Evolving Instinctive Behaviour in Resource-Constrained Autonomous Agents Using Grammatical Evolution

Ahmed Hallawa[1], Simon Schug[1], Giovanni Iacca[2], and Gerd Ascheid[1]

[1] Chair for Integrated Signal Processing Systems
RWTH Aachen University, 52056, Aachen, Germany
{hallawa,schug,ascheid}@ice.rwth-aachen.de
[2] Department of Information Engineering and Computer Science,
University of Trento, Povo 38123, Italy
giovanni.iacca@unitn.it

Abstract. Recent developments in the miniaturization of hardware have facilitated the use of robots or mobile sensory agents in many applications such as exploration of GPS-denied, hardly accessible unknown environments. This includes underground resource exploration and water pollution monitoring. One problem in scaling-down robots is that it puts significant emphasis on power consumption due to the limited energy available online. Furthermore, the design of adequate controllers for such agents is challenging as representing the system mathematically is difficult due to complexity. In that regard, *Evolutionary Algorithms* (EA) is a suitable choice for developing the controllers. However, the solution space for evolving those controllers is relatively large because of the wide range of the possible tunable parameters available on the hardware, in addition to the numerous number of objectives which appear on different design levels. A recently-proposed method, dubbed as *Instinct Evolution Scheme* (IES), offered a way to limit the solution space in these cases. This scheme uses *Behavior Trees* (BTs) to represent the robot behaviour in a modular, re-usable and intelligible fashion. In this paper, we improve upon the original IES by using *Grammatical evolution* (GE) to implement a full BT evolution model integratable with IES. A special emphasis is put on minimizing the complexity of the BT generated by GE. To test the scheme, we consider an environment exploration task on a virtual environment. Results show 85% correct reactions to environment stimuli and a decrease in relative complexity to 4.7%. Finally, the evolved BT is represented in an if-else on-chip compatible format.

Keywords: Grammatical Evolution · Behavior Tree · Autonomous Agents.

1 Introduction

Resource-constrained miniaturized autonomous robots (or sensory agents) are becoming available on a wide scale. Their compact dimensions enable flexible usage in a wide range of applications ranging from monitoring of underground

infrastructure [1] and exploration of natural resources, such as oil and gas, to human body diagnostics [2]. However, the available energy on these robots is restricted by their scaled-down size, which puts a special emphasis on the reduction of power consumption. In many cases, the agents are kinetically passive and exposed to noisy and dynamically changing environments, thus requiring robust behaviors capable of dealing with uncertainty [3]. Consequently, sophisticated methods are required in order to develop adaptive behaviors that empower these robots to collaboratively achieve a given set of objectives with such limited hardware resources. Evolvable hardware addresses this problem by autonomously reconfiguring hardware using *Evolutionary Algorithms* (EAs) [4]. Inspired by the biological process of evolution, EAs are a class of meta-heuristics which are well-suited for dealing with complex search spaces. They have proven to be useful in a wide range of applications [5], especially in situations where conventional optimization techniques can not find satisfactory solutions due to a lack of a priori knowledge of the problem under investigation. Broadly speaking, two kinds of approaches have been proposed to apply EA to reconfigurable hardware, namely *extrinsic* and *intrinsic* evolution. In extrinsic evolution, candidate solutions are evaluated using a virtual simulation of the corresponding hardware behavior. Consequently, a static hardware configuration is generated by the algorithm and, subsequently, flashed onto the target device. By contrast, intrinsic evolution refers to approaches that conduct fitness measurements by implementing each candidate solution on the device and observe its behavior. As this eliminates the need for complex simulation environments, the validity of this method is enhanced [6]. However, both extrinsic and intrinsic evolution show some limitations when it comes to the learning process of an agent's behavior. Extrinsic evolution suffers from the inability to adapt the final solution to changes in the problem definition after it has been deployed on the hardware. Though intrinsic evolution is capable to overcome this drawback by continuously adapting the agent, it requires extensive computational power that is not available in miniaturized robots, due to their scaled-down size. In this work we progress on this research area by proposing and verifying a Behaviour Trees (BTs) evolution model integratable with an EA recently proposed in the literature, the Instinctive Evolution Scheme (IES) [7], with the purpose of evolving a behaviour suitable for resource-constrained autonomous robots or sensory agents. The rest of the paper is structured as follows: in Section 2 a background on different behaviour schemes in robotics is given, followed by an introduction to IES. Section 3 introduces the proposed model based on BTs and IES. Finally, Sections 4, 5 and 6 discuss the case study used to verify the model, the results from our experiments, and conclusion respectively.

2 Background

Representing Behaviour: Representation of a behaviour is critical for its development and optimization. In many applications in robotics, *Artificial Neural Networks* (ANNs) are used to encode adaptive behavior. ANNs are capable of

performing complex computational tasks and can easily be combined with EAs. Consequently, there have been numerous successful attempts of evolving robot controllers using ANNs, e.g. [8]. Though ANNs are powerful, they are generally treated as a black box. Understanding and validating the inner workings of a successfully evolved controller, thus becomes a very difficult task. Furthermore, even subtle changes in the specifications of the controller require retraining algorithms as it is unfeasible to manually adapt a completed controller. On the other hand, one alternative is *Finite State Machines* (FSMs). In FSMs, the behavior is abstracted by means of states and transitions that define the response of the agent to a given external output. FSMs lend themselves well for user-defined autonomous behaviors but have also seen successful application with EAs e.g. [9]. Compared to ANNs, they are easier to understand/explain and therefore support the analysis of the behavioral spectrum of an evolved controller. Despite their usefulness in simple action sequences, FSMs quickly become illegible for complex tasks, as the amount of states grows exponentially, a phenomenon termed *state explosion* [10]. Behavior Trees (BTs) represent a promising alternative to FSMs as a mathematical model of plan execution. Based on a hierarchical network of actions and conditions, BTs are capable of describing complex behaviors readily intelligible for users, and allow for easy decomposition and reuse of encapsulated sub-behaviors [11]. Furthermore, there have been attempts of adopting EAs to automatically generate BTs for a given task [12]. Though the general feasibility of the procedure has been shown, several parts of the design process were done manually. As BTs allow for flexible design of actions and conditions specifically tailored to the platform they are deployed on, they do not necessarily require hardware with extensive computational resources. Therefore, they are suited to be used with resource-constrained miniaturized robots.

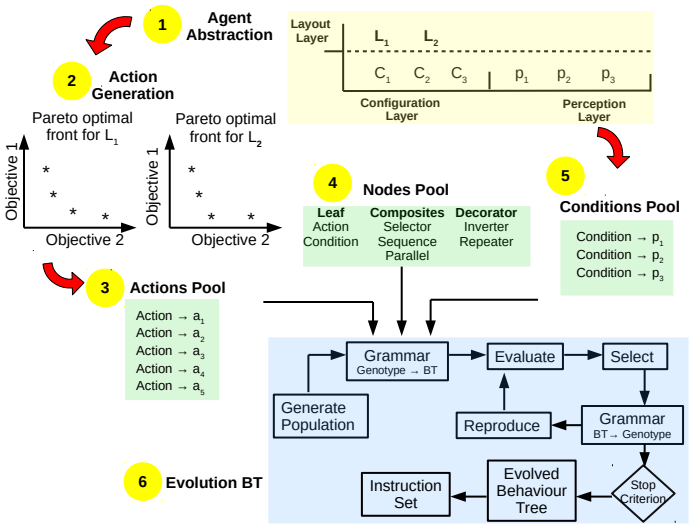


Fig. 1: Instinct Evolution Scheme as proposed by [7]

Instinct Evolution Scheme: The Instinct Evolution Scheme (IES), introduced in [7], attempts to mimic the instinctive behaviour found in nature, where biological entities react to environment stimuli in a fast-reflexive way, with no complex processing. This is adequate for resource constrained robots or sensory agents. The main idea is to use extrinsic evolution, but after minimizing the solution space using "lower-level" objectives. The reduced solution space is then set as the new solution space for the evolution of the behaviour using GE on BTs. In other words, the scheme provides a methodology to limit the solution space for evolving an optimum instinctive behaviour represented by a BT via identifying a set of "interesting" actions and conditions to be used for evolving the behaviour. The scheme consists of six steps as shown in Fig. 1: Firstly, an agent is abstracted as a *layout layer*, then each element in the layout layer (L_1, L_2 ...) includes a set of possible tunable configurations in the *configuration layer* (C_1, C_2 ...) and finally, there is a *perception layer*, which is the set of all perception interfaces with the environment, e.g. sensors. In other words, the layout layer consists of configurable modules that are provided by the hardware architecture of the robot, e.g. a communication module, a compression module ... etc. For each of these modules, a set of configurations are possible, which can be found in the configuration layer, e.g. the Variable Gain Amplifier (VGA) voltage. In step 2, the solution space is minimized, i.e. a multi-objective optimization algorithm is used to identify the Pareto front of the variables in the configuration layer relative to "local" objectives. For example, in case of communication module, the local objectives might be the Signal-to-Noise Ratio (SNR) vs. power consumption. This is done for each element in the layout layer (L_1, L_2 ...) using, for example, R2-indicator based Evolutionary Multi-objective Optimization Algorithm (EMOA). This optimization process will lead to a set of Pareto fronts for each module in the layout layer, relative to its local objectives. In addition, each solution in these Pareto fronts is a configuration, which is "interesting" as it is already dominating other solutions that are not on this Pareto front. In other words, the solution space has shrunk to only solutions that would offer a gain relative to the local objectives for each of the modules in the layout layer. In the next step (step 3), these solutions in the generated Pareto fronts are used to produce a pool dubbed as an *action pool*. This is now the new solution space of possible reactions from the robot to the environment, and it will be used in step 6 to evolve a BT, i.e. all actions in the evolved BT in step 6 are from this action pool. On the other hand, in step 4, a set of possible types of nodes are defined in the *nodes pool*. This basically sets the hardware allowable types of nodes in the BT, e.g. if the hardware allow *parallel* nodes, which can check multiple conditions or actions at the same time, then it is added to the nodes pool. This is done for all types of possible nodes, e.g. invert nodes, repeaters nodes ... etc. In step 5, the *perception pool* is generated. This is done via defining all possible sensors (or any points of interaction between the robot and the environment). Then, possible conditions to be checked for each of these sensors are set. Obviously, in an unknown environment, defining exactly what are the conditions to be checked is not possible, however, if the robot swarm conduct one real experiment in the

environment before the evolution of BT, the data extracted can be statistically studied and a set of "interesting" conditions for each sensor in the perception layer can be defined.This paper is based on IES, hence, in our work the objective is to evolve a BT given an action, condition and nodes pools.

3 Methodology

Implementation of Behavior Trees: The literature offers multiple implementations of BT, such as in [13, 14]. The main differences are mainly in the execution logic and the possible node types. These two factors play an important role in the performance of any BT. As mentioned in Section 2, the BTs' main strengths are their modularity and re-usability. These are possible because all node types used in a BT return one of the three possible states: *Success*, *Failure* and *Running*. Furthermore, a *tick* function is used as an intermediate layer that selects the appropriate routine based on the node type which has been called. In our work, four possible nodes were implemented: selector node, sequence node, action node and condition node. These nodes are sufficient to achieve a wide range of behaviors. Furthermore, a parsing function was implemented to produce an instruction set using if-else conditions only, which is more hardware-friendly. And as stated earlier, this work is based on the IES described in Section 2. Accordingly, the actions pool is considered as an input to the GE process presented in this work. Furthermore, it is assumed that the number of available sensors is also given (the perception layer). For the conditions pool, each given sensor is fed a set of environment data during the initialization of the virtual environment used in the evolution process of the BT, then thresholds are defined by $threshold = M_i \pm n \cdot \sigma_i$, based on the statistical analysis of the sensed data by the virtual environment. A condition node in the evolved BT then checks whether the current sensor readings are located within the range defined by two neighboring thresholds. The resulting condition pool for the ith sensor can be seen in Table 1. Finally, to facilitate the BT evaluation, each executed BT has an action log and condition log, where all actions and conditions that were triggered at all time steps are recorded and later used for its evaluation.

Condition 1	$x < M_i - \sigma_i$
Condition 2	$M_i - \sigma_i \leq x < M_i$
Condition 3	$M_i \leq x < M_i + \sigma_i$
Condition 4	$M_i + \sigma_i \leq x$

Table 1: Condition pool for the ith sensor

Representation in a Context-Free Grammar: A central task of incorporating BTs into the framework of GE is to find a grammatical representation that can be used in the genotype-to-phenotype mapping. Therefore, a formal language needs to be abstracted from the syntax and semantics of BTs transforming the two-dimensional hierarchical structure of a BT into a one-dimensional

string representation. The final language L_{BT} was derived based on the alphabet $\Sigma_{BT} = \{a, c, s, q, \langle, \rangle, \circ, 0, 1, ..., 9\}$ (See Table 2).

Notation	Meaning
s	Selector node
q	Sequence node
a_i	Action node with the identifier i
c_k	Condition node with the identifier k
$\langle \quad \rangle$	Encapsulation of a subtree
$\circ$	Separation of two nodes/subtrees with the same parent

Table 2: Symbols of the language L_{BT} designed for BTs description

In the next step, we develop a context-free grammar $G_{BT} = (V, \Sigma_{BT}, P, S)$ capable of describing the language L_{BT}. Given the variables $V = \{A, C, N, T\}$ and the start symbol $S = \{< N > \langle < T > \circ < T > \rangle\}$ the following production rules P were created:

$$\begin{aligned} < T > ::= \; & a < A > \;\mid\; < T > \circ < T > \;\mid \\ & c < C > \;\mid\; < N > \langle < T > \circ < T > \rangle \end{aligned} \tag{1}$$

$$< N > ::= \; s \mid q \tag{2}$$

$$< A > ::= \; 1 \mid 2 \mid ... \mid \mathcal{N}_A \tag{3}$$

$$< C > ::= \; 1 \mid 2 \mid ... \mid \mathcal{N}_C \tag{4}$$

The variable $< T >$ represents a subtree, while the variable $< N >$ is a placeholder for a composite node that controls the flow of its respective subtree. Composite nodes are required to have at least two child nodes. Hereby the useless nesting of single composite nodes within each other is prohibited. Hence, the start symbol already includes two subtrees for the root node and further composite nodes can only be inserted with two respective subtrees. Furthermore, the variables $< A >$ and $< C >$ can be replaced by a number representing their unique identifier, where $\mathcal{N}_A$ is the number of possible actions and $\mathcal{N}_C$ is the number of possible conditions to choose from.

Selection: Early tests showed that roulette selection method suffered from premature convergence, thus rank-based selection and tournament selections were implemented instead. Rank-based offered a better solution than the roulette, however, while the tournament selection did not add any further benefits despite its relative higher computational costs. Therefore, rank-based selection was adopted in the implementation. Furthermore, *elitism* was also implemented.

Crossover: After two parents have been chosen by rank-based selection, they are subject to crossover with the probability $p_{crossover}$, where $p_{crossover}$ denotes the *crossover rate*. The well established one-point crossover is implemented as it represents the *de facto* standard for GE and has proven to provide good performance in different applications [15]. For each parent, a random crossover point is drawn from the discrete uniform distribution $\mathcal{U}\{2, k_{end,i} - 1\}$, where $k_{end,i}$ denotes the end of the coding region of the genotype of parent i. If the

length of the resulting genotype exceeds $l_{genotype}$, the excess codons are discarded and the last codon is replaced by the escape character '0' to properly define the end of the coding region. If, on the other hand, the length of the resulting genotype underruns $l_{genotype}$, the missing codons are generated from the discrete uniform distribution $\mathcal{U}\{1, c_{max}\}$, similarly to the initialization process. As the one-point crossover is applied to the genotype without access to phenotypic information, it does not respect the tree structure of the phenotype. As a result, changes on the genotype with one-point crossover can lead to drastic differences in the phenotypic appearance. As an alternative, a more sophisticated crossover operation has been implemented that allows for the protection of the integrity of the BT by exchanging arbitrary subtrees between the parents. Moreover, to realize a more general procedure that can be applied to arbitrary grammars, the extension to GE proposed by [16] can be used. In this regard, two new variables $< X >$ and $< Y >$ are introduced to the grammar marking the beginning and ending of a subtree. Consequently, the extended grammar $G_{BT+} = (\tilde{V}, \Sigma_{BT}, \tilde{P}, \tilde{S})$ with the variables $\tilde{V} = \{A, C, N, T, X, Y\}$ and the start symbol $\tilde{S} = \{< N > \langle < X >< T >< Y > \circ < X >< T >< Y > \rangle\}$ is created. The production rules $\tilde{P}$ were extended to denote possible crossover points encapsulating subtrees:

$$
\begin{aligned}
< T > ::= \; & < X >< T >< Y > \circ < X >< T >< Y > \; \mid \\
& < N > \langle < X >< T >< Y > \circ < X >< T >< Y > \rangle \; \mid \\
& a < A > \; \mid \; c < C > \qquad (5)\\
< N > ::= \; & s \mid q \qquad (6)\\
< A > ::= \; & 1 \mid 2 \mid \ldots \mid \mathcal{N}_A \qquad (7)\\
< C > ::= \; & 1 \mid 2 \mid \ldots \mid \mathcal{N}_C \qquad (8)
\end{aligned}
$$

The crossover points for each individual can then be generated during the parsing process. When the parser encounters $< X >$ during execution it will not use the currently active codon. Instead, it creates a new row in the $N \times 2$ matrix $XOSites$ that is kept for each individual. It stores the current position within the genotype in the first column and adds a placeholder value in the second column. Afterwards the variable $< X >$ is removed from the phenotype and the parsing process continues as usual. When $< Y >$ is encountered the last row of $XOSites$ which contains a placeholder value in the second column is searched. The placeholder value is then replaced by the index one before the current position within the genotype and the variable $< Y >$ is removed from the phenotype. For a valid BT, each row of $XOSites$ denotes a section of the genotype that can be exchanged without disrupting the integrity of the BT. The crossover operation can then randomly pick a pair of crossover points from the $XOSites$ matrix of each parent and exchange the respective sections of the gene to create the genotypes for the two children.

Mutation: Mutation is realized by iterating through the genotype of an individual until the escape character '0' is reached. With the predefined probability $p_{mutation}$, commonly referred to as the *mutation rate*, a single codon is replaced

by a new codon drawn from the discrete uniform distribution $\mathcal{U}\{1, c_{max}\}$. Thus, only the non-coding region of the genotype is subject to mutation.

Fitness Function: The fitness function can be split into two components: the first component includes the primary objective that should be solved by the BT. Naturally, it is dependent on the problem domain and varies in its concrete implementation. This part can get computationally expensive, as each BT in the population needs to be executed for a certain amount of time within the simulated environment in order to evaluate the embodied behavior. The second component of the fitness function deals with the complexity of the BT. As seen in [12] evolving BTs often results in cluttered trees with a high amount of redundancy. In this regard, the number of certain node types in the BT are calculated. This can be done by counting the number of occurrences of the corresponding symbol in the phenotype. Thus, the complexity measurement is set as follows:

$$C_i = \frac{n_i}{N_{max,i}} \ \forall \ i \in \{1, 2, 3\} \tag{9}$$

where C_i is the complexity with regard to node type i, n_i is the number of nodes of type i present in the BT and $N_{max,i}$ is the maximum amount of nodes of type i that can be encoded within a single genotype of size $l_{genotype}$. The subindeces denote the node types considered in this work: 1 stands for action node, 2 for condition node and 3 for composite nodes. Consequently, it holds $C_i \in [0, 1)$. The maximum amount of nodes of a certain type strongly depends on the structure of the production rules of the grammar. For a given genotype length, it can be approximated by determining the number of codons that are typically needed to create a certain node type:

$$N_{max,1} = N_{max,2} = \frac{l_{genotype}}{4} \tag{10}$$

$$N_{max,3} = \frac{l_{genotype}}{2} \tag{11}$$

For $N_{max,1}$ and $N_{max,2}$ it is assumed that one codon is needed in order to create a new subtree $< T >$ by applying the rule $< T > ::= < T > \circ < T >$. Additionally, three codons are necessary to generate an action or condition node and its respective node identifier. Furthermore, for $N_{max,3}$, typically only two codons are needed. First a general composite node with two child subtrees is created with one codon by applying the rule $< T > ::= < N > \langle < T > \circ < T > \rangle$. Subsequently, another codon is used to replace $< N >$ and decide on the type of the composite node. As this sequence cannot generate leaf nodes, $N_{max,3}$ represents an upper bound.

4 Case Study

To be able to test the proposed GE framework, a virtual environment inspired by a real-world problem is set such as in Figure 2 . Typically, in a real-world case of exploring a GPS-denied, hardly accessible unknown environment, a swarm of

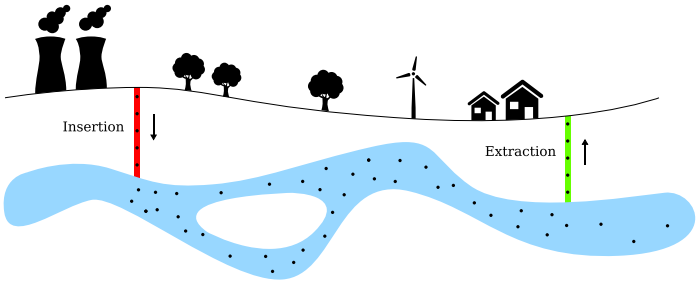


Fig. 2: GPS-denied, hardly accessible environment scenario

robots or sensory agents are injected from one point to be later extracted from another point. Due to their limited resources, the agents need to minimize their power consumption while simultaneously guaranteeing that the collected data obtained from the environment are sufficient to identify the environmental properties of interest. The robot will under go all five steps in IES, thus we assume that all pools of IES are already generated, because the focus of this work is the evolution of the behaviour tree given these pools (Step 6 in IES). The objective here is to evolve a BT given a virtual environment. To achieve this, the environmental properties which the agents are exposed to are created artificially, this will help us to define deterministically identifiable zones, and thus test the ability of the evolved behaviour to correctly identify a zone. Furthermore, in order to assess how good an evolved BT is, each environment zone introduced artificially has a score for each of the n possible actions available in the action pool. In addition, to make the test more challenging, the environment zones where designed such that it is not possible to identify a zone using only a single sensor, or using all sensors (i.e. some sensor readings are redundant). And since running the controller itself involves computational costs, we set a special emphasis on decreasing the complexity of the behaviour tree representation on the robot.

Therefore, to evaluate an evolved BT, firstly the BT is executed within the simulation of a virtual environment, and an action log is created, which contains the identifiers of the invoked actions for each time step. Afterwards, the action log is analyzed in order to rate the quality of the embodied behavior of an agent by comparing it to the predefined optimum solution, and the distance to that optimal solution. The latter is realized by calculating the Hamming distances d_i for each zone i between the identifier of the action invoked by an agent and the corresponding optimum action identifier defined by the virtual environment. i.e. each possible action from the action pool has a fitness distance from the optimal action for each zone. Furthermore, as mentioned earlier the complexity of the BT is important to consider, BTs with more nodes are penalized. This is achieved by incorporating the weighted complexity with regard to different node

types into the fitness function as follows:

$$F = D - \underbrace{\Theta \cdot \sum_{i=1}^{3} \phi_i \cdot C_i}_{\mathrm{C}} \tag{12}$$

where F is the fitness, D can be interpreted as the relative quality of the embodied behavior of an agent and $D \in [0, 1]$ holds. Finally, $\Theta, \phi_i \in [0, 1]$ are weighting coefficients: Θ specifies the extent of the penalty on complex individuals. $\phi_i \in [0, 1]$ are weighting coefficients determining the portion of the individual complexity measurements on the combined complexity measurement C. Therefore, it must hold $\sum_{i=1}^{3} \phi_i = 1$. The values for ϕ_1, ϕ_2, ϕ_3 should reflect the costs of the node types, e.g. action or condition or composite. In this work action nodes are regarded as the most resource demanding, followed by condition nodes, while composite nodes have the smallest impact on performance. Accordingly, the weighting coefficients were set to $\phi_1 = 0.5$, $\phi_2 = 0.3$ and $\phi_3 = 0.2$.

5 Results

Scenario A: 4 Zones: The first tests investigating the ability of the GE to create adaptive BTs in a zone identification task were conducted with 4 zones and 6 sensors. For each zone two properties have been defined that can be detected with two separate sensors. As a property is always shared by two zones, two condition nodes need to be checked in order to unambiguously identify an individual zone.

Parameter	Value
Population Size ($N_{population}$)	100
Mutation Rate ($p_{mutation}$)	0.1
Crossover Rate ($p_{crossover}$)	0.1
Complexity Penalty (Θ)	0.5
Genotype Size ($l_{genotype}$)	300
Number of Zones (n)	4

(a) Parameters

	Zone 1	Zone 2	Zone 3	Zone 4
Sensor 1	1	1	-1	-1
Sensor 2	-1	-1	1	1
Sensor 3	1	-1	1	-1
Sensor 4	-1	1	-1	1
Sensor 5	0	0	0	0
Sensor 6	0	0	0	0

(b) Distribution of sensor properties

Table 3: Settings for the GE with 4 zones and 6 sensors.

Scenario B: 8 Zones: To investigate the ability of the proposed scheme to evolve adaptive BTs, a zone identification task were conducted on an environment with 8 zones and with 8 available sensors on the robot. As shown in Table 4b, for each zone, four properties have been defined that can be detected with four separate sensors out of the eight available ones. Since two environment properties are always shared by two zones, at least two conditions are needed to unambiguously identify an individual zone, which is challenging for the grammatical evolution scheme. Furthermore, four sensors are redundant in the zone identification, requires the GE to check only the relevant ones and discard the rest. The used settings are summed up in Table 4a. Fig. 3a presents the average identification performance over 25 runs. Moreover, the average complexity of the

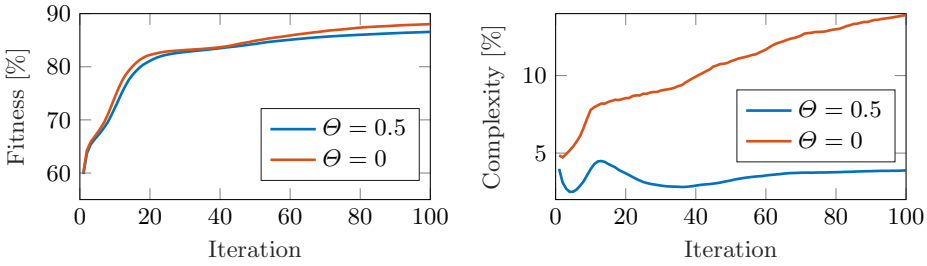
Parameter	Value
Population Size	200
Mutation Rate	0.1
Crossover Rate	0.9
Complexity Penalty	0.5
Genotype Size	300
Number of Zones	8

(a) GE tuning parameters

	Z1	Z2	Z3	Z4	Z5	Z6	Z7	Z8
Sensor 1	1	1	-1	-1	0	0	0	0
Sensor 2	-1	-1	1	1	0	0	0	0
Sensor 3	1	-1	1	-1	0	0	0	0
Sensor 4	-1	1	-1	1	0	0	0	0
Sensor 5	0	0	0	0	1	1	-1	-1
Sensor 6	0	0	0	0	-1	-1	1	1
Sensor 7	0	0	0	0	1	-1	1	-1
Sensor 8	0	0	0	0	-1	1	-1	1

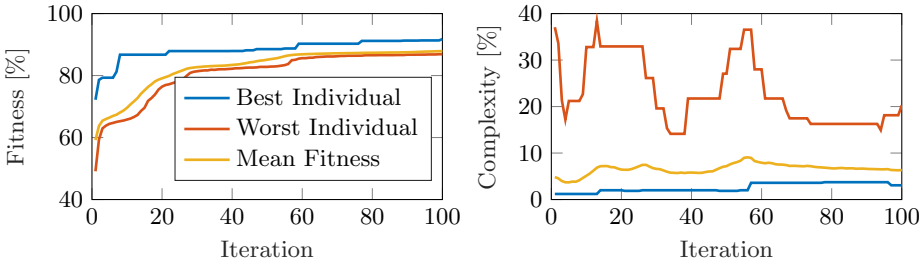
(b) Virtual environment zone properties

Table 4: Settings for the GE with 8 zones and 8 sensors.



(a) Zone Identification (b) Complexity

Fig. 3: Fitness convergence over 25 runs for 8 zones and 8 sensors.

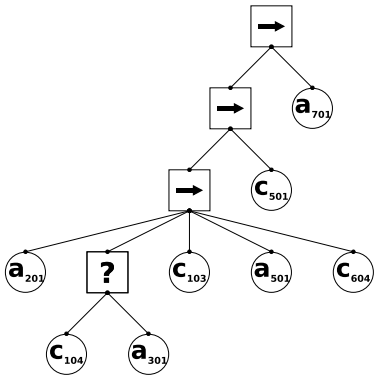


(a) Zone Identification (b) Complexity

Fig. 4: Fitness convergence over a single run for 8 zones and 8 sensors.

BTs shown in 3b, as shown, it increases considerably when compared with the one influenced with complexity penalty. Moreover, looking at a single run of the GE for the 8 zones scenario as shown in Fig. 4, the zone identification task performance shows a consistent convergence throughout the whole run, while the complexity of the underlying BTs is subject to substantial changes indicating a proper exploration of more complex trees regardless of the involved penalty. Moreover, the corresponding BT is shown in Fig. 6a, in total 4 composite nodes, 4 condition nodes and 4 action nodes are used by the tree. Note that it would

require 8 actions to perfectly identify each zone. Finally, Fig. 6b shows the generated if-else control flow.



(a) Behavior Tree

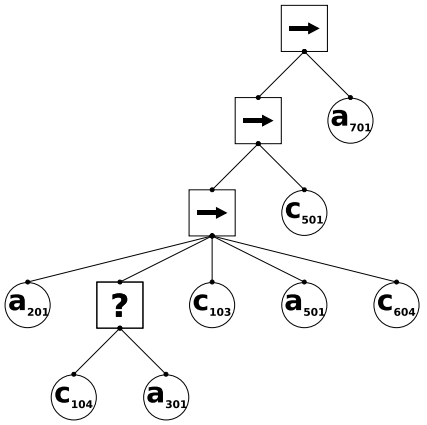
```
1  state = action(201);
2  if state != FAILURE
     then
3  |  state = condition(104);
4  |  if state == FAILURE
   |   then
5  |  |  state =
   |  |   action(301);
6  |  end
7  end
8  if state != FAILURE
     then
9  |  state = condition(103);
10 end
11 if state != FAILURE
     then
12 |  state = action(501);
13 end
14 if state != FAILURE
     then
15 |  state = condition(604);
16 end
17 if state != FAILURE
     then
18 |  state = condition(501);
19 end
20 if state != FAILURE
     then
21 |  state = action(701);
22 end
```

(b) Control Flow

Zone	1	2	3	4	5	6	7	8
Actual Solution	201	201	301	301	501	501	701	701
Optimum Solution	101	201	301	401	501	601	701	801

(c) Action Log

Fig. 5: Best BT solution for 8 zones environment and 8 sensors.



(a) Best Evolved BT for Scenario 8 Zones

```
1  state = action(201);
2  if state != FAILURE then
3  |  state = condition(104);
4  |  if state == FAILURE
   |   then
5  |  |  state = action(301);
6  |  end
7  end
8  if state != FAILURE then
9  |  state = condition(103);
10 end
11 if state != FAILURE then
12 |  state = action(501);
13 end
14 if state != FAILURE then
15 |  state = condition(604);
16 end
17 if state != FAILURE then
18 |  state = condition(501);
19 end
20 if state != FAILURE then
21 |  state = action(701);
22 end
```

(b) Control Flow of 6a

Fig. 6: Best Evolved BTs and Corresponding Control Flow

6 Conclusion

In this work, a Grammatical Evolution (GE) learning model for Behavior Trees (BTs) was successfully integrated into the Instinct Evolution Scheme (IES), thereby providing an offline framework for evolving an online behavior to resource-constrained autonomous robots. A special emphasis was put on minimizing the complexity of the evolved BTs. Furthermore, tests on virtually developed environments showed the effectiveness of the used fitness function and grammar.

References

1. Stoianov, I., Nachman, L., Madden, S., Tokmouline, T.: PIPENET A wireless sensor network for pipeline monitoring. Proceedings of the 6th international conference on Information processing in sensor networks - IPSN '07 (2007) 264
2. Nelson, B.J., Kaliakatsos, I.K., Abbott, J.J.: Microrobots for Minimally Invasive Medicine. Annual Review of Biomedical Engineering **12**(1) (2010) 55–85

3. Fister, I., Strnad, D., Yang, X.s., Jr, I.F.: Adaptation and Hybridization in Computational Intelligence. Volume 18. Springer (2015)
4. Higuchi, T., Liu, Y., Yao, X.: Evolvable Hardware. Springer Science & Business Media (2006)
5. Mattiussi, C., Floreano, D.: Analog genetic encoding for the evolution of circuits and networks. IEEE Transactions on Evolutionary Computation **11**(5) (2007)
6. Glackin, B., Maguire, L.P., McGinnity, T.M.: Intrinsic and extrinsic implementation of a bio-inspired hardware system. Information Sciences **161**(1-2) (2004)
7. Hallawa, A., De Roose, J., Andraud, M., Verhelst, M., Ascheid, G.: Instinct-driven dynamic hardware reconfiguration: Evolutionary algorithm optimized compression for autonomous sensory agents. In: Proceedings of the 2017 Annual Conference on Genetic and Evolutionary Computation, ACM (2017)
8. Pintér-Bartha, Á., Sobe, A., Elmenreich, W.: Towards the Light - Comparing Evolved Neural Network Controllers and Finite State Machine Controllers. 10th International Workshop on Intellegent Solutions in Embedded Systems **0** (2012)
9. König, L., Mostaghim, S., Schmeck, H.: Decentralized evolution of robotic behavior using finite state machines. International Journal of Intelligent Computing and Cybernetics **2**(4) (2009) 695–723
10. Valmari, A.: The state explosion problem. Lectures on Petri Nets I: Basic Models, Lecture Notes in Computer Science Volume 1491 **1491** (1998) 429–528
11. Colledanchise, M., Ogren, P.: How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture, and Decision Trees. IEEE Transactions on Robotics **33**(2) (2017) 372–389
12. Nicolau, M., Perez-Liebana, D., O'Neill, M., Brabazon, A.: Evolutionary Behavior Tree Approaches for Navigating Platform Games. IEEE Transactions on Computational Intelligence and AI in Games **PP**(99) (2016) 1
13. Bagnell, J.A., Cavalcanti, F., Cui, L., Galluzzo, T., Hebert, M., Kazemi, M., Klingensmith, M., Libby, J., Liu, T.Y., Pollard, N., Pivtoraiko, M., Valois, J.S., Zhu, R.: An integrated system for autonomous robotics manipulation. IEEE International Conference on Intelligent Robots and Systems (2012) 2955–2962
14. Marzinotto, A., Colledanchise, M., Smith, C., Ögren, P.: Towards a unified behavior trees framework for robot control. In: Robotics and Automation (ICRA), 2014 IEEE International Conference on, IEEE (2014) 5420–5427
15. Keijzer, M., Ryan, C., O'Neill, M., Cattolico, M., Babovic, V.: Ripple Crossover in Genetic Programming. In: Proceedings of the 4th European Conference on Genetic Programming, - EuroGP, Lake Como, 18-20, April, 2001. Volume 2038. (2001)
16. Nicolau, M., Dempsey, I.: Introducing Grammar Based Extensions for Grammatical Evolution. Proceedings of the 2006 IEEE Congress on Evolutionary Computation (April) (2006) 2663–2670