Island-based evolutionary learning

Pieter Spronck Ida G. Sprinkhuizen-Kuyper Eric O. Postma

Universiteit Maastricht, IKAT/Infonomics
P.O. Box 616, NL-6200 MD Maastricht
{p.spronck, kuyper, postma}@cs.unimaas.nl

Abstract

When applying evolutionary learning to tasks containing easy and hard instances, the evolved populations tend to be swamped by solutions to the easy instances. This paper proposes island-based evolutionary learning in an attempt to prevent the unbalanced treatment of easy and hard instances. The proposed evolutionary technique splits the population into a number of sub-populations, each of which is assigned to an island. Sub-populations are evolved using partial fitness functions on either easy or hard instances. After being evolved in isolation for some time, the sub-populations are merged and the resulting population is evolved on the overall fitness function. In this way, the easy and hard instances are treated on a par by the evolutionary-learning procedure. The island-based evolutionary approach is evaluated on a box-pushing task. We present and discuss our results in terms of effective evolutionary learning procedures. It is concluded that island-based evolutionary learning yields better results on hard instances than conventional evolutionary learning, although the overall results in our experiments achieved with the island-based approach do not outperform those achieved with conventional approaches. However, it is argued that focussing on hard instances and ignoring easy ones in an early stage of evolutionary learning can be beneficial for the final fitness results.

1 Introduction

Many tasks involve instances with a solution complexity ranging from easy to hard. Learning to solve these tasks requires solutions to be found for as many instances as possible, ranging from easy to hard. Applying evolutionary learning to tasks involving easy and hard instances is problematic because the evolutionary optimisation tends to favour the larger number of solutions to easy problems over the smaller number of solutions to hard problems. This paper aims at alleviating this problem by proposing an approach, inspired by parallel genetic algorithm techniques, called *island-based evolutionary learning*. This approach prevents the unbalanced treatment of easy and hard instances by splitting the original population into a number of sub-populations each of which is assigned to an "island." The island populations are evolved in isolation from each other. Each population is evolved on one of the instances. Subsequently, all sub-populations are merged and the resulting population is evolved on the original fitness function. In this way, solutions to easy and hard instances are equally represented in the population.

In order to assess the viability of the island-based evolutionary approach to yield solutions to both easy and hard instances, the approach is evaluated on a box-pushing task involving easy and hard starting positions. Before turning to an experimental study of island-based learning of box-pushing behaviour, we discuss the nature of the box-pushing task.

The box-pushing task was introduced by Lee, Hallam and Lund [4] and involves the pushing of a box between two walls. Pushing an object (in our case a circular box) between two walls [7] is an elementary behaviour that is relevant in, for instance, the game of robot soccer in which a ball has to be pushed towards the opponent's goal. Elementary behaviours, of which box-pushing is only an example, are believed to underlie more complex behaviours such as target following, navigation and object manipulation. Our research employs evolutionary robotics [1, 3] to optimise the box-pushing behaviour of specific neural-network topologies [8, 10]. The fitness function we apply for the box-pushing task uses several different starting positions for the robot and the box, defining the fitness as the mean result of all these starting positions. Our results revealed some starting positions to be easy and others to be hard. Therefore, the box-pushing task is an appropriate touchstone for island-based evolutionary learning.

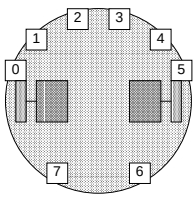
The outline of the paper is as follows. Section 2 describes the experimental procedure employed for evaluating the effectiveness of the island-based approach on evolving box-pushing behaviour. The experimental results are presented in section 3 and discussed in section 4. Finally, section 5 concludes and points at future work.

2 Experimental Procedure

2.1 The robot simulator and previous research

In our box-pushing studies we employ a publicly available mobile robot simulator based on the widely used mobile robot Khepera [5]. A schematic illustration of the robot is shown in figure 1. The (simulated) Khepera is equipped with eight proximity sensors and two motors. We augmented the simulator with a movable object, i.e., a circular box.

One of our studies focused on the type of fitness function to be used for successful box-pushing behaviour. Experiments revealed that a fitness function that combines a global fitness measure and an external fitness measure yields the best box-pushing results [7]. Two other studies focused on the appropriate neural network topology for box-pushing behaviour [8, 10]. The results of these studies showed recurrent networks (see section 2.3) augmented with edge-detecting inputs to outperform other non-recurrent networks or networks without edge-detecting inputs. The results from these previous experiments serve as starting point for the experiments described in this paper.



Neural controller inputs:

0 to 7: Sensor 0 to 7
8: Sensor 0 - 1
9: Sensor 1 - 2
10: Sensor 2 - 3
11: Sensor 3 - 4
12: Sensor 4 - 5
13: Sensor 6 - 7

Inputs 8 to 13 are the edge detectors.

Figure 1: Schematic top-view of a Khepera robot with eight infrared proximity sensors (the white squares) and two motors coupled to wheels (the large shaded rectangles). The sensors measure light intensity and proximity. In the present study, only the proximity measurements are used.

2.2 The simulation environment

All simulations described in this paper were performed using "Elegance", an environment for evolving neural controllers [9]. Elegance was originally developed to evolve neural controllers for simulated plants. It was adapted for application to neural-controller evolution for our experiments described in [10]. For the present research, Elegance was further augmented with support for island populations.

2.3 The neural controllers

A single neural-controller topology was selected for the present experiments: a layered recurrent network with four hidden nodes. This choice was made because this particular topology gave the best results in our earlier experiments [8, 10]. Figure 2 illustrates the topology. The layered recurrent network is less general than a completely recurrent network, because only recurrent connections within a layer are allowed.

The controller directs the two wheels of the robot. However, if a neural controller for one of the wheels is developed, the same network, with some of its inputs mirrored and a few signs switched, can be used to direct the other wheel. We exploited this mirror-symmetry of the control problem (as we did in our earlier experiments) by creating a network with two output nodes, one for each of the motors, while copying the appropriate connections and disallowing the superfluous ones. A linear transfer function was used to compute the output of the nodes. A sigmoid transfer function was used for the output node to map its values to the range [-10,+10]. Subsequently, the output value was rounded to the nearest integer value, as is required for controlling the motors of the simulated robot.

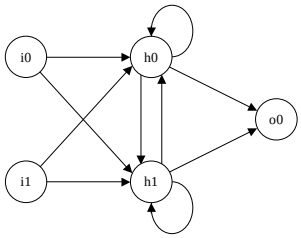


Figure 2: Illustration of a recurrent network of the type used in the experiments. All connections within the hidden layer are recurrent connections. The recurrent connections feed the node values from the previous time step into the target node. The network drawn here contains two input nodes (i0 and i1) and two hidden nodes (h0 and h1), as opposed to the the network used in the simulations, which contains fourteen input nodes and four hidden nodes.

2.4 The evolutionary algorithm

The evolutionary algorithm operates on the neural network that is represented by a chromosome consisting of an array of "connection genes." Each connection gene represents a single possible connection of the network and consists of a single bit coupled to a real number. The bit represents the presence or absence of a connection and the real value specifies the weight of the connection. The following six genetic operators are employed:

1. Uniform crossover,
2. Biased weight mutation [6] with a probability of 10% to change each weight, in a range of [-0.3,+0.3],
3. Biased nodes mutation [6], changing just one node within the same range as the biased weight mutation,

4. Nodes crossover [6] picking half the nodes of each parent,
5. Node existence mutation [9], with a probability of 95% to remove a node and a probability of 5% to completely activate a node, and
6. Connectivity mutation [9], with each connection having a 10% probability to flip.

During evolution, one of these operators was randomly selected, whereby uniform crossover and biased weight mutation were on average selected twice as often as the others. For the crossover operators, the best of the children was added to the population, and the other one removed. Thierens *et al.*'s [11] treatment of competing conventions was used to support the crossover operators. Newly generated individuals replaced existing individuals in the population, taking into account elitism. Crowding with a factor of 3 was used as replacement policy. For the selection process tournament selection with a tournament size of 2 was used.

We employ the "global-external" fitness measure that was shown to yield the best results in our earlier experiments [7]. The measure is based on the distance between the box at its start position and the box at its end position minus half the distance between the robot and the box at their end positions, after 100 time steps.

The nine different starting positions of varying complexity are shown in figure 3. The fitness is defined as the mean fitness over the nine starting positions. For fitter controllers, we averaged over multiple trials to reduce the effects of noise intrinsic to the robot simulator. Controllers with a fitness higher than 250 are averaged over 10 trials and the fittest controller in the (sub-)population is averaged over 100 trials.

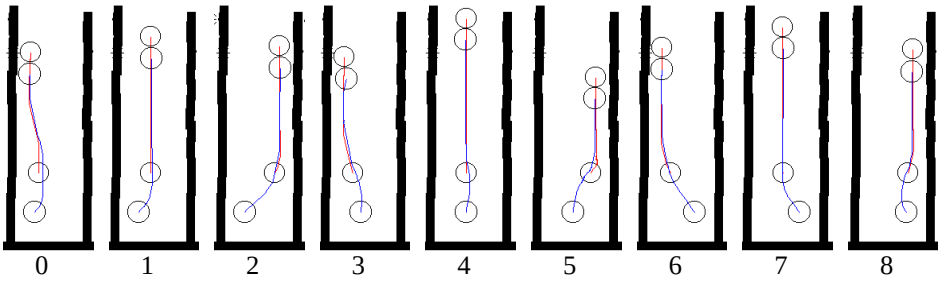


Figure 3: The nine starting positions (0 to 8) and typical trajectories of the experiments. The circles shown are the robot (large circles) and the circular box (small circles) in their starting (bottom) and ending (top) positions. The lines running from the starting to the ending positions represent the paths followed by the robot and the box. Note that the walls, especially the right one, are rough, making the task of sliding the box along it difficult due to friction.

2.5 Island populations

In our previous experiments [8, 10] we found that simple controllers that can cope with the instances associated with starting position 0, 4 and 8 are readily learned. These instances are easier to handle than the other six starting positions, because the path the robot has to travel from its starting position to the box is shortest for these instances, and the required box-pushing behaviour is fairly simple (being able to run forward is about the only requirement). Hence, a good controller for these three starting positions is assigned a higher fitness and is developed more easily than controllers for the other positions. The tendency of the easy instances associated with starting positions 0, 4 and 8 to overshadow the other starting positions makes our box-pushing task a perfect

candidate for the island-based evolutionary learning approach. The island-based approach entails two phases, which are described in detail below.

The first phase starts with the initialisation of nine small populations, each consisting of 25 randomly generated individuals. The networks in the starting populations are initialised by setting the "presence bit" of the connection gene to true for a random 50% of the connections. Subsequently, the associated connection weights are assigned random values within the range [-1,+1]. The nine islands use the same fitness measure, but each based on a single, different starting position. Each island generates new individuals using parents from the island's own population. The evolution on an island stops when one of its individuals exceeds a fitness value of 325 or when a total number of 475 new individuals are generated.

At the start of the second phase, the best ten individuals of each of the island's populations are selected and merged into a new population. For all the 90 individuals in this population the fitness is recalculated, now using the overall fitness measure which takes into account all nine starting positions. After that, the evolution process continues until 1800 new individuals are generated or an individual with a fitness exceeding 325 is found.

The approach described here is inspired by techniques used to run evolutionary algorithms on parallel processors [2]. The main difference is that with these parallel techniques, the fitness function for each of the islands is the same, while in our experiments the islands have different fitness functions (namely with different starting positions).

3 Results

The results averaged over eight runs of the experiments as described in the previous section are listed in table 1. The table also contains the results of experiments without islands (columns A and B). Below, we discuss the results in detail.

We found, quite to our surprise, that in most experiments the best performing individuals after merging the sub-populations tended to be those evolved on starting position 5 (see column D in table 1). This position proved to be the most difficult to control in our previous studies [10]. Apparently, the controllers that cope successfully with the hardest instance (i.e., starting position 5), are also able to handle the other instances (i.e., the other starting positions).

Just after the merge, four out of eight experiments had an individual generated by the island responsible for starting position 5 at the top of the population, which sometimes led to this island being the sole source for the genetic material in the final population. In three of the remaining four experiments, at least one individual from this island belonged to the five fittest individuals. In addition, controllers evolved on the island responsible for position 7 often ended up very high in the population, as did the controllers of the island for position 1. The islands associated with evolving controllers for starting positions 1, 5 and 7 formed the major source for the genetic material for the final population in almost all experiments. Sporadically, islands associated with the positions 2, 3 and 6 contributed significantly to the merged population. The islands associated with positions 0, 4 and 8, however, mainly populated the bottom of the merged population and therefore contributed little to the final results.

	A	B	C	D	E	F
	Experiments without islands		Mean fitness of islands		Experiments with islands (final results)	
Starting position	**Mean fitness**	St. dev.	End of phase 1	Start of phase 2	**Mean fitness**	St. dev.
Position 0	**351.0**	8.4	351.7	133.0	**357.1**	9.7
Position 1	**335.1**	8.4	332.7	191.4	**346.0**	7.3
Position 2	**250.0**	11.1	213.4	143.4	**244.8**	11.3
Position 3	**308.0**	24.2	256.9	158.0	**294.2**	50.4
Position 4	**374.8**	5.6	354.8	65.4	**367.4**	5.5
Position 5	**231.2**	22.0	236.4	237.8	**253.5**	17.1
Position 6	**284.3**	46.0	196.0	139.6	**268.5**	73.7
Position 7	**339.2**	6.5	324.5	207.9	**344.0**	6.1
Position 8	**348.2**	14.9	356.2	92.0	**338.2**	34.3
Overall	**313.5**	51.9			**312.6**	55.8

Table 1: Mean values of the results of the best controllers over eight evolution runs. The meaning of the columns is as follows:

Column A: The mean fitness of the best individuals calculated over eight evolution runs, in an experiment without islands. The results differ slightly from those reported in [10], because we repeated those experiments to rule out influences of small changes in the environment caused by our adaptations to Elegance. The setup for these experiments was the same as for the second phase of the island-based evolution, except that a total of 4410 new individuals were generated.

Column B: The standard deviations belonging to column A.

Column C: The mean fitness of the best individuals calculated over eight evolution runs for each of the islands, at the end of phase 1. Note that the fact that these results are generally lower than those reported in column A is mainly caused by the fact that the evolution run until the merging of the islands is very short, the size of the population on each of the islands is small, and the evolution is stopped when a target fitness of 325 has been reached.

Column D: The mean fitness of the best individuals calculated over eight evolution runs for each of the islands, at the start of phase 2. This is *after* the fitness has been recalculated for *all* starting positions, instead of the single starting position that was used for the island evolution. This column therefore indicates how suitable the starting position represented by each of the islands is for evolving a solution which works well on all starting positions.

Column E: The mean fitness of the best individuals calculated over eight evolution runs, in an experiment with islands.

Column F: The standard deviations belonging to column E.

We ran some preliminary experiments to remedy the tendency of the top of the population to contain genetic material from only a limited number of sources. In these experiments during the first phase of the evolution, newly-generated individuals on one of the islands had a 5% chance of "hopping" to another island. Although the island-hopping results yielded more diverse genetic material in the top of the population after the merge, the final results were not significantly different from the regular experiments. Comparing the final fitness values from the experiments without and with islands (table 1, columns A and E, respectively), we see that starting positions 0, 1, 5 and 7 yield an overall higher fitness in the experiments with islands, whereas the other fitness values are lower. The results for starting positions 1, 5 and 7 can be easily explained.

Controllers evolved on these starting positions formed the majority of the genetic material in the merged population, in most experiments. As a consequence, the characteristics required to deal with these starting positions were abundantly present in the final populations. Obviously, some of the other starting positions suffered from their controllers being underrepresented in the final populations. Our analysis shows that the island-based evolutionary learning approach has a considerable effect on the evolution process. Nevertheless, the final results do not differ significantly from the standard results (cf. the overall results in the bottom row of columns A and E in table 1).

4 Discussion

The results of our experiments clearly point at the importance of hard instances for early evolutionary learning. The sub-population of the island responsible for evolving controllers for the difficult starting position 5 plays a pivotal role in obtaining the best controllers in the second phase of the island-based evolution process. Our results seem to suggest that the early evolutionary learning of hard instances is beneficial for the final fitness of the population. This suggestion is borne out by some preliminary experiments that we ran. In these experiments, we first evolved a controller on starting position 5, which is the hardest. The evolution was permitted a very large number of runs. We thus obtained a very good controller for this starting position. Subsequently, the controller was submitted into a random population and evolved in the standard manner (without islands). Five repetitions of the experiment yielded very high fitness values, namely all between 319 and 326. These results are considerably better than those obtained in all our previous studies. Running the same experiments with starting positions 2 also gave better-than-standard results, though not as consistently as for starting position 5. Starting positions 3 and 7 gave about normal results. The other starting positions yielded worse or much worse performances. These results strongly support the suggestion that using more difficult starting positions as the basis for the evolution process yields better results. Moreover, they suggest that using easier starting positions actually is detrimental to the final fitness results. More detailed experiments are required to confirm the suggested importance of starting with hard instances.

5 Conclusions and future work

We conclude by stating that island-based evolutionary learning yields better results on hard instances than conventional evolutionary learning. Although the overall results achieved with the island-based approach in our experiments do not outperform those achieved with the conventional approach, the idea of separating out part of the population turned out to be fruitful. Our experimental results led us to an effective procedure to deal with tasks involving easy and hard instances. In an early phase of evolutionary learning, focussing on hard instances while ignoring the easy ones is beneficial for the later stage, in which individuals of the population are tested on all instances. Thus, for evolving a more general box-pushing controller, the difficult starting positions should be identified and the controller should be trained on a selection

of those. Our future work aims at exploring the benefits of selective evolutionary learning of hard instances in more detail.

References

[1] R. Arkin. *Behavior-based robotics*. MIT-press, Cambridge, 1998.

[2] D.E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesly Publishing Company, 1989.

[3] I. Harvey, P. Husbands, D. Cliff, A. Thompson and N. Jakobi. Evolutionary robotics: the sussex approach. *Robotics and autonomous systems*, 20:205-224, 1997.

[4] W-P. Lee, J. Hallam and H.H. Lund. Applying genetic programming to evolve behavior primitives and arbitrators for mobile robots. In *Proceedings of IEEE 4th International Conference on Evolutionary Computation*, IEEE Press, 1997.

[5] F. Mondada, E. Franzi and P. Jenne. Mobile robot miniaturisation: A tool for investigating in control algorithms. In T. Yoshikawa and F. Miyazaki, editors, *Proceedings of the Third International Symposium on Experimental Robotics*, pp. 501-513. Springer-Verlag, Berlin, 1993.

[6] D. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pp. 762-767, Morgan Kaufman, California, 1989.

[7] I.G. Sprinkhuizen-Kuyper, R. Kortmann and E.O. Postma. Fitness functions for evolving box-pushing behaviour. In A. van den Bosch and H. Weigand, editors, *Proceedings of the Twelfth Belgium-Netherlands Artificial Intelligence Conference*, pp. 275-282, 2000.

[8] I. Sprinkhuizen-Kuyper, E.O. Postma and R. Kortmann, Evolutionary Learning of a Robot Controller: Effect of Neural Network Topology. In A. Feelers, editor, *Proceedings of the Tenth Belgium-Dutch Conference on Machine Learning (BENELEARN'01)*, pp. 55-60, 2001.

[9] P. Spronck and E. Kerckhoffs. Using genetic algorithms to design neural reinforcement controllers for simulated plants. In A. Kaylan and A. Lehmann, editors, *Proceedings of the 11th European Simulation Conference*, pp. 292-299, 1997.

[10] P.H.M. Spronck, I.G. Sprinkhuizen-Kuyper, E.O. Postma. Evolutionary Learning of a Neural Robot Controller. In *Computational Intelligence for Modelling & Control, Proceedings of the CIMCA 2001*, Idea Group Publishing (To be published), 2001.

[11] D. Thierens, J. Suykens, J. Vandewalle and B. de Moor. Genetic Weight Optimization of a Feedforward Neural Network Controller. In R.F. Albrechts, C.R. Reeves and N.C. Steel, editors, *Artificial Neural Nets and Genetic Algorithms*, pp. 658-663, Springer-Verlag, New York, 1993.

The Khepera simulator described in section 2 is available from the WWW:
http://diwww.epfl.ch/lami/team/michel/khep-sim/.
The program "Elegance" is also available from the WWW:
http://www.cs.unimaas.nl/p.spronck/.