A Cognitive Architecture made of a Bag of Networks

Alexander W. Churchill[1] and **Vera Vasas**[1] and **Goren Gordon**[2] and **Chrisantha Fernando**[1]

Abstract. Our aim was to produce a cognitive architecture for modelling some properties of sensorimotor learning in infants, namely the ability to accumulate adaptations and skills over multiple tasks in a manner which allows recombination and re-use of task specific competences. The control architecture we invented consisted of a population of compartments (units of neuroevolution) each containing networks capable of controlling a robot with many degrees of freedom. The nodes of the network undergo internal mutations, and the networks undergo stochastic structural modifications, constrained by a mutational and recombinational grammar. The nodes used consist of dynamical systems such as dynamical movement primitives, continuous time recurrent neural networks and high-level supervised and unsupervised learning algorithms. Edges in the network represent the passing of information from a sending node to a receiving node. The networks in a compartment working together encode a space of possible subsumption-like architectures that are used to successfully evolve a variety of behaviours for a Nao H25 humanoid robot.

1 Introduction

Learning a controller for a complex multiple degree of freedom device such as a child or a robot is a non-trivial problem and the principles by which development achieves goals with such high-dimensional systems has been the subject of considerable investigation from the early work of Bernstein [18], the dynamical systems approach of Thelen and Smith [23], to the robotics experiments of Schaal's group on Dynamic Movement Primitives [16]. The critical property of developmental systems that fascinated us was their ability to accumulate adaptations without catastrophic forgetting, a topic that has recently been highlighted by Miller's group [11] as being highly suggestive of a role for structural plasticity in the brain along constructivist principles [1]. A holy grail in developmental robotics would be to achieve open-ended adaptation over long periods of time, with the same cognitive architecture being able to bootstrap prior knowledge and competences to achieve more and more complex tasks. Whilst incremental evolution has been demonstrated for one or two steps [4], an open-ended accumulation of adaptation has not been achieved. An attempt has been made in genetic programming to evolve subsumption architectures [24] but scaling up to high-dimensional robots has not been attempted.

We use a directed graph based control architecture capable of parallel processing, in which cyclic networks are allowed. The network is analogous to a neural network controller, however, the nodes can be much more complex than simple neurons. There is a long history of evolving parallel potentially cyclic directed graph representations, for example Teller and Veloso use parallel evolution for data mining (the PADO algorithm) to solve supervised learning problems [22]. Ricardo Poli uses a parallel genetic programming system called PDGP (parallel distributed genetic programming) to evolve feedforward parallel programs to solve the XOR problem amongst others [13]. Cartesian genetic programming [12] also evolves feed-forward programs. The dominant paradigm for structural evolution of networks is now NEAT [19], a powerful system because it allows diversity maintenance, protects innovations, and controls the dimensionality of the genotype carefully. Our system differs from the above approaches in several respects. First, it is intended to be a cognitive architecture for the real-time control of robots, secondly the nodes are intended to be complex machine learning algorithms, not arithmetic primitives or simple mathematical functions, thirdly, our system is intended to be capable of controlling a humanoid robot. As in the existing systems our nodes have multiple functions and transmit information between each other. Our motivation for using the graph representations was to eventually achieve compositionality, systematicity, and productivity because it is possible for specific node types to be physical symbol systems [3] if the rules for operating on them are grammatical. Previous attempts have been made to achieve grammar based genetic programming [10] in tree structures, but not in cyclic distributed parallel graph structures of the types described above, to our knowledge. The challenge is to discover an evolvable grammar that generates fit variants with high probability, a task not unlike that faced in language learning by the fluid construction grammar [20].

The structure of the paper is as follows. The network representation is described followed by the stochastic grammatical variation operators, and the overall evolutionary algorithm that controls the robot in real-time by generating and selecting active units. Some preliminary adaptation results are presented, and the network structures responsible for the behaviour are described. We have not yet achieved the accumulation of adaptation, but we have shown that a variety of distinct tasks can be evolved with disjoint network representations that are suitable for recombination. Several critical problems are encountered and discussed, and an approach is sketched for making progress.

2 Methods

The basic building block of the model is a node, it gets vector inputs, performs a processing operation, and sends vector messages to other nodes with some delay. Nodes when activated, transmit this activity state to downstream nodes, and remain active for a fixed period. Nodes can have internal states. Nodes are connected to create potentially disjoint directed graphs. The unit of neuro-evolution consists of a set of such disjoint graphs in a compartment. **Sensory nodes** get raw sensory input, or act as internal pattern generators that encode the amplitudes and frequencies of a vector of sine waves. These

[1] Queen Mary, University of London, UK, email: {a.churchill,v.vasas,c.t.fernando}@qmul.ac.uk
[2] MIT Media Lab, Cambridge, MA, USA, email:goren@gorengordon.com

are formally sensor nodes because they can be initiator nodes for a graph, i.e. are active unconditionally. **Motor nodes** contain a vector of motors that they control. Their input vector is used to set the motor angles of these motors at each time-step the node is active, and a re-afferent copy or corollary discharge of the command angles is always output [2]. If there is no input, a default motor angle for each motor is stored. **Processing nodes** receive inputs *only* from other nodes. They include Euclidean distance nodes that gets a vector from its input nodes and calculates the distance to an internally stored parameter vector; a linear transform node that does the dot product of the input vector and an internally stored weight matrix to produce a vector of outputs; an Izhikevich neuron based liquid state machine [7] node. A **reinforcement learning node** has a two part input vector, a data part and a reward part. The data part consists of inputs from a set of other nodes, typically transform nodes or sensory nodes e.g. signalling joint angles. The reward part must be a single scalar value obtained from a transform node, for example this could signal the prediction error or the proximity to a desired state. The output of a learning node is a vector that will typically encode some parameters for downstream motor nodes to use, or it might be a temporal difference error signal. Types of reinforcement learning node include a **stochastic hill climbing node**, an **actor-critic node**, a **simulated annealing node** and a **genetic algorithm node**. **Supervised learning nodes** undertake online supervised learning based on an input training vector, and an input target vector. They do function approximation (regression) or classification. Such an node is essential for efficiently learning models of the world, e.g. forward models, or inverse models [17, 9, 5]. The type of model depends purely on the identity of the transform node that sends it the training and target vectors during its training. **Unsupervised learning nodes** take an input vector and compress it, e.g. a **k-means clustering node** does online clustering and outputs the class of each novel data point input to it; a **Principle Component Analysis node** takes an input vector, has a parameter n which is the number of principle components to output, and the output vector is the intensity of those n first principle components. In the Nao setup, there are high-level **face recognition nodes**, i.e. a sensor node that receives input from a camera and outputs a label and x,y retinal coordinate of a face; and a **speech recognition node**, which is a sensor atom that receives input from a microphone and outputs either the text or label of the speech. Other high level nodes in the Nao include **walk nodes**, motor nodes that control the legs to walk to a particular location and take high level Cartesian instructions as input. To summarise, the repertoire of nodes is essentially unbounded and is only limited by their encoding into the framework. In this manner, any combination of such nodes can form, thus enabling an open-ended evolution of complex controllers. This advantage is also a disadvantage, the number of nodes is huge, and the search space of possible networks is absolutely enormous. This makes evolution in such a space extremely difficult.

Activation of the networks in a compartment always start at the sensory nodes, these activate in parallel the downstream processing nodes with the appropriate delays, which in turn are typically connected to motor nodes. There may be recurrent connections. The unit of neuroevolution is a "bag" of disjoint graphs to which fitness is assigned as a unit. The motivation for this is that it enables the robot to be subject to potentially multiple parallel independent behavioral policies that together control the whole robot in parallel. By allowing graphs to be disjoint, a bag of potentially useful network motifs can be preserved silently and become utilised by the system in later variation events during evolution. This unusual nature of of compartment representations will become more clear when the initialisation of networks and the variation operators acting on them is described below.

2.1 Evolvable Action Grammar

We now describe the variation operations or graph rewrite rules [14]. **De novo node constructor** operators make new networks at initialisation giving the robot a primary sensorimotor repertoire of reflexes. For example such a network may consist of a single sensory node with a random number of sensory inputs connected to a linear transform node with a delay. The linear transform unit may have a random initial weight matrix sending output with random delays to three motor nodes. Each motor node may control 1 to 4 motors, randomly chosen. A complex constructor is also used throughout evolution as a macro mutation device for inserting functional motifs that consist of random sensory-(linear transform)-motor triplets in a chain of activation. **Graph replicator** operators produce perfect copies of graphs. These are required for replication and evolution of compartments (bag of graphs). **Graph connectivity operators** only influence the connectivity between already existing nodes within a single graph i.e. the messages that nodes get from other nodes within the molecule, the delays with which the messages influence activation, and the time period for which a node is active after activation. For example, the **connection deletor** removes a message i.d from the input vector of a node. A **connection adder** adds a new message i.d. of a node in the graph to the input vector of another node in the graph. For now, all nodes are allowed to connect to/from any other node type, but evolvability in future work will demand this to be constrained. A critical feature of the above operators is that deletion of an edge may create two disjoint graphs that remain in the same compartment. **Node duplication** operations make copies of nodes within a single graph. The **parallel node replicator** takes a node and copies it entirely along with its input and output links such that this node now exists in parallel with its parent node, with the same inputs and outputs of the parent node. The **serial node replicator** copies a single node but instead of copying the inputs and outputs of the node, it makes the input to the offspring node the output of the parent node, and makes the offspring node connect to those nodes that the parent node connected to, i.e. it inserts a copy of the parent node downstream of the parent node. A **node deletor** operator removes a node along with its input and output edges, thus potentially disconnecting a graph into two (or more) components. All these operators act on the message i.d list in the input vector of a node. Thus connectivity is encoded at the input end of nodes, not at the output ends. **Intranode operators** modify the sensory inputs, the motors controlled, the transfer functions and the internal states of the nodes. Intra-node mutations depend on the specific node type. Intra-node operators do not mutate the type of the node. This greatly restricts the mutability of the genome, but increases its evolvability. **Multi-graph recombinators** take N networks and construct a network that is derived from properties of the N parent networks. In analogy with genetic programming [8], a standard operator is the **branch crossover operator** that is defined only on trees and not recurrent graphs. This chooses two nodes and crosses them over along with any downstream nodes to which they are connected. **Motif crossover operators** allow crossover only between functionally isomorphic digraph motifs, in order to prevent crossover from being destructive due to the competing conventions problem.

The evolutionary algorithm consists of a population of 20-100 units of evolution (compartments). A binary tournament genetic algorithm is used to select pairs of units which are evaluated sequen-

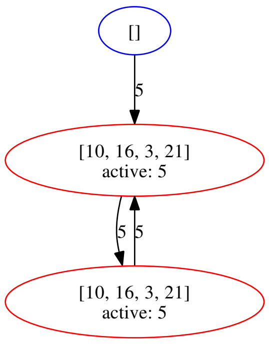


Figure 1: Showing a hand-designed actor graph, where graph edge labels refer to activation delay times, "active" shows the number of time steps a node remains active for and motors are labelled in the motor atom (red). Motor key: 0:HeadYaw, 1:HeadPitch, 2:LShoulderPitch, 3:LShoulderRoll, 4:LElbowYaw, 5:LElbowRoll, 6:LWristYaw, 7:LHand, 8:LHipYawPitch, 9:LHipRoll, 10:LHipPitch, 11:LKneePitch, 12:LAnklePitch, 13:LAnkleRoll, 14:RHipYawPitch, 15:RHipRoll, 16:RHipPitch, 17:RKneePitch, 18:RAnklePitch, 19:RAnkleRoll, 20:RShoulderPitch, 21:RShoulderRoll, 22:RElbowYaw, 23:RElbowRoll, 24:RWristYaw, 25:RHand.

tially on the robot. Their fitness is compared and the winner overwrites the loser with mutation and recombination as described above in the operator section. This continues until the behaviour is interesting, or the robot breaks or overheats.

3 Results

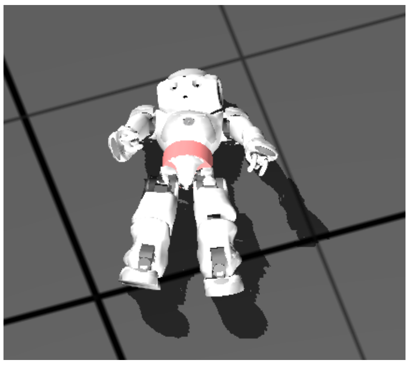
This section should be read while referring to the accompanying videos in the Supplementary Material.[3] Simulations are conducted in Webots for Nao, and real world experiments are conducted with the Nao robot itself.

3.1 Maximising distance travelled by a humanoid robot

In the first experiment the fitness of a compartment is defined as the distance travelled by the Nao during a single episode. The compartments successfully evolved to increase the distance travelled by the robot in a fixed time period. In order to test the efficacy of the intra-atomic mutations, an initial graph de novo constructor was used. An example initial graph can be seen in Figure 1 and is formed of three nodes - a sensor that is used for activation only, i.e. with no output, and two motor nodes, which operate the same motors. This graph is called a SMM molecule. The activation and delay times were initialised to create an oscillating effect by alternating between motor angles set to opposite directions in each motor node.

In the first evolutionary run of this system, only the motor angles and the delay and activation times of the actor graph were mutable, not the graph topology. This meant that fitness could only be improved through increasing the speed and strength of oscillations during the fixed time period. Each trial lasted for 50 time steps of 0.1 seconds, and at the start the simulated Nao robot was set to a resting supine position, illustrated in Figure 2. Evolution was performed for 250 generations, each consisting of 10 binary tournaments and 20

[3] Supplementary material is available at http://www.robozoo.co.uk/research/papers/aisb2014


(a) Front

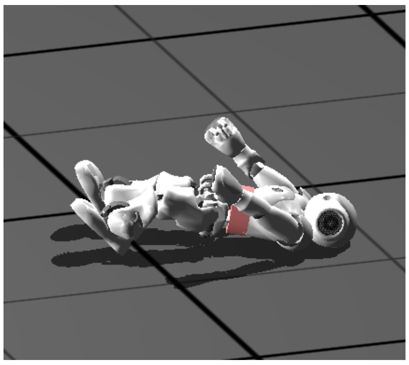
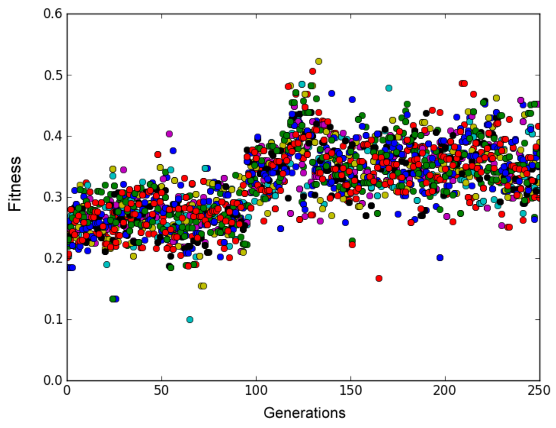

(b) Side

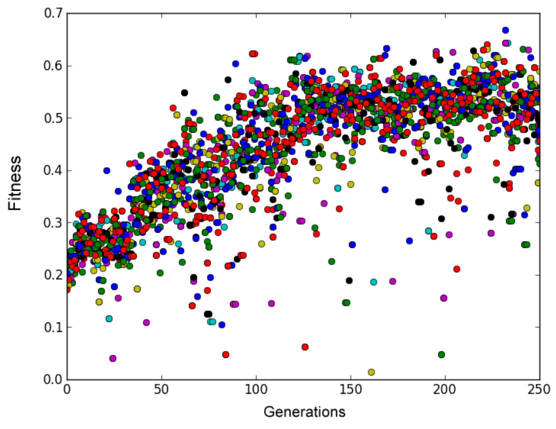
Figure 2: Showing the resting position of the NAO, which it was reset to at the start of each trial. Images are from different angles of the same scene.

evaluations. The fitness of the individuals at every 10 generations is shown in Figure 3(a). There is a clear, although not dramatic increase in fitness (distance travelled) over the course of evolution. Fitness improvements occur after 100 and 120 generations, where the best solutions move from a distance of around 0.3 metres to close to 0.4 and then close to 0.5 metres. In terms of behaviour, the initial setup conditions create a largely symmetrical rocking motion that moves the robot slowly towards its left. The best behaviour after 50 generations rocks the robot with much greater force, achieved by raising the legs higher, which causes it to move further to the left over the course of a time trial. At 100 generations, the legs move higher and are raised for more time, which causes the robot to rotate further, and thus increases the amount of distance travelled. At 150 generations, it employs the same behaviour, moving to its right, but the legs oscillate more quickly. This final behaviour appears to be a local optimum for the robot.

In a second evolutionary run, the complexity of node mutations was increased to make motor identities mutable, but again with graph topology held fixed. The same starting conditions, i.e. the node structure and parameters, were used as in the first run. The fitness of individuals over the course of evolution is shown in Figure 3(b). Here, a similar performance to the first run is achieved after only 70 generations, and by 100 evaluations the system exceeds the best results found on the first run. After 50 generations the behaviour looks similar to the first run. The robot rotates from side to side, biased towards movements on its left side. The molecular graphs in Figure 4 show that after 50 generations, 5 left sided motors and 3 right sided motors are employed. After 100 generations, the behaviour of the best com-

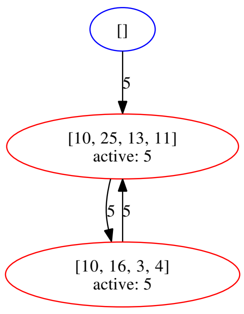


(a) Activation and Delay time and motor angle mutations only

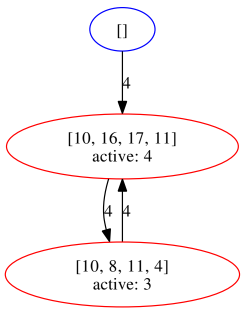


(b) Activation and Delay time, motor angle and motor mutations

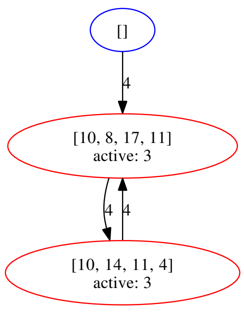
Figure 3: Showing the fitness of actor molecules over 250 generations of evolution on the maximise distance task, for (a) when only activation and delay time and motor angles mutations permitted and (b) where additionally the motors used could be mutated. Each generation consists of 10 binary tournaments.



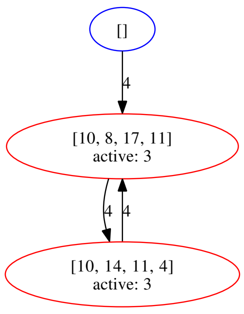
(a) Generation 50



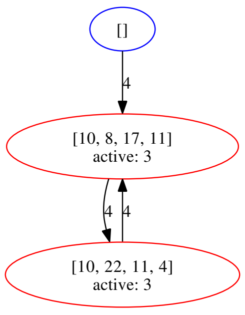
(b) Generation 100



(c) Generation 150



(d) Generation 200



(e) Generation 250

Figure 4: Showing the atomic structure of the best actor molecules after (a) 50, (b) 100, (c) 150, (d) 200 and (e) 250 generations on the second evolutionary run of the maximise distance task. Sensor atoms (blue) provide activation signals only. Edge numbers indicate delay times and "active" shows the number of time steps a motor atom (red) remains active for. Refer to Figure 1 for the motor key.

partment has changed. The left leg now extends quite far and then drags the robot towards its left side. Looking at the node structure, the second node is now composed entirely of left sided motors. At 200 generations the right leg also extends and a stepping motion emerges. The behaviour of the best controller after 100 and 5000 evaluations is shown in video V1 in the Supplementary Material, both in simulation and on the real robot.

3.2 Evolving headstands in a humanoid robot

In the second experiment, compartments were evolved for 3,000 binary tournaments (6,000 evaluations) with all the variation operators included. Fitness was obtained by maximising the z-accelerometer sensor in the torso of Nao over the course of 150 time steps. For example this would be maximised by the robot standing on its head. Figure 5 shows the fitness of each evaluated member of the population at each pseudo-generation of 10 binary tournaments. The compartments are clearly increasing in fitness over the course of evolution. There is an especially large jump over the first 50 generations, and then a more gradual but positive ascent throughout the rest of evolution. The graph on the right of Figure 5 shows the earlier fitness history of the population. After 32 generations there is a sudden drop in fitness. This occurs because the Nao moves into a position on its side where it is balanced and unable to return to the rest position on its back. This means that compartments are now selected in effectively a different environment. After 6 generations, Nao moves out of this position and can reset normally again.

The compartment representing the fittest individual found at the end of evolution can be seen in Figure 6. The compartment is made up of 3 disconnected graphs that get activated and produce movement over the course of a trial, labelled as Groups A - C, with a small number of atoms that will lie dormant (never activated). Among the dormant nodes there is a sensor node connected to a linear transform node that will be activated but will have no effect on the behaviour

Table 1: Fitnesses of the individual and combinations of disjoint graphs from best compartment found on the maximise z-accelerometer task, labelled as shown in Figure 6.

Group	Fitness
A	575
B	459
C	272
A + B	772
A + C	634
B + C	590
A + B + C	807
A + B + C (Changed Z-accelerometer sensors to X-accelerometer sensors)	467

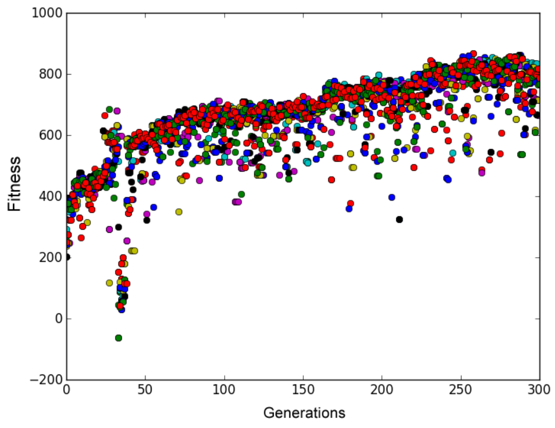
of the robot. The fitnesses scored by the three separate active groups can be seen in Table 1. The behaviour of each entry in the table can be seen in Video V2. The compartment containing all three parts performs the best, and the fitness is considerably greater than any one group on its own, showing that the disjoint graphs do interact behaviourally to create a more adaptive behaviour than any single disjoint component of the compartment. The final robot stance obtained by this compartment can be seen in Figure 7(a). Group A, the largest disjoint graph, has the highest fitness when considered on its own. It causes both of Nao's legs to move, as well as its right arm, which causes it to move backwards into an arch-like position, raising its torso and so increase its z-accelerometer reading. Group B also scores well by itself, mainly moving Nao's legs, with a larger emphasis on the right one. This leads it to move its torso backwards. Group C does not produce much movement when starting from a resting position. The combination of Groups A and B moves both legs far back underneath the torso, arching Nao backwards at a sharp angle, as can be seen in Figure 7(b). With all three groups together, the robot moves its legs even further backwards, drawing the torso even closer to a vertical position.

The two highest scoring groups (A and B) both make use of the Z-accelerometer sensor (143 in Figure 6). To test the impact of this sensor, the molecule was modified to replace all occurrences of the Z-accelerometer sensor with the X-accelerometer, which had the affect of oscillating the robot's torso from side to side, leading to a much reduced fitness score. The final stance produced by the modified compartment can be seen in Figure 7(c). This demonstrates that a true closed-loop solution has been found.

An interesting variant of the above task involves initialising the robot from the prone position rather than the supine position. Figure 8 shows that evolution to max z from a prone solution is much faster and more effective evolution to max z from a supine position. Video V3 shows the best behaviour evolved after 1400 evaluations. The solution is quite different, exploits the initial position of the Nao and is elegantly simple.

3.3 Evolving bouncing behaviour in a humanoid robot on a Jolly Jumper

In the third set of experiments, a physical NAO robot is placed in a Jolly Jumper infant bouncer (see Figure 9), which consists of a harness attached to a stand by a spring. This is influenced by Thelen's work [23], which describes how an infant placed in a bouncer discovers configurations of movement dynamics that provide acceptable solutions and then tunes these to produce efficient bouncing. The robot is placed in the harness and suspended with the tip of its feet firmly touching a cushion on the ground, in the first experiment, and just touching the floor in the second. The fitness function is to maximise



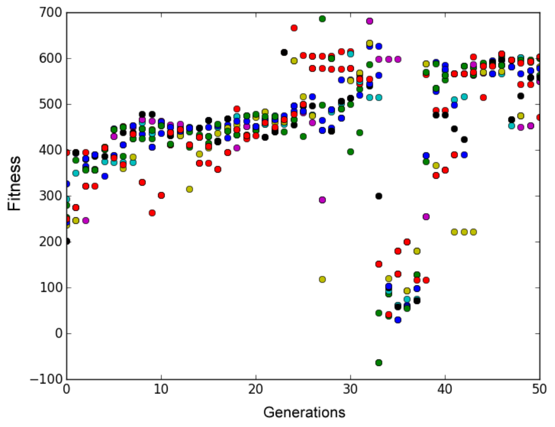


Figure 5: Showing the fitness of compartments over 300 generations (top) and 50 generations (bottom) of evolution on the maximise z-accelerometer task. Each generation consists of 10 binary tournaments and 20 evaluations. Both figures show data from the same run.

the first derivative of the z-accelerometer (z) at each time step (t), $f(x) = \sum_{t=1}^{T} |z_{t+1} - z_t|$. Each trial lasts for $T = 100$ time steps of 100ms each. At the end of the trial, the robot's joints are relaxed and it rests for 3 seconds. It may not return to the same position at the start of each trial, and the spring will not be completely dampened from the previous appraisal, which introduces noisy fitness evaluations.

In the first experiment, a Microbial GA with a population of 10 is applied for 800 evaluations to a fixed graphical structure, evolving only the parameters. The compartment can be seen in Figure 10, and consists of four Dynamic Motor Primitive (DMP) [15] nodes, connected to the knee and ankle motors, and receiving afferent signals from the motors as input. Additionally, two separate graphs take the force sensitive resisters (FSR) from each foot as input, and output to the ankle and knee motors for their respective sides of the body. These graphs will take control if the FSR signal is above an evolvable threshold. Figure 11 shows the fitness of the population over the course of evolution. The run proceeds in several stages, which are shown in Video V4. Initially a kicking motion develops, that enables fitness to increase quickly. After 120 evaluations, the feet get stuck on the cushion, and the robot is unable to use its previous motion. A new strategy emerges, moving the knee as far back as possible before kicking. After 200 evaluations, with the feet now unstuck, a new solution is found, moving the left knee far back and relying on a fast kicking motion in the right leg to bounce. After 350 evaluations, the

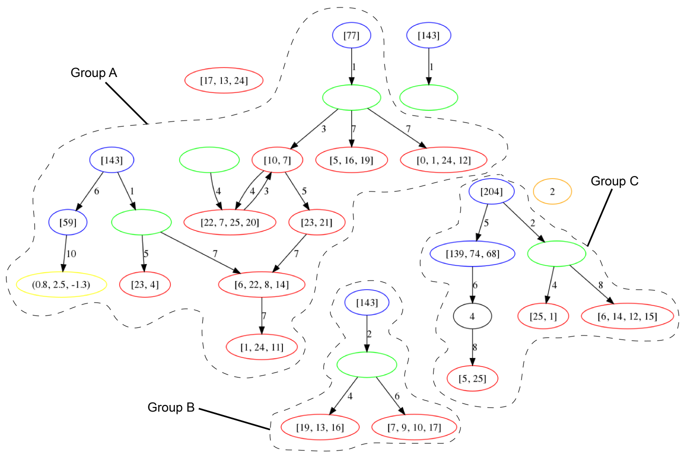
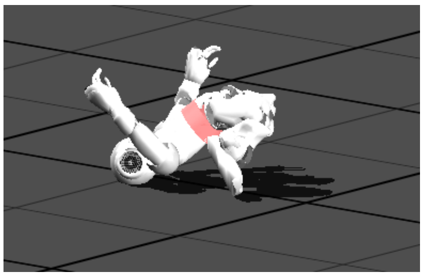
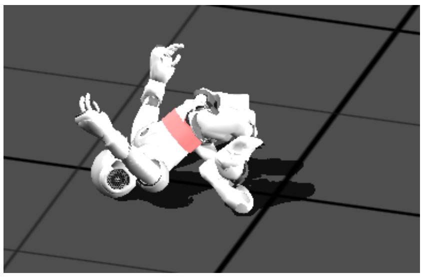


Figure 6: Showing the compartment that represents the best individual found on the maximise Z-accelerometer task. There are three disjoint connected graphs that get activated during a run, and these have been highlighted and labelled A, B and C. Blue show sensors with the numbers representing sensors shown in the key below, green represents linear transform nodes with a maximum of 5 outputs, yellow Euclidian distance nodes with their target vector denoted inside, orange shows PCA nodes with the number of outputs denoted inside, black shows a K-Means node with the number of clusters denoted inside and red shows motor nodes with motors denoted inside (refer to Figure 1 for the motor key). Sensor key: 59:HeadYaw, 68:LKneePitch, 74:RElbowRoll, 77:RHipPitch, 139:GyroscopeY, 143:AccelerometerZ, 204:DistanceZ.

cushion is removed, and the robot is now suspended above the ground at the start of each trial. Fitness falls sharply, as previous strategies relying on contact with the ground are much less successful. Fitness quickly improves, and solutions adapt to this new condition, with the knees travelling much greater distances during each oscillation. After 500 evaluations fitness returns to the previous level and then is quickly surpassed, with the best solutions achieving a fitness of over 9. This shows that the presented architecture is able to quickly adapt in a dynamic environment.

A second experiment explores a similar task but with an evolved topology. A more complex evolutionary process is employed in this experiment, inspired by NEAT [19]. A population is seeded with 20 identical copies of a small graph consisting of a DMP node connected to the left knee and receiving an afferent signal as input. This is initialised to perform a basic oscillatory motion. As in NEAT, each atom is assigned a unique id. An individual, i, is sequentially compared to each member, j, of each species using the similarity measure, $s(i,j) = \frac{1}{4}[s_s(i,j) + s_p(i,j) + s_m(i,j) + s_{shc}(i,j)]$. Here, $s_s(i,j)$, $s_m(i,j)$, $s_p(i,j)$, are functions which return the number of shared sensors, motors and processing node functions respectively, and $s_{shc}(i,j)$ returns the number of shared connections between nodes with the same unique id. If $s(i,j)$ is below a threshold, s_{thresh} (0.5 in this experiment), i is assigned to the species of j and the search is halted. Otherwise, i is assigned to a new species. The fitness score of each individual is divided by the number individuals in its species. A mutation count, mut_c, initialised at 0, is also assigned to each new individual, and incremented if a mutation event occurs. Structural mutations are only permitted if $mut_c > 3$. In this way, innovations are protected and behavioural diversity is encouraged in the population. Additional variation operators pertaining to the DMP


(a) Groups A, B and C together


(b) Groups A and B together

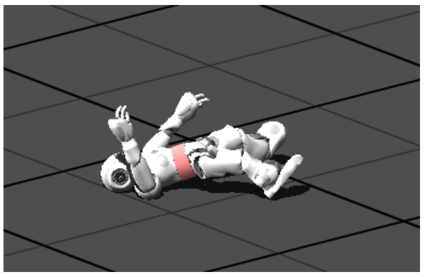
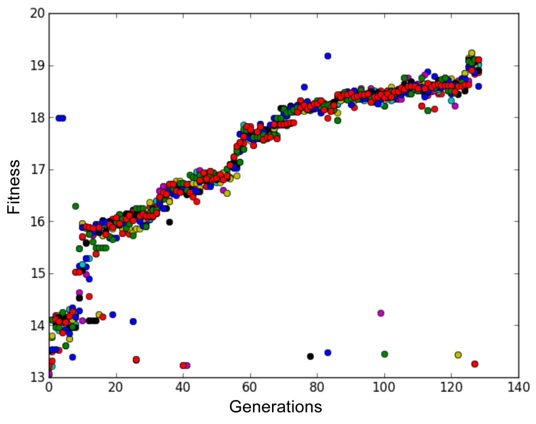

(c) Groups A, B and C together, X-accelerometer substituted for Z-accelerometer

Figure 7: Showing the final position of the Nao after different molecular groups are used for the actor, labelled as shown in Figure 6.

atoms are also included in this experiment. Disjoint graphs containing a DMP have an explicit probability of duplicating, and a DMP in one graph can be replicated and replace an existing DMP in another.

Evolution was again run directly on a real NAO robot, over the course of 11 hours. Figure 12 shows the fitness of each individual in the population after each evaluation. There is a clear improvement in competence on this task from the individuals in the initial population to those at the end of the run. The initial behaviour that the population was seeded with provided an acceptable fitness of between 3 and 5. After around 500 evaluations, structural mutations begin to have a noticeable effect on fitness. There is a climb in fitness from evaluations 500 to 1000, where the best individuals improve from fitness of 5 to 6.5, and again from evaluations 1100 to 1500, where the fitness of the best increases to around 8.5. Figure 13 shows the compartment of the best individual from the final population. As with the compartment shown in Figure 6, it contains redundancy, with several graphs lacking sensors or motors. The main driving force is a single DMP, which is used to control the left hip, right hip, left shoulder and right knee. A second molecule also controls the right knee, which can reset actions sent from the DMP, and create additional oscillatory movements. This solution produces rapid kicking motions, enabling it to move quickly along the longitudinal axis, and thus attain fast changes in its z-accelerometer sensor.



(a) Evolutionary fitness dynamics

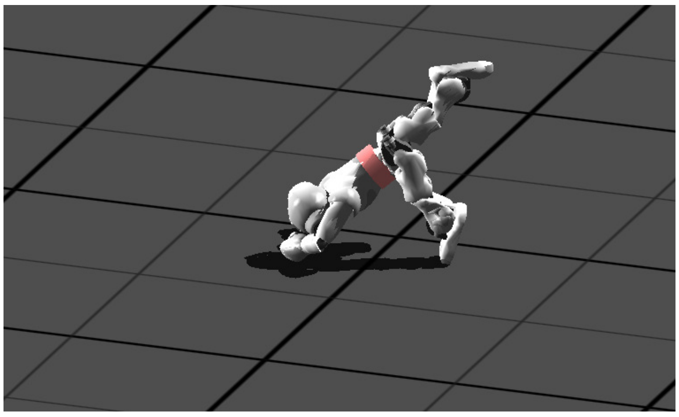

(b) Final pose obtained

Figure 8: Evolution to maximise the z-accelerometer from the prone position. (a) There is a punctuated increase in fitness. (b) The final pose obtained is of high fitness.

4 Conclusion

An outline and some preliminary investigations of a modular and evolvable control architecture for high-dimensional robotic systems has been described. It remains to see whether the architecture is capable of the accumulation of adaptation and transfer learning across multiple tasks by the use of an archive of control motifs. The problem with the architecture is simultaneously its strength, i.e. it is too rich. A detailed investigation of the evolvability of the architecture will be needed in future if it is to be made efficient. One aspect missing from the architecture currently is gating and channeling at the inputs and outputs. Evolvable control flow operations are also needed. This adds an even greater complexity to the evolvability problem. An alternative approach is to begin with existing architectures for modular robot control, e.g. [6] [21], encode them within the above framework, and allow them to mutate. What is clear is that methods are acutely needed to deal with the issue of poor evolvability arising from the large space of possible networks in the current system.

ACKNOWLEDGEMENTS

The work is funded by the FQEB Templeton grant "Bayes and Darwin", and the FP-7 FET OPEN Grant INSIGHT.

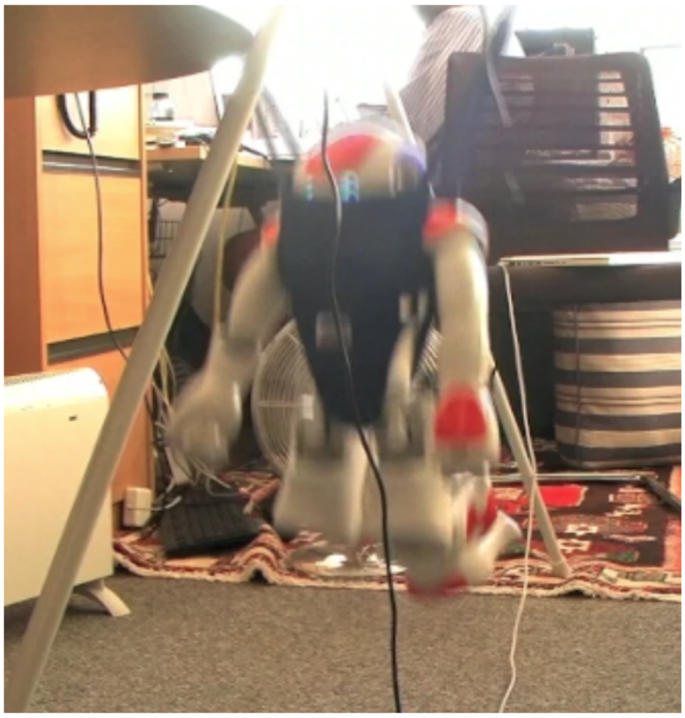


Figure 9: The NAO robot in a Jolly Jumper infant bouncer.

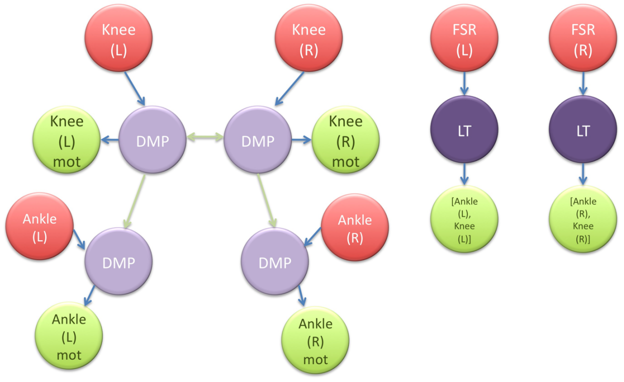


Figure 10: The fixed topology compartment used in the first Jolly Jumper experiment. DMP = Dynamic Motor Primitive, FSR = Force Sensitive Resistors. The red nodes are sensors and the green nodes are motors.

REFERENCES

[1] D.B Chklovskii, B.W. Mel, and K. Svoboda, 'Cortical rewiring and information storage', *Nature*, **431**, 782–788, (2004).

[2] T. B. Crapse and M. A. Sommer, 'Corollary discharge across the animal kingdom', *Nat Rev Neurosci*, **9**(8), 587–600, (2008). Crapse, Trinity B Sommer, Marc A R01-EY017592/EY/NEI NIH HHS/United States Research Support, N.I.H., Extramural Research Support, Non-U.S. Gov't Review England Nature reviews. Neuroscience Nat Rev Neurosci. 2008 Aug;9(8):587-600.

[3] J.A. Fodor and Z.W. Pylyshyn, 'Connectionism and cognitive architecture: A critical analysis', *Cognition*, **28**, 3–71, (1988).

[4] Faustino Gomez and Risto Miikkulainen, 'Incremental evolution of complex general behavior', *Adaptive Behavior*, **5**(3-4), 317–342, (1997).

[5] G. Gordon and E. Ahissar, 'Hierarchical curiosity loops and active sens-

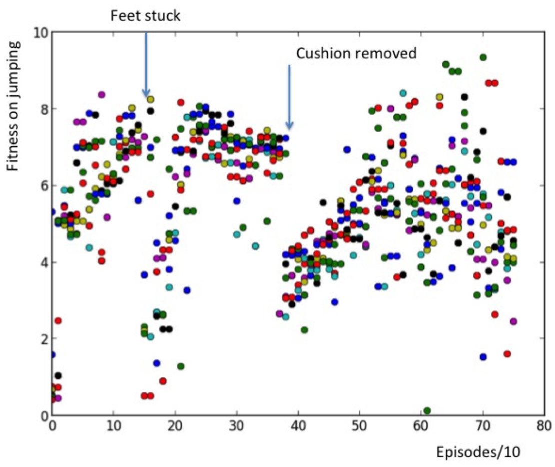


Figure 11: Showing the fitness solutions of every 10 evaluations, over 800 evaluations on the first Jolly Jumper experiment.

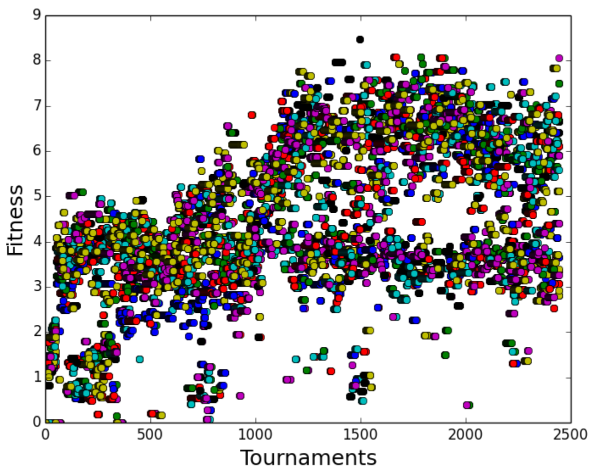


Figure 12: Showing the fitness of all solutions over 11 hours of evaluations on the second Jolly Jumper experiment.

ing', *Neural Netw*, **32**, 119–29, (2012). Gordon, Goren Ahissar, Ehud Neural Netw. 2012 Aug;32:119-29. Epub 2012 Feb 14.

[6] M. Haruno, D.M. Wolpert, and M. Kawato, 'Mosaic model for sensorimotor learning and control', *Neural Computation*, **13**, 2201–2220, (2001).

[7] E. M. Izhikevich, 'Polychronization: computation with spikes', *Neural Comput*, **18**(2), 245–82, (2006). Izhikevich, Eugene M Neural Comput. 2006 Feb;18(2):245-82.

[8] John R. Koza, *Genetic programming III : darwinian invention and problem solving*, Morgan Kaufmann, San Francisco, 1999. 99010099 John R. Koza ... [et al.]. ill. ; 25 cm. Includes bibliographical references (p. [1081]-1114).

[9] H. Lalazar and E. Vaadia, 'Neural basis of sensorimotor learning: modifying internal models', *Curr Opin Neurobiol*, **18**((6)), 573–581, (2008).

[10] Robert I Mckay, Nguyen Xuan Hoai, Peter Alexander Whigham, Yin Shan, and Michael ONeill, 'Grammar-based genetic programming: a survey', *Genetic Programming and Evolvable Machines*, **11**(3-4), 365–396, (2010).

[11] Julian F Miller and Gul Muhammad Khan, 'Where is the brain inside the brain?', *Memetic Computing*, **3**(3), 217–228, (2011).

[12] Julian F Miller and Peter Thomson, 'Cartesian genetic programming', in *Genetic Programming*, 121–132, Springer, (2000).

[13] Riccardo Poli, *Parallel distributed genetic programming*, Citeseer, 1996.

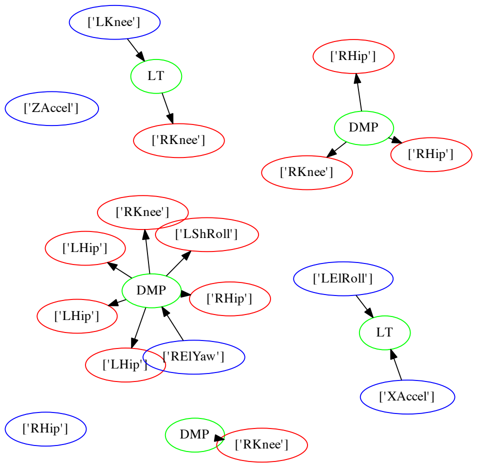


Figure 13: Showing the best compartment found at the end of evolution on the second Jolly Jumper experiment.

[14] Grzegorz Rozenberg, *Handbook of graph grammars and computing by graph transformation*, World Scientific, Singapore ; New Jersey, 1997. 96037597 edited by Grzegorz Rozenberg. ill. ; 23 cm. Includes bibliographical references and indexes. v. 1. Foundations.

[15] Stefan Schaal, 'Dynamic movement primitives-a framework for motor control in humans and humanoid robotics', in *Adaptive Motion of Animals and Machines*, 261–280, Springer, (2006).

[16] Stefan Schaal, Jan Peters, Jun Nakanishi, and Auke Ijspeert, 'Learning movement primitives', in *Robotics Research*, 561–572, Springer, (2005).

[17] Reza Shadmehr, 'Generalization as a behavioral window to the neural mechanisms of learning internal models', *Human Movement Science*, **23**, 543–568, (2004).

[18] O. Sporns and G. M. Edelman, 'Solving bernstein's problem: A proposal for the development of coordinated movement by selection', *Child Development*, **64**, 960–981, (1993).

[19] Kenneth O Stanley and Risto Miikkulainen, 'Evolving neural networks through augmenting topologies', *Evolutionary computation*, **10**(2), 99–127, (2002).

[20] L. Steels and J. De Beule. Unify and merge in fluid construction grammar, 2006.

[21] J. Tani and S. Nolfi, 'Learning to perceive the world as articulated: an approach for hierarchical learning in sensory-motor systems', *Neural Networks*, **12**, 1131–1141, (1999).

[22] Astro Teller and Manuela Veloso, 'Program evolution for data mining', *International Journal of Expert Systems Research and Applications*, **8**, 213–236, (1995).

[23] Esther Thelen, 'Motor development: A new synthesis.', *American psychologist*, **50**(2), 79, (1995).

[24] Julian Togelius, 'Evolution of a subsumption architecture neurocontroller', *Journal of Intelligent and Fuzzy Systems*, **15**(1), 15–20, (2004).