Autonomous Skills Buildings and Re-Use of Software in Robotics

Lorenzo Riano and T. M. McGinnity
Intelligent Systems Research Centre
University of Ulster
Londonderry, BT48 7JL, UK
l.riano@ulster.ac.uk, tm.mcginnity@ulster.ac.uk

I. Introduction

A close look at the literature in robotics reveals that robots are increasingly being provided with sophisticated skills. Many of these skills are coded in programs or routines freely available to researchers and engineers alike, so that they can build more sophisticated systems. However, in spite of the huge array of skills available, the creation of truly effective autonomous robotics systems still evade researchers.

When faced with the task of designing a robot that solves a particular problem, a roboticist has to answer the following questions:

- Which skills are needed and how to combine them.
- How skills should be modified to work in cooperation with others.
- Which skills are not available and need to be created.

In this paper we propose to address these problems in a autonomous way and within the same framework. We rely on the assumption that many low and high level actions can be reliably performed by a robot using *ad-hoc* algorithms. These include for example object detection, motion planning and grasping. The role of our proposed algorithm is therefore to i) structure and organise the execution of available actions, ii) adapt these actions to solve a particular problem, iii) create new actions when necessary. This process can be carried on in a hierarchical fashion.

In our proposed approach an action is performed by a Finite State Automaton (FSA, plural automata) [1], whose nodes represent skills that are externally provided to the robot (or previously created actions) and whose transitions are the outcomes of the actions. Each action can have a set of parameters. The FSA are instantiated by a an evolutionary process [2] that simultaneously evolves the topology of the FSA and the parameters of the actions. Given the particular problem that we aim to solve we have been required to devise new evolutionary operators.

The proposed approach does not depend on a particular implementation of an action. Therefore if a new action is provided that performs better than an old one, the corresponding node in the evolved FSA can be replaced with the new action without impairing the functionalities of the FSA. Reuse of components thus becomes an important advantage of our proposed algorithm.

We have conducted several experiments, both on a real robot and on a simulator, and we conducted extensive testing to prove that the obtained FSA can reliably drive the robot to solve the problem for which they were evolved. The code for the all the experiments is available on-line[1].

II. Techniques

A. Finite State Automaton

Given our particular application, our formulation of the FSA differs from the classic one adopted for example in [1]. We define an FSA as a quadruple (A, O, δ, s), where:

- A is a finite, non-empty set of actions.
- O is a finite set of outcomes. Each outcome j of state a_i is denoted by $out_j[a_i]$.
- $\delta : A \times O \to A$ is the transition function.
- s is the initial state.

In addition to the above, all the states have a (potentially empty) set of real-valued parameters and they share a common memory where they can read and write data. The role of the shared memory is to share information so as to enable passing of data between actions.

B. Evolutionary Algorithm

The evolutionary algorithm we used follows the general standard structure described for example in [2]. The implementation made use of the library PyEvolve described in [3].

C. Genome Representation

The genome is represented as a directed graph $G = (V, E)$ with parallel edges, where V is the set of the nodes and E is the set of edges. A node $v_i \in V$ is associated with a single action type $a_j \in A$, and it has a (possibly empty) list of real-valued parameters $0 \leq \alpha_i \leq 1$. The meaning of the parameters is action-specific. As every action a_j has a fixed number of outcomes, every node will have a specific number of outgoing edges, each of them representing the specific outcome of an action. In addition to the nodes and edges, the genome encodes the FSA starting state. There is no restriction on the action type the node can be associated to, and several nodes can have the same action type.

[1] https://github.com/lorenzoriano/Graph-Evolve

D. Genetic Operators

Mutation happens both at the graph-level and at the node level. At the graph level new nodes can be added or deleted, and the graph structure changes accordingly. A the node level outgoing edges could change their destination node, paramters are mutated or the whole node is recreated with a new associated action.

Crossover between two graphs is perfomed by selecting two random subgraphs from each parent and swapping them, ensuring that in the resulting offsprings all the nodes are reachable from the starting state. Following the idea that nodes which are close in the graph are likely to be working together, the two subgraphs are chosen starting from a random initial node and selecting neighbours using breadth first search.

E. Simulator

Our approach requires a simulator to evolve a FSA. However a simulator only needs to represent high-level actions and their effect, rather than the physics of the robot and its interactions with the environment. For example we deal with a grasping action in a high-level way: the result of *grasp(object_i)* is simply "*object_i is in the robot gripper*", without caring about the low-level details of the grasping itself. The main assumption behind this work is that several actions are already implemented, including high level ones like perception, grasping, navigation and localisation. If a better grasping algorithm is provided, the old one can be replaced with the new one without affecting the simulations or any previously evolved FSA.

III. Experiments

We tested our proposed system in several experiments, both in simulation and on a real mobile manipulator PR2 robot. Here we show the results we obtained in two main experiments.

A. Moving to Grasp

Many robotics applications require the robot to be able to manipulate objects. The approach we use to grasp an object [4] works only if it is reachable by the robot. However during our experiments we found that an object is often hard to reach, even if it is close to the robot. This is due to physical constraints of the robot's arms that are not easy to analytically model. The main goal of this experiment is to show how our proposed approach can generate novel actions when the available ones are not sufficient to solve a problem. Our generic action is represented by a fully recursive neural network with fixed topology [5] whose weights are represented by the parameters of the associated node in the FSA.

The evolutionary algorithm took 1633 generations to converge to a solution. The resulting FSA has been tested in 395 random starting locations. We only allowed the object to be not more than 1.5 meters away from the robot and not behind it. The robot was able to reach and push the object 323 times out of the 395 tests, thus obtaining a success rate of about 82%.

B. Stacking Objects

The sceond experiment's goal is to show how previously evolved actions can be used in a new evolutionary configuration and to study the interplay between parameters of different actions. We devised a scenario where a robot is facing two objects and it has to stack one over the other. There are no restrictions on which object the robot is allowed to take, as long as at the end of the trial one object is positioned over the other.

Our proposed algorithm obtained a successful FSA after 603 generations. We tested the FSA in 20 different scenarios, with objects of different shapes and positions. The robot successfully executed the stacking action in all the scenarios. In one scenario the robot successfully placed a bottle inside a bag. This proves that the evolved FSA can deal with scenarios for which it was not evolved.

IV. Conclusions and Future Work

The main contributions of this work are:

- A framework that combines structuring, adapting and creating new actions to solve robotics problems.
- A new evolutionary algorithm to evolve both the topology and the parameters of FSA.

We performed experiments to prove that our proposed evolutionary algorithm has good performance in a variety of scenarios. Moreover we proved that, although the evolutionary process is performed in simulation, the results have been straightforwardly applied to a real robot. We have included videos to document the experiments and the source code for the evolutionary algorithm is available on-line.

V. Acknowledgements

This research has been supported by the Centre of Excellence in Intelligent Systems (CoEIS) project, funded by Northern Ireland Integrated Development Fund and InvestNI.

References

[1] J. Hopcroft, R. Motwani, and J. Ullman, *Introduction to automata theory, languages, and computation*. Addison-wesley Reading, MA, 1979.

[2] T. Bäck, *Evolutionary algorithms in theory and practice*. Oxford University Press New York, 1996.

[3] C. S. Perone, "Pyevolve: a Python open-source framework for genetic algorithms," *SIGEVOlution*, vol. 4, no. 1, pp. 12–20, 2009.

[4] K. Hsiao, S. Chitta, M. Ciocarlie, and E. Jones, "Contact-reactive grasping of objects with partial shape information," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, 2010, pp. 1228–1235.

[5] R. D. Beer and J. C. Gallagher, "Evolving Dynamical Neural Networks for Adaptive Behavior," *Adaptive Behavior*, vol. 1, no. 1, pp. 91–122, June 1992.