate	0.3
Population size (plans)	50	Two-point recombination rate	0.5
Number of ADFs	4	IS transposition rate	0.3
Number of homeotic genes	1	RIS transposition rate	0.3
Head length of homeotic gene	10		

Table 6. Global parameters.

Parameter	Value for box-collecting BN	Value for box-stacking BN
π	25	19
θ	87	39
ϕ	53	65
δ	92	46
γ	3	20

to replan because it finds a better solution in runtime; and (3) to be more deliberative (thoughtful) than reactive, because the robot has to plan all its movements towards whichever box and storage before starting moving. On the other hand, the BN for *box-stacking* task requires: (1) to be more oriented to goals than to situation because the environment is not affected by meaningful external changes while the robot is stacking the boxes; (2) to be more biased to ongoing plans rather than adaptable, so the robot can address behavior activation to the ongoing goals and subgoals of the task without a significant interference of external changes; (3) to be more susceptible to avoiding goal and subgoal conflicts through the arbitration of behavior activation; and (4) to be more deliberative than reactive. All of these characteristics were evolved through the runtime adjustment (adaptation) of global parameters, as summarized in Table 6.

Note that in order to avoid very long plan simulations we used a simple criterion that prunes the space of simulations and improves the overall performance. So, we evaluated plans trivially: "at every epoch, all the plans are simulated concurrently using an independent processing thread for each plan; therefore, the first simulated plan to finish must be the route that takes least execution steps and time (duration) to traverse, and all the others could be aborted without loss." Thus, in Figure 12 we only present the curve for the best plan found in every epoch.

Table 7 summarizes the statistical data from this experiment. The error of *MSE* corresponds to every plan whose fitness is negative after being evaluated by Equation 20 in every evolutionary epoch. Note that the best fitness reduces the *MSE* of the mean fitness curve by 64.38% ($[1 - (MSE_{best}/MSE_{mean})]$).

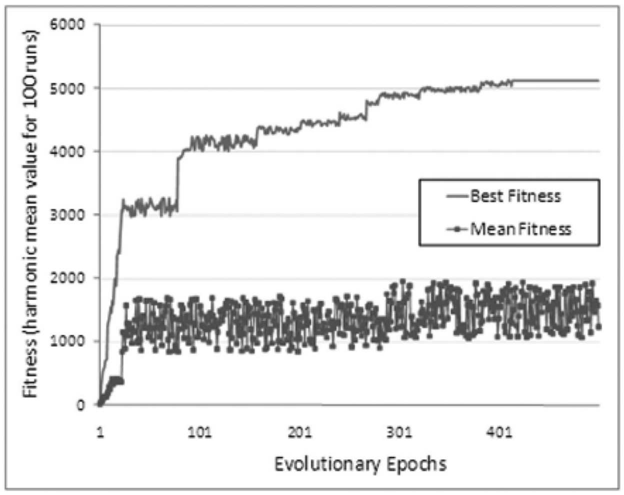
Figure 12 shows the progress in time of the harmonic mean value for 100 runs of the best simulated plan (the one which took least execution steps to find a solution in every epoch). The robot found (on average) an optimized solution after epoch 400. In the plot, the harmonic mean value for the optimized solution converged at epoch 417 and took 576 execution steps. The optimized plan discovered by the evolutionary process is described next.

The homeotic gene (which is responsible for building deliberative plans as ordered-task sequences) uses two function nodes (**T** and **D** connectivity functions) and four terminal nodes that are connected to a specific ADF (ADF_1, green-box-collecting BN; ADF_2, orange-box-collecting BN; ADF_3, stack-green-boxes BN; and ADF_4, stack-orange-boxes BN). Both ADF_1 and ADF_2 receive a parameter in order to focus the spreading activation dynamics of their corresponding BN. For example, if ADF_1 received parameter *a*, it would mean that the BN would pursue the `<store-a-box>` subgoal; if ADF_2 received parameter *f*, it would mean that the BN would pursue the `<store-f-box>` subgoal, and so on. We can extract the corresponding plan as a sequence of ordered tasks obtained after applying the tree-traversal process in postorder on the ET of Figure 13, as follows:

Best Plan: [[ADF_2:k] [ADF_1:a] [ADF_1:d] [ADF_1:e] [ADF_1:c] [ADF_2:j] [ADF_2:f] [ADF_2:h] [ADF_2:g] [ADF_1:b] [ADF_3] [ADF_2:i] [ADF_4]]

Translation:

```
  Task 1:  pick up ``k'' box and store it at
           orange storage
  Task 2:  pick up ``a'' box and store it at
           green storage
  Task 3:  pick up ``d'' box and store it at
           green storage
  Task 4:  pick up ``e'' box and store it at
           green storage
  Task 5:  pick up ``c'' box and store it at
           green storage
  Task 6:  pick up ``j'' box and store it at
           orange storage
  Task 7:  pick up ``f'' box and store it at
           orange storage
```



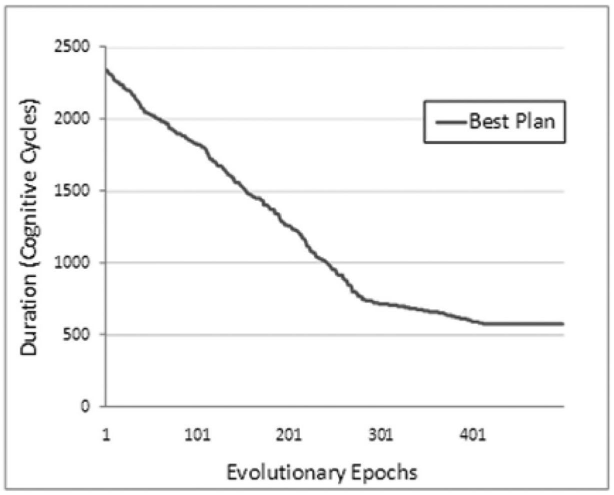


Figure 12. Evolutionary convergence curve for the population of plans: (a) behavior of the GEP population best and mean fitnesses during the evolutionary epochs; (b) best plan convergence curve.

Table 7. Statistics for plan evolution.

	Min. value	Max. value	Average	*SD*	Variance	*MSE*
Mean Fitness	20	1948	1330.98	351.96	123878.54	345.67
Best Fitness	53	5126	4372.07	888.15	788810.76	123,12

```
 Task 8:  pick up ''h'' box and store it at
          orange storage
 Task 9:  pick up ''g'' box and store it at
          orange storage
Task 10:  pick up ''b'' box and store it at
          green storage
Task 11:  stack stored green boxes in a
          downward order
Task 12:  pick up ''i'' box and store it at
          orange storage
Task 13:  stack stored orange boxes in a
          downward order
```

It is important to notice that this structure corresponds to a meta-plan sequence, that is, it defines the global tasks that the robot has to execute in the world, but it does not define the fine-grained tasks and actions because these are determined by every behavior and behavior network during the simulation and/or execution of the plan. For example, between the execution of tasks 3 and 4, the robot activated the *recharge-battery* behavior (and all the corresponding actions) before picking up "e" box which was close to the battery recharger (see Figure 14d). If the robot had decided to recharge the battery afterwards, it would probably have needed to travel a longer distance and, therefore, to perform too many more execution steps. Similarly, the *recharge-battery* behavior was activated before picking up "h" box.

After the best simulated plan was found, the robot executed it on the world. It is worth mentioning that there was not a large difference between the simulated plan and the executed plan: whereas the simulated plan estimated 576 execution steps, the executed plan took 653 real execution steps. This difference was mostly due to obstacle-avoidance actions that were improvised by

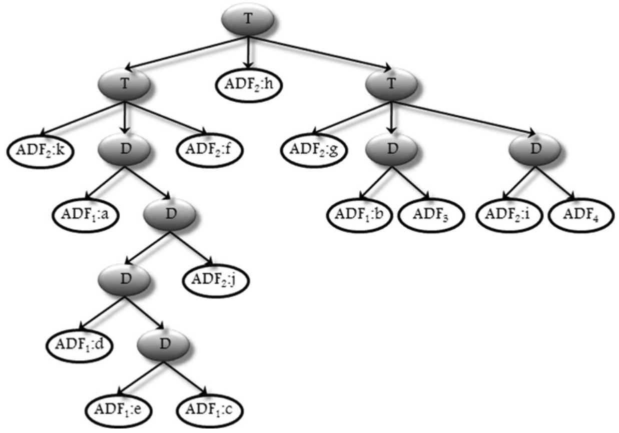


Figure 13. Expression tree for the homeotic gene of the optimized plan. **T** and **D** are connectivity functions. {*a, b, c, d, e*} are terminal nodes that correspond to the green-box-collecting BN (ADF_1). {*f, g, h, i, j, k*} are terminal nodes that correspond to the orange-box-collecting BN (ADF_2). ADF_3 is the stack-green-boxes BN, and ADF_4 is the stack-orange-boxes BN.

the robot along the way. The step-by-step plan executed by the robot is presented in Figure 14.

6 Conclusions

In this article we have described a hybrid decision-making approach for autonomous agents that is supported by the robustness of both the behavior networks model and gene expression programming. The proposed model is able to adaptively build complex decision-making structures as a result of interacting evolutionary dynamics.

Specifically, the proposed evolutionary model focuses on the on-line "development" of both behavior networks as task-oriented decision-making structures, and plans as hierarchical complex decision-making structures. Behavior networks have been proposed in previous works (Franklin, 2006; Maes, 1989; Shanahan & Baars, 2005) as "control mechanisms for selective focusing of attention," and we tried to follow the same philosophy in our approach. However, we argue that this kind of decision-making structure must have the capacity of being adaptable and flexible throughout the agent life cycle, instead of being a pre-wired, hand-coded, and fixed structure. Thus, we proposed an evolutionary mechanism based on GEP which evolves the BNs depending on both the dynamic environmental interactions experienced by the agent and the internal changing states of itself. From the experimentation, it is possible to infer that the evolutionary approach was able to evolve different kinds of BNs. This diversity can be seen from both a structural perspective (e.g., diverse net topologies determined by varying relationships of competition and cooperation among behaviors), and a functional perspective (e.g., different global parameters configurations determine diverse features of BNs such as goal orientedness vs. situation relevance, adaptivity vs. bias to ongoing plans, deliberation vs. reactivity, sensitivity to goal conflicts, etc.).

We also demonstrated how the evolutionary process was able to generate more complex decision-making structures (plans) through the emergent assembly of simpler ones. At this point, it is worth highlighting four main advantages of this mechanism to generate plans in comparison with classic planning solvers: (1) our planning mechanism does not require a priori hand-coded programs that define all the formal semantics of a plan (such as sets of axioms, methods, premises, and operators, large search spaces, complex search algorithms, etc.), as classic planners usually do (e.g., STRIPS and HTN planners); (2) as a consequence of the previous advantage, our evolutionary plan generation model is an emergent process which discovers the best solution in runtime execution, and even though the plan fits into the problem domain, the "planning mechanism" whereby complex plans are built is completely independent from it; (3) the best plan found by the simulation process is closely similar to the real execution of the plan because it predicts (with some degree of accuracy) all the actions that the robot has to execute considering both the current environmental conditions and the dynamic changes that could happen during the performance, and thus it allows the robot to act accordingly; and (4) the generated plan is not a rigid and fixed plan, but it is flexible enough to allow the replanning process (as a consequence of the spreading activation dynamics of the BNs) throughout runtime execution.

In our approach it is possible to identify several levels of planning: (1) short-term planning carried out by the anticipatory classifier system and behaviors, which predicts the outcomes of actions executed in the prior state, so the agent can react opportunely; (2) medium-term planning driven by spreading and accumulation of energy dynamics of BNs, through which a set of expectations, goals, and subgoals are pursued; and (3) long-term planning produced by homeotic genes, which defines the agent acting in terms of high-level task sequences, and which drives the global behavior of the agent towards the achievement of more general (global) goals. All these levels allow the agent to exhibit a deliberative behavior focused on goals achievement, although with the ability to reactively replan and change the course of action when (internal and external) perturbations from state require it. Therefore, it is the system that is in charge of producing its own plans, mentally executing them (through internal simulation), checking against its real world execution and, more importantly, evolving them over time with the feedback received.

7 Future work

In our future work we will continue working on designing a more adaptive and self-configurable decision-making model, incorporating emotional and meta-cognition modules. One concrete application of this research will be the development of a module for emotive pedagogical agents where the agent will be able to self-learn perspectives, beliefs, desires, intentions, emotions, and perceptions about itself and other agents, using the proposed approach.

Currently, our approach uses a basic motivational model based on *drives*, but we hypothesize that using a more sophisticated model (e.g., one that includes affective states, frames of mind, personality features, moods, emotions, etc.), from which new inputs to the behavior networks can be generated, will make the agent's performance more believable. Thus, the GEP process will have to take these new inputs into account in order to address the evolution of better and more complex behavior networks.

In the current approach, the initial set of modules of behavior networks must be defined a priori and the

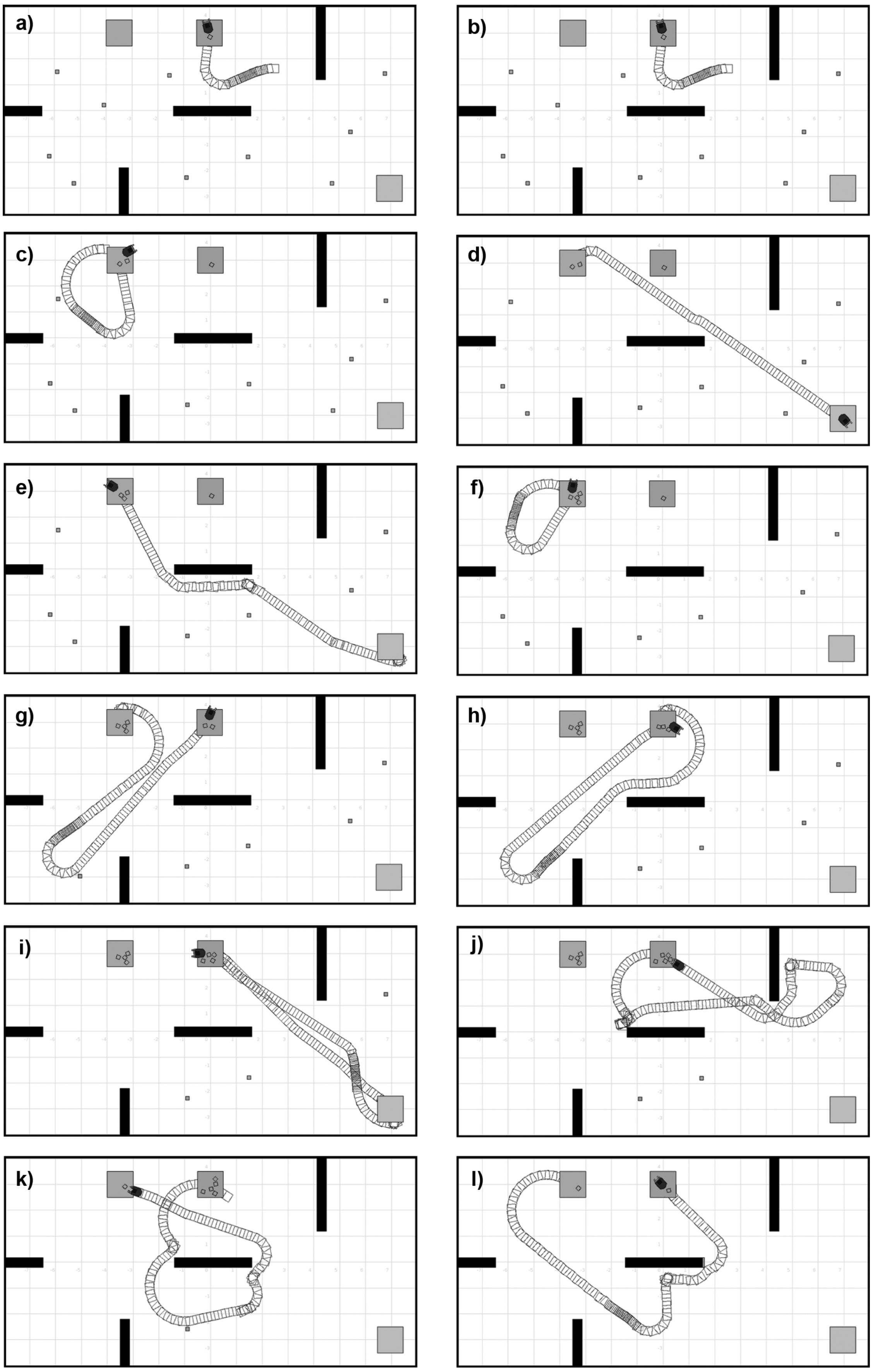


Figure 14. Best plan: (a) collect "k" box; (b) collect "a" box; (c) collect "d" box; (d) recharge battery; (e) collect "e" box; (f) collect "c" box; (g) collect "j" box; (h) collect "f" box; (i) recharge battery and collect "h" box; (j) collect "g" box; (k) collect "b" box and stack green boxes; (l) collect "i" box and stack orange boxes.

GEP process is in charge of connecting them. However, our goal is to find a mechanism that can propose new behaviors from interaction between the agent and its environment, so no pre-definition of this set will be required. After that, this mechanism could refine the inferred behaviors set through generalization and specialization processes, and validating them during execution time. Then, behaviors can be interconnected to other existing behaviors using GEP.

Finally, we want to add a new hierarchical level of knowledge: the *macro-plans*. As we explained in Section 4, plans are encoded as a set of ADFs and one homeotic gene, which constitute a *cell*. We hypothesize that expanding the number of homeotic genes, using a multi-cell approach, will allow the creation of macro-plans. Every homeotic gene will specify different activation conditions of the plan (e.g., if proposition "x" is true then execute plan "A," otherwise, if proposition "b" is true then execute plan "B"…). Therefore, every homeotic gene will encapsulate the execution of a micro-plan, and there will be a master homeotic gene in charge of invoking neither ADFs nor BNs but other homeotic genes.

Notes

1 These processing modules (or behaviors) have been explained in our previous work (Romero & de Antonio, 2008, 2009b) and consist of hybrid components made of production rules, neural networks, and artificial immune systems.
2 Behaviors and rule processing are explained in previous work (Romero & de Antonio, 2008, 2009b).
3 The open source platform is available at http://playerstage.sourceforge.net/.

Acknowledgments

This research was supported by the Fundación Universitaria Konrad Lorenz. This work was carried out in close collaboration with the Intelligent Systems Networks and Robotics Systems (ISN) Laboratory, Imperial College London.

Funding

This work was supported by the Fundación Universitaria Konrad Lorenz.

References

Booker, L., Goldberg, D., & Holland, J. (1989). Classifier systems and genetic algorithms. *Artificial Intelligence*, *40* (1–3), 235–282.

Butz, M. V., Goldberg, D. E., & Stolzmann, W. (2002). The anticipatory classifier system and genetic generalization. *Natural Computing*, *1*, 427–467.

De Jong, E., & Pollack, J. (2004). Ideal evaluation from coevolution, evolutionary computation. *Evolutionary Computation*, *12*, 159–192.

Dorer, K. (1999). Behavior networks for continuous domains using situation-dependent motivations. *Proceedings of the 16th International Joint Conference on Artificial Intelligence* (Vol. *2*, pp. 1233–1238).

Dorer, K. (2004). Extended behavior networks for behavior selection in dynamic and continuous domains. *Proceedings of ECAI-04 Workshop on Agents in Dynamic and Real-Time Environments*.

Ferreira, C. (2000). Genetic representation and genetic neutrality in gene expression programming. *Advances in Complex Systems*, *5*, 389–408.

Ferreira, C. (2001). Gene expression programming: A new adaptive algorithm for solving problems, complex systems. *Cognitive Science*, *13*, 87–129.

Ferreira, C. (2006). Automatically defined functions in gene expression programming. *Computational Intelligence*, *13*, 21–56.

Franklin, S. (2006). The LIDA architecture: Adding new modes of learning to an intelligent, autonomous, software agent. In *Proceedings of the International Conference on Integrated Design and Process Technology*, Cambridge.

Koza, J. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.

Maes, P. (1989). How to do the right thing. *Connection Science Journal*, *1*, 291–323.

Maes, P. (1990). Situated agents can have goals. *Robotics and Autonomous Systems*, *6*, 49–70.

Maes, P. (1991). The agent network architecture (ANA). *SIGART Bulletin*, *2*, 115–120.

Maes, P. (1992). Learning behavior networks from experience. *Proceedings of the 1st European Conference on Artificial Life* (pp. 48–57).

Nebel, B., & Babovich-Lierler, Y. (2004). When are behaviour networks well-behaved? *Proceedings of ECAI* (pp. 672–676).

Poli, R., Langdon, W., & McPhee, N. (2008). *A field guide to genetic programming* (Tech. Rep.). Standford University, MA, USA.

Romero, O., & de Antonio, A. (2008). Bio-inspired cognitive architecture for adaptive agents based on an evolutionary approach. *Adaptive Learning Agents and Multi-Agent Systems, ALAMAS + ALAg* (pp. 68–72).

Romero, O., & de Antonio, A. (2009a). Hybridization of cognitive models using evolutionary strategies. *IEEE Congress on Evolutionary Computation CEC 09* (pp. 89–96).

Romero, O., & de Antonio, A. (2009b). Modulation of multilevel evolutionary strategies for artificial cognition. In. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO* (pp. 279–292).

Sacerdoti, E. (1977). *A structure for plans and behavior*. Amsterdam: Elsevier.

Shanahan, M. P., & Baars, B. (2005). Applying global workspace theory to the frame problem. *Cognition*, *98*, 157–176.

Stolzmann, W. (1999). Latent learning in khepera robots with anticipatory classifier systems. *Proceedings of the 1999 Genetic and Evolutionary Computation Conference Workshop Program* (pp. 290–297).

Sussman, G. (1975). *A computer model of skill acquisition* [Book of doctoral dissertation]. New York: Elsevier.

About the Author

Dr Oscar Romero López has been a researcher since 2009 at Politécnica de Madrid University (Spain), where he works on developing artificial intelligent agents over virtual environments. Since 2010 he has also been a principal investigator at Konrad Lorenz University (Colombia), where he directs the Biologically Inspired Autonomous Agents Team. He received his Ph.D. in intelligent and complex systems at Politécnica de Madrid University in 2010. He was a visiting researcher at Imperial College London, where he researched intelligent robotics.