A Robot that Autonomously Improves Skills by Evolving Computational Graphs

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

***Abstract*—We propose an evolutionary algorithm to autonomously improve the performances of a robotics skill. The algorithm extends a previously proposed graphical evolutionary skills building approach to allow a robot to autonomously collect use cases where a skill fails and use them to improve the skill. Here we define a computational graph as a generic model to hierarchically represent skills and to modify them. The computational graph makes use of embedded neural networks to create generic skills. We tested our proposed algorithm on a real robot implementing a "move to reach" action. Four experiments show the evolution of the computational graph as it is adapted to solve increasingly complex problems.**

I. Introduction

A robot is usually equipped with skills that allows it to solve tasks it is designed for. Although efforts are made to ensure a robot is capable of applying a skill to different environments, no guarantee can be given that a skill will yield positive results in any situation. Autonomously learning and improving skills is therefore a thriving area of research in robotics.

Improving a skill means modifying it so that the robot can solve the same problem more efficiently, or it can solve instances of the same problem it could not solve before. For example a vacuum cleaner robot could improve its cleaning skill so that it will be more efficient in houses with many staircases, or a laundry folding robot can learn to fold more and more types of clothes. Once a robot managed to improve a skill, the newly acquired knowledge can be re-used by other robots to solve a similar task.

Autonomous skills improving faces several challenges. Among them, in this work we address the following questions:

- What is a generic representation of skill?
- How can a skill be improved in one area without impairing its performances in another?

To answer the first question we define in section IV-A a computational graph as a generic representation of a sequence of actions carried out by a skill. A computational graph is a graph where nodes represent actions, or routines externally provided by researchers and programmers. In addition to these routines a node can represent a neural network whose output influences the other nodes and whose parameters are created during the graph creation phase. Edges between nodes represent transitions between actions.

Skills improvement is performed by the evolutionary algorithm we recently proposed in [1]. Here we extended the previous work to allow computational graphs to improve the performances of the associated skills. Constraints are applied so that any newly created skill is at least as capable as the previous one. We describe the main steps involved in skills improvement in section III.

A computational graph represents a generic computational unit. Actions are described in terms of what is their effect on the robot's perception of the world. For example, moving a robot will change the perceived position of objects in the environment or grasping an item will ensure that the item is in the robot's gripper. By abstracting away implementation details of the particular actions a computational graph can be directly or with little modification applied to a variety of robots, provided that they share the same capabilities.

To test our proposed approach we conducted several experiments on a real robot to improve a "move to pre-grasp position" skill. These experiments are described in section VI. A mobile robot is required to grasp an unreachable object. To this aim it has to move the base so that it will be able to reach the object while navigating around obstacles. We started with a simple hand-coded routine that allows the robot only to grasp objects positioned in front of it. We then showed in three more experiments how this skill is autonomously improved so that the robot can reach objects in harder-to-reach locations.

We conclude this paper with a discussion of the results in section VII and we draw the conclusions in section VIII.

II. Related Work

This paper is an extension of the work in [1], where a novel evolutionary algorithm was proposed that allowed the creation and parameterization of new skills by re-using existing software. We extended this work with a novel approach to skills improving.

Splitting the execution of a program into sub-routines is common practice in complex systems, including robotics. One of the best known earlier example has been Brook's subsumption architecture [2], where simple reactive modules called behaviours are executed in parallel to generate an emergent robot's behaviour. This idea has been lately extended to include reasoning and planning in the hierarchical architectures proposed by [3]. A different approach is proposed in [4], where an high-fidelity simulator is used to plan in advance the robot's actions. In section V we show that, in order to improve a skill, a simple transition model is sufficient.

An approach similar to ours has been proposed in [5]. Here a reinforcement learning scheme is used to build and improve skills as the agent interacts with the environment. However no steps are taken to avoid newly acquired knowledge to impair an old skill. We believe this is a crucial feature required by a self-improving robot, as we illustrate in the next section.

In [6] the authors proposed an active logic based architecture that allows a robot to reason about its own failures and to deduce the necessary steps to avoid repeating the same mistake. Although this work is very promising, it is still in its early stage and it requires a proper experimental basis to validate the proposed approach. An extension is in [7], where a reinforcement learning agent learns to adapt to changes in the environment.

Although we co-evolve the computational graph topology with the embedded neural networks, our approach to neuro-evolution is simple compared to the mainstream literature (see for example [8] for a review). The neural networks embedded in our graph are simple computational units that allow an evolved skill to adapt to novel stimuli. Therefore we did not find necessary to employ advanced neuro-evolution techniques like [9], [10], [11]. We are however investigating how graphs evolution can benefit from these approaches.

A common problem in robotics is where to move the robot base so that a manipulator can reach an object. Even if we ignore the potential obstacles around the robot, the non-linearity of the robot actuators and manipulators render this problem very challenging [12], [13]. We used this problem as a testbed for our proposed approach, as described in section VI.

III. Overview of the Proposed Approach

Skill improving is performed by continuously testing a starting skill in an environment, collecting use cases where this skill succeeds and where it fails, and using these use cases to generate a new skill capable of dealing with scenarios it couldn't solve before. This is summarised in algorithm 1.

In this work we will use generic percepts $p_t \in \Re^n$. A percept is a possibly elaborated sensorial input the robot receives at time t. We consider, without loss of generality, a percept to be a n sized real numbers vector. Examples of percepts are the distance between the robot and nearby obstacles, the robot odometric position and orientation, or virtual percepts like the position of objects of interest in the world. We describe the percepts we will use in this work in section VI.

The robot starts with a skill S_0, a classifier C and a transition model Γ. A robot's skill maps a robot's percept $p_t \in \Re^n$ to a motor action $u_t \in U$ for $t \in \{1..T\}$, and its goal is to solve a given problem. A skill therefore is a sequence of motor actions that unfolds over T time steps, and it can terminate with either "success" or "failure". The classifier $C : \Re^n \rightarrow \{0..1\}$ produces, given a robot's perceptual input p_t, the probability $C(p_t)$ that the skill will terminate with success. The transition model $\Gamma : \Re^n \times U \rightarrow \Re^n$ predicts what the next percept will be given the current percept and a motor action. Together S_0, C and Γ describe the outcomes of a skill and how close is the robot to solve a given task. Our proposed algorithm will make extensive use of these models to build new and better skills.

A skill S_1 uses two previously generated sets W_{S_0} and M_{S_0} of scenarios where the skill successfully solved a task and where it did not. Each element $p_i \in W \cup M$ is a snapshot of the robot's perceptions before performing the skill. Given the model described above an element $p_t \in W_{S_0}$ is such that $C(p_t) > 0.5$, while an element $p_t \in M_{S_0}$ will have $C(p_t) \leq 0.5$. We can therefore express the goal of improving the skill S_0 as generating a new skill S_1 such that the constraints in eq. 1 are satisfied.

$$\begin{cases} \forall p \in W_{S_0}\ S_1 \text{ succeeds.} & \text{(1a)} \\ \exists p \in M_{S_0}\ S_1 \text{ succeeds.} & \text{(1b)} \end{cases}$$

Eq. 1a states that for every situation (initial percept) where S_0 succeeds, S_1 has to succeed too. This avoids forgetting previously acquired knowledge. Eq 1b states that the new skill has to solve at least one instance of the problem where S_0 previously failed.

Once a new improved skill S_1 is created the same process is used to create a better skill S_2, and any subsequent other. This creates a monotonically increasing sets series $W_{S_n} \subset W_{S_{n-1}} \subset \ldots \subset W_{S_0}$ of percepts.

Algorithm 1 The skills improvement loop. See text for details.

```
Require: Initial skill $S_0$, transition model $\Gamma$, classifier $C$.
1: $i \leftarrow 0$
2: Test the skill $S_i$ and collect the sets $W_{S_i}$ and $M_{S_i}$.
3: loop
4:     Generate a new skill $S_{i+1}$ using the evolutionary algorithm in Algorithm 2.
5:     if the constraints in eq 1 are met then
6:         $i \leftarrow i + 1$
7:         Goto line 2
8:     end if
9: end loop
```

Skill improvement is performed by using the evolutionary algorithm described in section IV. We describe the classifier C and the transition model Γ in section V.

IV. Evolving Computational Graphs

A. Computational Graph

A skill is composed by a sequence of actions linked to form a directed graph with parallel edges (see Figure 1). Every action can be either an externally provided software routine or a previously generated skill. The latter case represents a hierarchical composition of skills. Given the composable nature of skills, in this section we will refer to a node in the graph with the term "action", meaning either an hand-coded program or a previously acquired skill. Every action a_i has a set of outcomes $out_j[a_i]$ that represent the result of applying

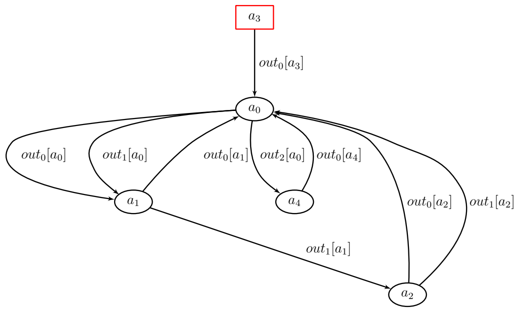

Fig. 1. An example computational graph, represented as a directed graph with parallel edges. Every node is associated to an action. The red squared node is the starting node. To the right of every edge is its label, i.e. an outcome of an action.

that action. Typical outcomes are "success" or "failure". An outcome $out_j[a_i] = a_k$ means that, after executing a_i, if the outcome is $out_j[a_i]$ then the computation switches to the next action a_k. In the graph this is represented as a directed edge between nodes a_i and a_j, labeled with $out_j[a_i]$. All the outgoing edges of a node must be connected to another node. The only exception is for outcomes that trigger the end of the computation, either because the problem has been solved or because the skill failed.

Every action a_i has a set $\Upsilon = \{\alpha^{(i)} \in \mathbb{R}^{n_i}\}$ of n_i real-valued parameters that influence the behaviour of the action. These parameters will be co-evolved with the graph structure, as described in section IV-B. The semantic of an action's parameters is specific to the action: they could represent fine-tuning values or weights in a neural network, as we will see in section VI. One node in the graph will be the starting node. Nodes can exchange data by either channeling messages over the graph's edges or storing values in a common memory shared by all the nodes.

The structure described above represents a **computational graph** (CG) and it is a generic way to represent skills in terms of sub-routines, their outcomes and the parameters that govern them. The computation starts from the starting node, which is executed and its outcome is registered. According to this outcome and the connections in the graph the computation moves to the next action (which could be the same node if the edge is looping). The computation continues until a terminal transition is produced or a maximum number of transitions is reached.

A node in the CG can represent a neural network. A typical use of the neural network would be to generate a motor output given the current percept. Another use would be branching the computation in the graph so that different actions can be applied given the state of the robot. Neural networks play a key role in our model in that they guarantee a general applicability of an evolved skill. We will provide examples of neural computation in section VI.

B. Evolutionary Algorithm

As this work leverages on our previous work described in [1], here we will only sketch the graphical evolutionary

Algorithm 2 The evolutionary algorithm

```
1: Initialize a population of $N$ individual skills $S_i$
2: while The conditions in eq 1 are not met do
3:     Execute the current individual CG using algorithm 3
4:     Extract the current percept $p_t$ from the simulator
5:     Fitness of an individual is $C(p_t)$
6:     for $i = 1 \rightarrow \frac{N}{2}$ do
7:         Select two individuals $g_1$ and $g_2$ from the population
           using Tournament Selection [14]
8:         Generate two children $c_1$ and $c_2$ using crossover [1]
9:         Mutate both $c_1$ and $c_2$ [1]
10:        Mutate the parameters according to eq. 2.
11:    end for
12:    Maintain the best individual of the previous generation
       (elitism).
13: end while
```

algorithm thereby proposed, leaving the details to the more in-depth description provided in [1].

The evolutionary algorithm we used follows the general standard structure described for example in [14] and summarised in algorithm 2. An evolutionary cycle consists in evaluating each individual in the population according to a fitness function and applying the mutation and crossover operators. The implementation made use of the library PyEvolve described in [15]. We improved the approach described in [1] by using evolutionary strategies [16] to control the algorithm's parameters, rendering our approach virtually parameters-free. In particular all the parameters ς_i of the evolutionary algorithm are modified during the mutation phase according to the rule:

$$\varsigma_{t+1} \leftarrow \varsigma_t \exp(\mathcal{N}(\prime, \tau)) \qquad (2)$$

where $\mathcal{N}(\prime, \tau)$ is a Normal distribution with variance τ. In our experiments we used a drifting value $\tau = 0.05$). Finally when skill improvement is performed, half of the initial population will be seeded with the last CG genome, so that a new improved skill can be built starting from the previous one.

The genetic operators affect at the same time both the topology of the CG and the parameters of each action. The topology changes by adding or removing nodes in the graph, changing the head of an edge or changing the starting node. Every mutation has to insure that every node in the CG is reachable by the starting node and every action's outcome has a corresponding edge in the graph. Mutation also affects each action's parameters vector by adding a zero-mean normally distributed Gaussian random number to each parameter. Crossover is performed by swapping subgraphs of two different CGs. These subgraphs are obtained by selecting nodes that are more likely to be strongly correlated. This allows for sub-solutions to be developed and maintained by subgraphs and to be preserved over generations if they contribute positively to the fitness function.

The semantic of a node's parameters in a CG is action's dependent. The evolutionary algorithm treats them as a list of real numbers with respect to mutation operators (no crossover is applied at the parameters level). Examples of parameters are the robot's velocity, the direction of a movement or branching factor in a decision tree. The generality of the parameters allow us to co-evolve neural networks with the CG topology, where the network's weights are represented by an action's parameters. Therefore by mutating an action's parameters we achieve co-evolution of the CG topology and neural networks embedded into the graph.

The fitness function we used in all our experiments in shown in eq. 3.

$$f(S_i) = \begin{cases} 0 & \text{if } \exists p \in W_{S_i} : S_i(p) \text{ fails} \\ C(p_t) & \text{otherwise} \end{cases} \quad (3)$$

The fitness of a skill S_i is 0 if at least one previously successful case cannot be solved, while it is equal to the probability of success, given the current percept, otherwise. Using probabilities instead of binary values allows the evolutionary algorithm to search in a smooth space instead of dealing with non continuous search spaces.

Algorithm 3 Simulating the steps executed by a CG.

```
Require: A computational graph CG, the transition model $\Gamma$
    and the classifier $C$.
 1: Current state is CG's starting state $s_0$
 2: Current percept is $p_0$
 3: $t \leftarrow 0$
 4: while The computation of the CG does not end do
 5:     Generate an action $u_t$ from the current state $s_t$
 6:     Generate next percept: $p_{t+1} \leftarrow \Gamma(p_t, u_t)$
 7:     Generate the action's outcome
 8:     Transition to next state $s_{t+1}$
 9:     if CG is successful then
10:         exit loop
11:     end if
12:     if Maximum number of transitions is reached then
13:         ex it loop
14:     end if
15:     $t \leftarrow t + 1$
16: end while
```

V. Simulating the CG

Evolutionary algorithms require hundreds if not thousands of iterations to converge to a solution. This might be prohibitively time consuming when performed on a real robotic platform. To avoid this problem evolutionary robotics is usually developed in simulation, then the result is applied to the real robot [17], [18], [19], [20]. This creates gaps between a solution found in simulation and its applicability to a real robot. Although these gaps can be reduced by using a very accurate simulator, this increases the computational cost of evolutionary algorithms, to the point that using a simulator becomes as impractical as using a real robot.

Our proposed solution is to abstract away the implementation details of the actions the robot uses to create a skill, and focus only on their effect on the robot's percepts. As described in section III, we are using a transition model Γ that predicts what the next perception will be given the current status and the robot's control signal. When the robot executes an action in the CG, the transition model will produce a new input percept to be used by the subsequent transition. The execution of a CG therefore represents a path in the perceptions space. Similarly the classifier C predicts the probability that the execution of skill will bring a robot from a given percept to a final one where the skill has succeeded. Evolving a computational graph is therefore a **search in the percepts space**. Algorithm 3 describes how a CG is used by the simulator.

Given the above considerations, the accuracy of the transition model does not play a key role in the success of the evolutionary algorithm. Let us consider for example that the robot starts with an initial percept p_0, executes an action a_i and the transition model provides an erroneous[1] next percept p_1. The next computation in the CG does not depend on the previous transition: it is equivalent to a fresh start from a new initial percept $p'_0 \equiv p_1$. We see this as the main advantage of our proposed approach over related approaches like reinforcement learning [21] or planning, that require a faithful transition model to find the correct sequence of steps to reach a goal state from the starting one. Problems might however arise if the transition model does not generate percepts that would happen in the real world, as the skill will never learn to deal with these situations. This means that the better the transition model, the more performant the improved skill will be. Even if the transition model were inaccurate, as long as the skill follows the requirements in eq. 1, it will be considered as an improvement.

[1]By erroneous we mean a transition that is very different from what would have happened in the real world had the robot executed the same action.

In our application we represented the transition model Γ as a Support Vector Machine [22] (SVM). This model is known to yield very accurate results in non-linear regression while being fast both during training and during prediction. We used SVMs to model the classifier C as well, as it is trivial to obtain a probabilistic interpretation of the SVM output. In our implementation we made use of the libsvm library [23] (see the same reference for details on how to obtain a probabilistic interpretation of the classifier's output). Both the classifier and the transition model have been trained in advance using cross-correlation over data collected during several distinct training sessions.

VI. Experiments

We applied our proposed algorithm to a "move-to-pre-grasp-position" skill. The robot starts with a simple hand-coded algorithm and it will have to autonomously improve it so that it will be able to reach objects in a variety of positions. We assume that the dominant scene in front of the robot is composed by a flat table with an object on top. The object is not initially reachable by the robot. The next sections illustrate four iterations of our skills improvement algorithm.

During our experiments we used a mobile manipulator PR2

Fig. 2. The mobile manipulator platform PR2. The robot has two $7DOF$ arms and an holonomic base. The pan-tilt head unit is equipped with two stereo cameras, one high resolution camera and one texture projector (the red light). A tilting laser and a fixed laser on the base are used for navigation and motion planning.

robot manufactured by Willow Garage[2]. It is two-armed with an omni-directional driving system. Each arm has 7 degrees of freedom. The torso has an additional degree of freedom as it can move vertically. The PR2 has a variety of sensors, among them a tilting laser mounted in the upper body, two stereo cameras (with narrow and wide field of view) and a laser scanner mounted on the base which is used for mapping and navigation. Several skills that we relied on were developed by the ROS[3] community, and they include:

- Detecting and grasping unknown objects using $3D$ information [24].
- Planning and executing a collision-free trajectory with the $7DOF$ arms [25].
- Navigation using an omni-directional base [26].

Our evolutionary algorithm made use of the following hand-coded actions:

- **MoveBase**: moves the robot base to a new x, y, θ position. The new position is specified in respect of the robot's frame of reference and it is received as a message from a previous node. Movements are therefore specified as displacements and are independent of an external frame of reference. This action's outcome are either "failure" if an obstacle lies along the robot's path, or "success".
- **Detect**: detects the table and the object over it. The table is a four elements vector $x_{min}, x_{max}, y_{min}, y_{max}$ that represent its boundaries, while the object has a x, y, z coordinates vector (as the object is right on top of the table, its z component corresponds to the table's one). All the coordinates are expressed in the robot's frame of reference. The table and the object vectors are composed to create an $7-$dimensional percept p_t that is used as input to all the actions. As we assumed that the table is a dominant feature in the robot's environment, this action will always have a single "success" outcome.
- **Grasp**: Attempts to grasp the object using the approach proposed in [24]. This action's outcomes are either "success" if the robot can grasp the object, or failure" otherwise. If the action is successful the computation of the CG terminates with success.
- **ContinousNN**: This is standard feed-forward neural network with sigmoid activation function for the hidden layer and linear activation for the output layer. The network has 7 inputs (its input is the world-table percept described above), 3 hidden layers and 3 output layers that correspond to the x, y, θ planar movement of the robot. The network's output is sent as a message to whatever action is activated after it. The network's weights are determined by the action's parameters. This action has a single "success" outcome. This node is primary used to adapt the other nodes to the sensorial inputs the robot receive. For example a neural network will be used to decide where to move the robot in respect of the table and the object's locations.
- **BranchingNN**: This is a feed-forward neural network with sigmoid activation function for both the hidden layer and the output layer. The network has 7 inputs, 3 hidden layers and 2 outputs. The network is interpreted as a binary classifier and it has two outcomes: "out1" and "out2". The role of this network is to allow different paths in the CG to be followed depending on the current percept. The network's weights are determined by the action's parameters and it has a single "success" outcome.

In addition to the above actions we provided a classifier C that outputs the probability that an object is graspable given the current percept and a transition model Γ that predicts what the next percept will be given the current one and the x, y, θ motion displacement. Both the classifier and the transition model have been trained in advance using data collected on the real robot.

The goal of our proposed evolutionary algorithm is to combine the above actions so that the computation of the CG terminates with success. Failure happens when the CG steps for more than 20 transitions (iterations at line 4 in Algorithm 3). The code for all the simulations and for the generic evolutionary algorithm is available online[4].

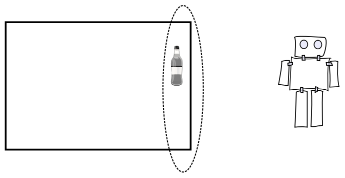

Fig. 3. A schematic representation of the first experiment. The robot is placed on one side of the table and the object is placed along the closest edge. Only objects located inside the dashed ellipsis can be grasped by the initial skill S_0.

A. *Evolving S_0*

In our experimental setup we provided the robot with a simple "out-of-the-factory" algorithm to reach and grasp an object. As the robot detects an object over a table, it will

[2]http://www.willowgarage.com
[3]http://www.ros.org
[4]http://github.com/lorenzoriano/Graph-Evolve

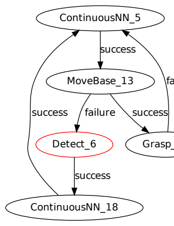

Fig. 4. The CG evolved as skill S_0. The red node is the starting node. Numbers are appended at the end of each node to distinguish between copies of the same action (e.g. ContinousNN_5 and ContinousNN_18).

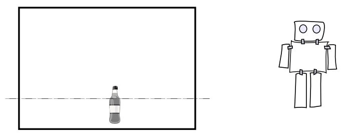

Fig. 5. A schematic representation of the second experiment. The robot is placed on one side of the table while the objects are placed along the left table's edge (from the robot's point of view). The ellipsis indicates the set M_{S_0}.

drive in a straight line towards the edge of the table while centering its base position with the object. As soon as the robot is close to the table's edge it stops and it attempts to grasp the object. This algorithm can successfully move the robot to grasp objects lying on the table's edge which is closest to the robot, as depicted in Fig. 3, but it fails if the object is located along any other edge.

We used this initial algorithm to collect a W_{S_0} set of percepts for we want the (not yet evolved) skill S_0 to terminate with success. The W_{S_0} set contained 5 different initial percepts (object-table locations). For this experiment we used an empty set M_{S_0}. We then used W_{S_0} to create the new CG implementing the skill S_0 shown in Fig 4. This skill perfectly mimics the hand-coded algorithm by managing to grasp the object whenever it was placed along the closest to the robot table edge. Not surprisingly S_0 fails if the object is located along any other edge.

B. Evolving S_1

We tested the newly created skill S_0 with the object placed on the left edge of the table (from the robot's point of view, see Fig. 5), collecting a set of failure cases M_{S_0}. We then applied our proposed algorithm to create a new skill S_1 given the two use cases sets W_{S_0} and M_{S_0}. The resulting CG representing skill S_1 is shown in Fig 6.

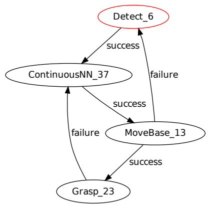

Fig. 6. The CG evolved as skill S_1. The red node is the starting node.

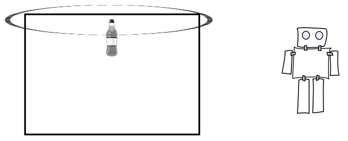

Fig. 7. A schematic representation of the third experiment. The robot is placed on one side of the table while the objects are placed along the right table's edge (from the robot's point of view). The ellipsis indicates the set M_{S_1}

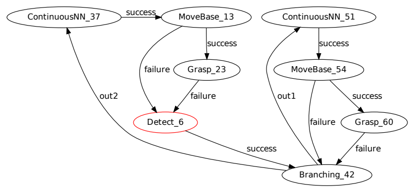

Fig. 8. The CG evolved as skill S_2. The red node is the starting node.

The two previously evolved neural networks ContinousNN_5 and ContinousNN_18 have been replaced by a new one, but the CG structure remained unaltered. The new neural network's behaviour is such that if the object is located close to the edge of the table the robot goes directly towards it, otherwise it performs a series of steps around the left edge of the table while attempting the grasp. The neural network therefore act as a finite state automaton whose input is the current percept.

We tested S_1 with 5 evenly spaced locations along the left edge of the table, obtaining a 100% success rate.

C. Evolving S_2

The next improvement enabled the robot to reach objects located on the right side of the table (Fig. 7). We constructed a new set $W_{S_1} = W_{S_0} \cup M_{S_0}$ while M_{S_1} was composed of percepts corresponding to the object being located on the right side of the table. The resulting new skill S_2 is in Fig. 8.

The evolutionary algorithm maintained the old solution found for S_1, as it can be observed in the left part of Fig. 8, while it evolved a completely new solution to deal with objects on the right side of the robot. We hypothesize that the new solution has been independently evolved only to be merged at a later stage via our proposed cross-correlation algorithm. Here the algorithm made use of the branching node as a way to choose which sub-behaviour to activate. This experiment shows how a previous skill is expanded into a new, better performing one.

We tested the new skill by placing 5 objects on the right side of the table. This time the robot systematically failed to reach an object when placed in the top-right corner of the table. This is due to the Branching_42 node that produced the outcome "out2", therefore attempting to reach the object by going to the left side of the table. This percept was therefore kept for further skill improvement in the next experiment.

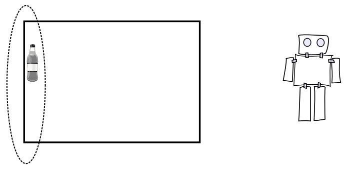

Fig. 9. A schematic representation of the fourth experiment. The robot is placed on one side of the table while the objects are placed along the opposite edge. The ellipsis indicates the set M_{S_2}

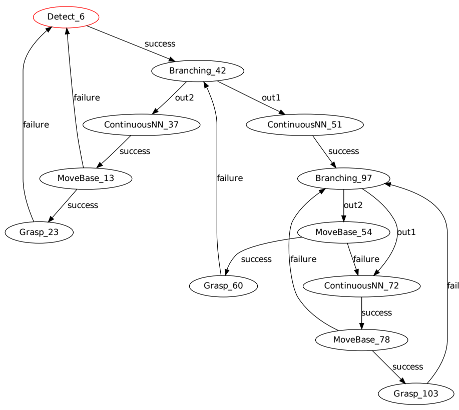

Fig. 10. The CG evolved to as skill S_3. The red node is the starting node.

D. Evolving S_3

In the final experiment we allowed the robot to reach objects placed on the far edge of the table (Fig 9). To this aim we constructed a new positive set $W_{S_2} = W_{S_1} \cup M_{S_1}$ while M_{S_2} was composed by percepts with the object on the far edge of the table and the single failure case from the previous experiment. The resulting CG is shown in Fig. 10.

Although the new skill is far more complex than the others, some patterns can be recognized. The original skill S_0 is still present while the previously evolved S_1 has been modified to accommodate a new branching node Branching_97. This node's output "out1" leads to a new neural network ContinousNN_72 that attempts to move the robot to the opposite of the table. This network is helped by the previously generated ContinousNN_51 that moves the robot along the right edge of the table. Here again we see how our proposed algorithm is re-using a previously evolved skill to generate a new a better performing one.

We tested the new skill by placing the object in 5 equally spaced locations on the far edge of the table. The robot managed to reach the object 3 out of 5 times. In addition to these the robot managed to reach the object in the top-right corner it could not reach before.

As a final test we placed the object in 40 random locations around the perimeter of the table. The skill S_4 obtained a score of 34 successes, with a success rate of 85%. Most of the failures were due to the branching nodes driving the robot to the wrong side of the table, and the CG not having a path to the appropriate area that deals with objects in that location.

VII. Discussion

In the four experiments above we showed how our proposed algorithm builds a solution of increasing complexity by re-using components it had evolved before. The first two skills S_0 and S_1 showed the simple CG structures highlighted in Fig. 4 and Fig. 6. The difference between the two figures points to an emergent feature of our proposed approach: in section IV-A we introduced a constraint on the maximum number of transitions that can happen in a CG. This limitation had been introduced to avoid infinite loops. The same limitation forces the evolutionary algorithm to create compact solutions, otherwise the CG will terminate before the task has been solved. The CG S_0 had two redundant neural network nodes (Continuous_5 and Continuous_18) of which only one's output was being used by MoveBase_13. This redundancy did not lower the CG fitness as the robot could reach the object well before the 20 transitions allowed. The same was not possible for S_1 as the same neural network was responsible for either moving the robot towards the close edge of the table or maneuvering it around the left edge. Therefore the evolutionary algorithm has been forced to create the more compact structure showed in Fig. 6.

Reaching objects on the right edge of the table required a branching node in the graph, as shown in Fig. 8. In this and in the following experiment the evolutionary algorithm adopted a conservatory strategy, i.e. it kept the parts of the CG corresponding to old skills non-mutated while it created new branches to deal with the new percepts. This behaviour is enforced by the constraints in eq. 1 and by the fitness function in eq. 3. However it leads to over-complicated CGs and sub-optimal solutions, as shown in Fig. 10. From the results obtained in the last experiment we can see that the scalability of our proposed approach with the problem complexity needs improvement, and it is the subject of ongoing research.

While both the M and W sets changed with the experiments, the classifier and the transition model did not. This choice is justified by the need of having an unsupervised self-improving system. The above experiments can be easily carried out autonomously by the robot in a "act, collect, improve" cycle. However changing both C and Γ requires an in incremental and life-long learning system which guarantees that no previous knowledge is destroyed by subsequent learning. To the best of our knowledge this is still the subject of ongoing research.

Keeping C fixed forces the evolutionary algorithm to move from its current state (in the perceptual space) to an area which is "covered" by C. As the classifier had been created alongside S_0, the algorithm tried to reach a point in the input space where S_0 would succeed. In the context of our experiments, this means that the robot had to manoeuvre around the table so that the object would be along the closest table's edge (the initial condition for S_0 to succeed) in order to be reachable. This is highlighted by the edge between Grasp_60 and Branching_42 in Fig. 10. This edge links the right side of the graph, responsible for moving the robot around the table, to the left side of the graph, responsible for the final approach and created during the first experiment in section VI-A.

A potential issue with our proposed approach is that the evolutionary operators are applied to the whole graph, independently from the particular area that needs improvement. For example the top-left area of the Fig 10 corresponds to the skill S_0 which reliably solved the problem of reaching a close object. Mutating this area would likely lower the fitness of the CG, thus wasting evolutionary steps to search in areas with low fitness. This problem becomes more evident as the size of the graph increases, slowing down the evolutionary process and hindering the scalability of our approach to complex problems. We believe this is what reduced the skills performances from 100% success rate in the first experiments to 85% in the subsequent ones. This also caused non-optimal graph structures to be evolved, as careful engineering might have produced a more compact CG.

VIII. Conclusions and Future Work

In this paper we presented a novel approach to autonomously skills improving in robotics. This paper is an extension of our previous work on skills building presented in [1]. We conducted several experiments that show how a simple initial skill S_0 undergoes several improvements to become a better performing skill S_3. As a testbed we used the "move to a pre-grasp position" problem. The last skill we developed showed to be capable of reaching objects placed arbitrarily along the table's edges. The computational graphs that have been created during the experiments show the evolution of the robot's skill as it has to solve problems of increasingly complexity.

As we highlighted in the previous section this approach, although successful in autonomously improving a skill, does not scale well with the task's complexity. We are currently investigating potential solutions to this problem, including "freezing" areas of the graph so that the search will not be wasted in low-fitness areas and the whole algorithm will converge more quickly to a solution.

The neural networks we used in the experiments have fixed topology. This does not limit the overall computational power of the computational graph, as different instances of neural network nodes can be used for different purposes. We are however still investigating if our graphical evolutionary algorithm will benefit from using adaptable neural network topologies.

IX. Acknowledgments

Dr. Riano is supported by InvestNI and the Northern Ireland Integrated Development Fund under the Centre of Excellence in Intelligent Systems project.

References

[1] L. Riano and T. McGinnity, "Automatically composing and parameterizing skills by evolving finite state automata," *Robotics and Autonomous Systems*, vol. 60, no. 4, pp. 639–650, 2012.

[2] R. Brooks, "Intelligence without representation," *Artificial Intelligence*, vol. 47, no. 1-3, pp. 139–159, 1991.

[3] R. C. Arkin, *Behavior-based robotics*. MIT Press, 1998.

[4] L. Mösenlechner and M. Beetz, "Using Physics- and Sensor-based Simulation for High-fidelity Temporal Projection of Realistic Robot Behavior," in *19th International Conference on Automated Planning and Scheduling (ICAPS09)*, 2009.

[5] G. Konidaris and A. Barto, "Skill discovery in continuous reinforcement learning domains using skill chaining," *Advances in Neural Information Processing Systems*, vol. 22, pp. 1015–1023, 2009.

[6] W. Chong, M. O'Donovan-Anderson, Y. Okamoto, and D. Perlis, "Seven days in the life of a robotic agent," *Lecture notes in computer science*, vol. 2564, pp. 243–253, 2003.

[7] M. L. Anderson, S. Fults, D. P. Josyula, T. Oates, D. Perlis, S. Wilson, and D. Wright, "A Self-Help Guide For Autonomous Systems," *AI Magazine*, vol. 29, no. 2, Oct. 2008.

[8] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary Intelligence*, vol. 1, no. 1, pp. 47–62, Jan. 2008.

[9] K. O. Stanley and R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[10] C. Mattiussi, "Analog Genetic Encoding for the Evolution of Circuits and Networks," *IEEE Transactions on Evolutionary Computation*, vol. 4193, no. 5, pp. 671–607, Oct. 2007.

[11] F. Gomez and R. Miikkulainen, "Accelerated Neural Evolution through Cooperatively Coevolved Synapses," *Journal of Machine Learning Research*, vol. 9, pp. 937–965, 2008.

[12] F. Stulp, A. Fedrizzi, and M. Beetz, "Action-related place-based mobile manipulation," *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3115–3120, Oct. 2009.

[13] D. Berenson, J. Kuffner, and H. Choset, "An optimization approach to planning for mobile manipulation," *2008 IEEE International Conference on Robotics and Automation*, pp. 1187–1192, May 2008.

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

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

[16] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies," *Natural computing*, vol. 1, no. 1, pp. 3–52, 2002.

[17] I. Harvey, E. Di Paolo, R. Wood, M. Quinn, and E. Tuci, "Evolutionary robotics: a new scientific tool for studying cognition." *Artificial life*, vol. 11, no. 1-2, pp. 79–98, Jan. 2005.

[18] N. Kohl, K. Stanley, R. Miikkulainen, M. Samples, and R, "Evolving a real-world vehicle warning system," *Proceedings of the Genetic and Evolutionary Computation Conference*, 2006.

[19] A. L. Nelson, G. J. Barlow, and L. Doitsidis, "Fitness functions in evolutionary robotics: A survey and analysis," *Robotics and Autonomous Systems*, vol. 57, no. 4, pp. 345–370, Apr. 2009.

[20] J. Urzelai and D. Floreano, "Evolution of adaptive synapses: Robots with fast adaptive behavior in new environments," *Evolutionary Computation*, vol. 9, no. 4, pp. 495–524, 2001.

[21] R. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT Press, 1998.

[22] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[23] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011, software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[24] 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.

[25] OMPL, "The Open Motion Planning Library (OMPL)," 2010. [Online]. Available: http://ompl.kavrakilab.org/

[26] E. Marder-Eppstein, E. Berger, T. Foote, B. P. Gerkey, and K. Konolige, "The Office Marathon: Robust Navigation in an Indoor Office Environment," in *International Conference on Robotics and Automation*, 05/2010 2010.