Infused Evolutionary Learning

Pieter SPRONCK Ida SPRINKHUIZEN-KUYPER Eric POSTMA
Universiteit Maastricht, IKAT/Infonomics
P.O. Box 616
NL-6200 MD Maastricht, The Netherlands,
p.spronck@cs.unimaas.nl

Abstract

In evolutionary learning of tasks containing easy and hard instances, the evolved populations tend to be swamped by solutions to the easy instances. This paper proposes infused evolutionary learning as an attempt to prevent the unbalanced treatment of easy and hard instances. In the proposed evolutionary technique initial populations are infused with a good solution to a single hard instance. The technique is evaluated on a box-pushing task, where the results show it to be beneficial to the performance. It is concluded that for box pushing infused evolutionary learning yields better results than conventional evolutionary learning.

Introduction

Many tasks involve instances with a solution complexity ranging from easy to hard. Learning to solve these heterogeneous-complexity tasks requires solutions to be found for as many instances as possible irrespective of their complexity. Applying standard evolutionary-learning techniques to heterogeneous-complexity tasks is problematic because the evolutionary optimisation favours a larger number of solutions to easy problem instances over a smaller number of solutions to hard ones.
In earlier experiments, we tried to alleviate this problem by using island-based evolutionary learning (Spronck *et al.* 2001). While this particular approach did not yield better results than obtained with conventional evolutionary learning, it inspired us to develop a new technique called *infused evolutionary learning*. In this approach the initial population is infused with a single solution to one of the hard problem instances. Conventional evolutionary learning (with a fitness function defined as the mean of the results of a number of problem instances) is then applied to the infused population. The infusion ensures that the genetic material needed to solve a hard instance is already present in the initial population. Subsequently, the evolutionary algorithm adapts the population to create an overall solution. To assess the effectiveness of infused evolutionary learning, it is evaluated on a box-pushing task involving easy and hard starting positions.
The outline of the paper is as follows. Section 1 analyses task learning and provides an intuitive explanation why infused evolutionary learning may create better solutions to heterogeneous-complexity tasks. Section 2 describes the experimental procedure employed for evaluating the effectiveness of infused evolutionary learning on the box-pushing task. The experimental results are presented in section 3. Section 4 discusses the approach and the results. We conclude by stating that infused evolutionary learning of heterogeneous-complexity tasks is successful, provided that the infused solution generalizes well to other problem instances.

1 Task learning

This section discusses four issues relating to task learning using evolutionary algorithms. First, it motivates the necessity of a global fitness measure for evolutionary task learning. Second, it identifies the combination of easy and hard instances as a potential problem for evolutionary algorithms. Third, it identifies and discusses similar forms of learning. Finally, it is argued why infused evolutionary learning might achieve better fitness than regular evolutionary techniques on task learning problems.

1.1 Necessity of a global fitness measure

Evolutionary algorithms have often been successfully applied to the creation of

controllers for diverse tasks. Having established the "best strategy" to deal with the task, the fitness of a controller can be determined at each of the constituent steps. However, in practice the best strategy is not known – if it were, it could be programmed directly and learning would not be necessary. Therefore, commonly the fitness of a controller can only be judged by a global fitness measure that indicates its performance on the task as a whole.

For many tasks, the detailed characteristics of the environment are unknown in advance. The evolution of a controller for such a task needs to take the unknown environment into account. By randomly generating task instances (environments) for each controller to be tested it is ensured that most situations are tested at some point. However, with randomly-generated instances evolutionary selection tends to favour lucky controllers over good controllers. Therefore, a better approach is to pre-select a number of specific task instances for the controller to work on and define the fitness as a function on the results achieved on those task instances (e.g., the mean fitness). Of course, care should be taken that these task instances form a good sample from the distribution of all instances, i.e., they cover all relevant aspects of the task as a whole.

In summary, evolutionary task learning requires a global fitness measure applied to a pre-selected number of relevant task instances.

1.2 Easy versus hard instances

In previous research, we have applied regular evolutionary techniques to task learning using task instances (Sprinkhuizen-Kuyper *et al.* 2000, 2001). We found some task instances to be more difficult to learn than others. Difficult task instances are associated with flat fitness landscapes with only a few peaks. For easy task instances peaks are higher and more numerous and therefore more likely to be found. As a case in point, in our earlier studies we noticed that solutions found to heterogeneous-complexity tasks tend to favour easy instances over difficult ones. Below we provide an intuitive explanation for this observation.

In a task for which the fitness function encompasses all instances, good controllers for easy instances are readily found. Individuals that solve an easy instance are quickly generated and assigned a high rank in the population. Even if they are incapable of dealing with all instances, obviously they outperform individuals that solve none of the task instances. From that point on the selection mechanism favours these individuals over the rest of the population. Ultimately, the evolution process converges to a solution that is biased to be successful on the easy task instances. Figure 1 provides an illustration of our explanation in terms of fitness landscapes for a hard instance (bottom), an easy instance (middle), and both instances (overall fitness, or the sum of the previous two). For the hard-instance fitness landscape, there is a single peak (indicated by 1 in figure 1). Settling on this fitness yields the best overall fitness (3), whereas one of the large number of good solutions for the easy-instance fitness landscape (the 2's) is easier to find (because there are so many of them) but yields lower overall fitness values.

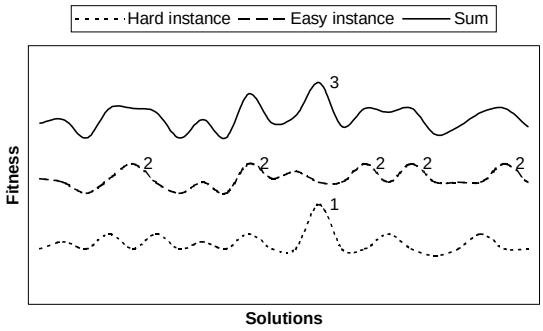

Figure 1: Illustration of the easy versus hard instances problem.

1.3 Similar forms of learning

Heterogeneous-complexity task learning is related to two other forms of learning: multi-objective evolutionary learning and constraint satisfaction learning.

In multi-objective evolutionary learning (Fonseca and Fleming 1995) a solution with multiple objectives has to be found. Our task instances correspond to the separate objectives in multi-objective learning. The main difference between these two forms of learning is that for multi-objective learning, the separate objectives are usually very different, whereas in task learning the objective is the same for each of the task instances. Therefore, with task learning problems, it is more than likely that a solution to one of the task instances also works well on some other task instances. This is especially true

for the solutions to hard instances because they often incorporate solutions to the easier instances. Therefore, techniques used for multi-objective learning are not a suitable approach to heterogeneous-complexity task learning.

Heterogeneous-complexity task learning is also related to constraint-satisfaction learning. Where task learning seeks a solution that solves a selection of (possibly) competing task instances, constraint satisfaction learning finds a solution that adheres to a selection of competing constraints. The main problem with constraint satisfaction learning is that as soon as several constraints have been satisfied, it is difficult to evolve a solution that satisfies the remaining constraints without violating the constraints already taken into account. For constraint-satisfaction learning several techniques to get around this problem have been proposed, such as COE (Co-evolution), SAW (Stepwise Adaptation of Weights) and MID (Micro-genetic Iterative Descent) (Eiben *et al*. 1998). Although some (but not all) of these techniques may be applied to heterogeneous task learning, they are probably too restrictive. In constraint satisfaction learning, *all* constraints must be satisfied *completely*, while in task learning satisfying the constraints (task instances) "reasonably well" is sufficient.

1.4 Infused evolutionary learning

Our first attempt to reach an overall better fitness on a task learning problem consisting of easy and hard task instances was *island-based evolutionary learning* (Spronck *et al*. 2001). With this technique, initially individuals are evolved on islands (isolated populations), whereby each island works on one of the task instances only. After some time, the best individuals of each of the islands are merged into one population, and the regular evolution process continues from thereon. This technique failed in achieving better overall results than we achieved previously with standard evolutionary algorithms, though it did manage to create solutions that solve the harder instances better at the cost of a small performance detriment on the easier instances.

These results inspired us to develop a new technique called *infused evolutionary learning*. In this technique, we infuse a randomly generated initial population with a very small number (usually just one) of the good solutions to the harder task instances. In this way, the evolution process is biased to evolve an overall solution in the neighbourhood of a peak in the fitness function of the harder task instances, without being forced towards the peaks in the fitness functions for the easier task instances. Note that infusion with a previously generated good solution won't work in evolutionary algorithms with a singular task because the process would simply get stuck in an already reached local optimum. We tested infused evolutionary learning on a box-pushing task.

2 Experimental procedure

This section discusses the box-pushing task, the design of the controller and the evolutionary algorithm employed to evolve the controller.

2.1 Box-pushing

The box-pushing task was introduced by Lee, Hallam and Lund (1997) and involves the pushing of a box between two walls. Pushing an object (in our case a circular box) between two walls 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. The task is non-trivial, because it needs constant adjustment from the robot since noise in its inputs and control mechanism prohibit it from taking a fixed position behind the ball and go forward in a straight line from there. Elementary behaviours, of which the box-pushing task is only an example, are believed to underlie more complex behaviours such as target following, navigation and object manipulation.

Our research employs evolutionary robotics (Arkin 1998) to optimise the box-pushing behaviour of specific neural-network topologies. 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 (Sprinkhuizen-Kuyper *et al.* 2000). Our results revealed some starting positions to be easy and others to be hard. Therefore, the box-pushing task is an appropriate touchstone for infused evolutionary learning.

In our box-pushing studies we employ a publicly available mobile robot simulator based on the widely used mobile robot Khepera (Mondada *et al.* 1993). The (simulated) Khepera is equipped with eight proximity sensors and two motors,

one for each of the wheels. We augmented the simulator with a movable object, i.e., a circular box. The simulator was ported to "Elegance", an environment for evolving neural controllers, which is especially suited to experiment with different set-ups for the evolutionary algorithm (Spronck and Kerckhoffs 1997).

2.2 The controller

A layered recurrent neural network topology with four hidden nodes was selected for the present experiments. This choice was made because this particular topology gave the best results in our earlier experiments (Sprinkhuizen-Kuyper *et al.* 2001). 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.

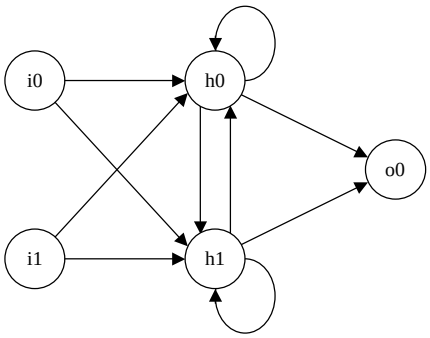

Figure 2: Illustration of a recurrent network of the type used in the experiments. All connections within the hidden layer are recurrent connections. The recurrent connections feed the node values from the previous time step into the target node. The network drawn here contains two input nodes (i0 and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

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 reversed, can be used to direct the other wheel. We exploited the 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.

The network has fourteen input nodes. Eight of these are used to input the proximity values received by the robot sensors. The six remaining ones are assigned the differences between neighbouring proximity sensors (the robot has six sensors at the front, which leads to five differences, and two sensors at the back, which leads to the sixth difference). We call these six extra inputs "edge detectors", because a high input value on one of them indicates an edge of some kind in its line of sight. Although the extra inputs are redundant, we found them to be beneficial to the evolution process (Sprinkhuizen-Kuyper *et al.* 2001).

2.3 The evolutionary algorithm

In the evolutionary algorithm the neural network 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 (Montana and Davis 1989) with a probability of 5% to change each weight, in a range of [-0.3,+0.3],
3. Biased nodes mutation (Montana and Davis 1989), changing all the inputs of just one of the neural network nodes within the same range as the biased weight mutation,
4. Nodes crossover (Montana and Davis 1989) picking for each child half the neural nodes (including their input connections) of each parent,
5. Node existence mutation (Spronck and Kerckhoffs 1997), with a probability of 95% to remove a neural node (by disconnecting all incoming and all outgoing connections) and a probability of 5% to completely activate a node (by connecting all possible incoming and all possible outgoing connections), and

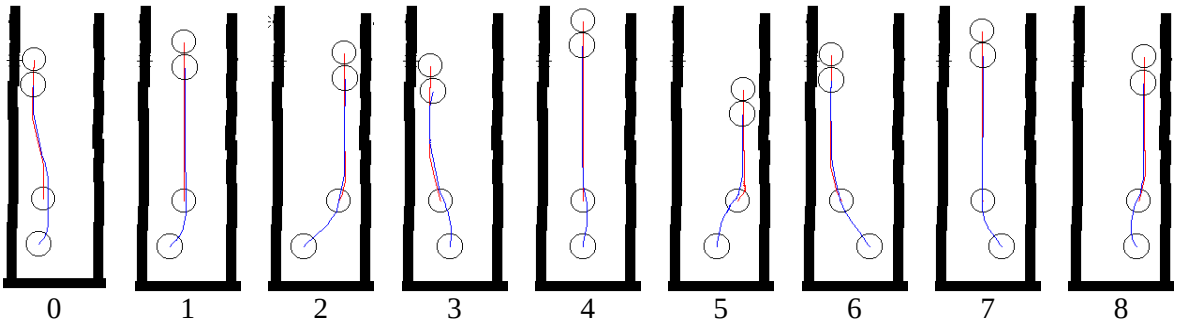

Figure 3: The nine starting positions (0 to 8) and typical trajectories of the experiments. The circles shown are the robot (large circles) and the circular box (small circles) in their starting (bottom) and ending (top) positions. The lines running from the starting to the ending positions represent 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.

6. Connectivity mutation (Spronck and Kerckhoffs 1997), with each connection having a 5% probability to switch from being connected to being unconnected and vice versa.

During evolution, one of these operators was randomly selected. For the crossover operators, the best of the children was added to the population, and the other one removed. Thierens *et al.*'s (1993) 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. The population size was 100, and the evolution was allowed to continue for 35 generations.

We employ the fitness measure that was shown to yield the best results in our earlier experiments (Sprinkhuizen-Kuyper *et al.* 2000). 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.

We used nine different starting positions of varying complexity. They are illustrated 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. Below a fitness of 250, a coarse indication of fitness is enough to rank an individual, and just one experiment is sufficient for that. Over a fitness of 250 we need a more reliable measure because differences between individuals become subtler, and therefore we use the average of 10 experiments. Since our aim is to replace the individual of the population that exhibits the best performance, we determine the fitness of the potentially best individual by averaging over 100 experiments to obtain a reliable estimate. Preliminary experiments showed that the standard error of the mean (Cohen 1995) for evolved controllers with 100 experiments is about 1.3, so the results have an accuracy of about 2.5 fitness points with 95% reliability.

Previous experiments showed that the easiest task instances are starting positions 0, 4 and 8. Positions 1 and 7 are slightly more difficult, 2 and 6 are still more difficult, and 3 and 5 are the hardest. It might seems surprising that starting positions 3 and 5 are harder than, for instance, positions 2 and 6, since positions 2 and 6 place the robot and the box furthest apart. However, it was shown that starting positions 3 and 5 suffer more from the roughness of the walls than the other positions, presumably because of the angle under which the robot first tends to push the box against the nearby wall.

2.4 Infusion

To test the effectiveness of infusion, good solutions for each of the starting positions were evolved using the evolutionary algorithm described in section 2.3. We allowed 75 (instead of 35) generations to be created and (obviously) the fitness was defined for a single starting position only. We then ran a number of experiments as described above (with 35

generations), whereby for each initial population one of the winning individuals on an isolated starting position was inserted. For each of these, we ran five or six experiments.

For comparison we ran six experiments where no infusion took place. We also experimented with infusion in one population of two pre-generated individuals for the two hardest starting positions; with infusion in one population of all nine pre-generated individuals; and with several different versions of the best individual for starting position 5. The results of all these experiments are presented in the next section.

It is important to remark that our experiment differs from standard machine learning experiments in the sense that our aim is to determine the effect of infusion rather than to determine generalization performance. In the latter case, inclusion of the starting position on which the algorithm is trained in the test data would give a positively biased and wrong indication of generalization performance. Our results should therefore be interpreted in terms of overall task performance due to infusion, rather than in terms of generalization performance.

3 Results

The results for the experiments with infusion of a single starting position, without infusion, and with infusion of all starting positions are graphically compared in figure 4. The straightforward experiment without infusion yielded similar results as in our previous experiments (Spronck *et al.* 2001). The average fitness settled at 311. The average fitness for all of the experiments with the infused populations settled around the same fitness value, except for those infused with a winning individual for one of the two hardest starting positions 3 and 5. Infusion with starting position 3 yielded an average fitness of 320, and infusion with starting position 5 an average fitness of 318. These are significant improvements over our previous results. Furthermore, the very best result over all experiments was achieved with an infusion of starting position 5, namely a fitness of 323.

Infusion with all nine starting positions led to an average fitness of 316. This is not a significant difference with the results achieved for infusion with starting positions 3 or 5 (especially not since for this experiment also an individual with a fitness of 323 was found). The same holds for infusion with both starting position 3 and 5 in one population.

To test whether infusion with a hard starting position always manages to achieve these high fitness results, we generated four different versions of a winning controller for starting position 5. All of these had about equal, high fitness ratings on the isolated starting position. Four repetitions of experiments with infusion of each of these individuals were executed. In three experiments the final result was equal to what we found in our first test with infusion of starting position 5. The fourth, however, achieved only the same average results we achieved in our experiments without infusion.

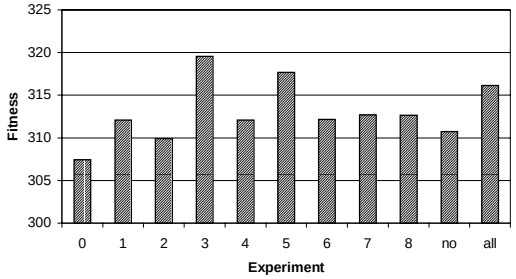

Figure 4: Fitness values of experiments with infusion of a single starting position ("0" to "8"), without infusion ("no") and with infusion of all starting positions ("all").

4 Discussion

The results show that for the box-pushing task infusion of an initial population with a winning individual that has been evolved on one of the hard task instances, generally leads to better overall results than an evolution without infusion. Of course, it is unclear whether these results can be generalized over other tasks. Furthermore, it should be noted that the results of the experiments with alternative versions of a winning individual for starting position 5 show that a fitness improvement over experiments without infusion is not guaranteed, since it is possible to infuse an "unlucky" individual.

In section 1, we argued that infused evolutionary learning might be a viable technique because it allows the algorithm to find a solution allocated near a peak of the hard–instance fitness landscape, which otherwise would be almost certainly ignored. It may be questioned whether this is the correct explanation for the effectiveness of infused evolutionary learning. Some support for the explanation can be seen in

the results shown in figure 5. When comparing the development of fitness on an experiment with infusion of starting position 5, and an experiment without infusion a clear difference is observed. With infusion of an individual that solved starting position 5 (the upper graph in figure 5), the fitness of the best individual in the population starts between 200 and 250. Within one or two generations, the fitness jumps to around 300. After that, fitness slowly grows towards a value around 320. Without infusion (the lower graph in figure 5), fitness starts anywhere between 0 and 200. At first the fitness increases quickly with large jumps to a value between 200 and 250. After that, the fitness increases only slowly towards a fitness value of around 310.

These different patterns of development, in particular the quick jumps at the start of the infused evolution process, indicate the infused evolution process to adapt the best available solution (in this case the one to starting position 5) to handle the other starting positions reasonably well. This is not the case in the process without infusion, because there the process reverts to small steps at a much lower maximum fitness value.

Further support for our explanation of the effectiveness of infused evolution is that in earlier box-pushing experiments we noted that solutions to the harder instances are better able to also handle the other task instances than the solutions to the easier instances (Spronck *et al.* 2001). While this observation supports our explanation, it indicates that the infusion technique requires the solution to be infused in the initial population to be able to generalise sufficiently well to other task instances.

It may be argued that a simple hill climbing algorithm starting with the infused solution would work just as well as our infused evolutionary learning. Whether this is true depends on the problem at hand – hill climbing does not work for our box-pushing task, because there is too much noise in the simulation to reliably determine the local fitness gradient. However, for noiseless problems hill climbing might indeed be a viable alternative.

Figure 5 also shows that the evolution of an infused experiment is much faster than evolution without infusion. Of course, before we can run an experiment with infusion, we first need to run an experiment to evolve an individual that can be infused. However, such a preparing experiment needs only 20-25% of the time a regular experiment needs, so in the end as an additional boon an experiment with infusion of a hard starting position not only gives better results, but also runs faster than a regular experiment.

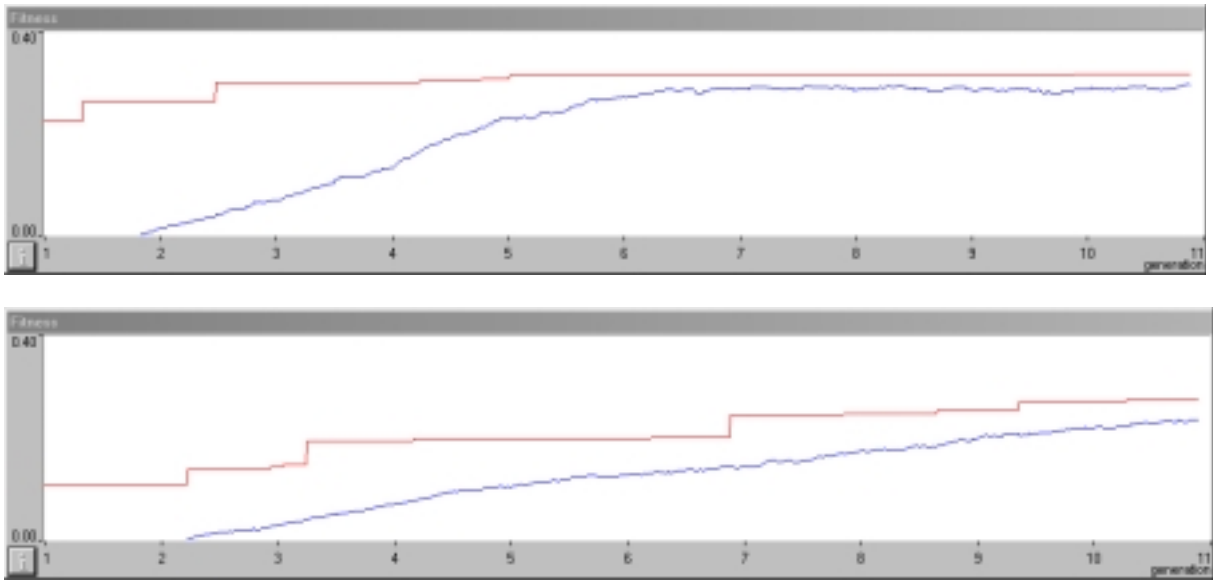

Figure 5: Two graphs showing typical developments of fitness for infused (top) and non-infused (bottom) evolutionary learning. The top curve in each of the graphs shows the maximum fitness in the population, the curve below it the average fitness. Multiply the function values by 1000 to get the actual fitness.

Conclusion

In heterogeneous-complexity task learning such as box pushing, infusion leads to better task solutions. We therefore conclude that infused evolutionary learning outperforms standard evolutionary learning provided that the infused solution to one instance performs well on the other instances.

In our future work, we intend to apply and assess the effectiveness of infused evolutionary learning on other tasks. In addition, we will compare infused evolutionary learning with regular hill climbing techniques in noise-free tasks.

References

R. Arkin. (1998) *Behavior-based robotics*. MIT-press, Cambridge.

P.R. Cohen (1995) *Empirical Methods for Artificial Intelligence*. MIT Press.

A.E. Eiben, J.I. van Hemert, E. Marchiori, and A.G. Steenbeek (1998) *Solving binary constraint satisfaction problems using evolutionary algorithms with an adaptive fitness function*. In "Proceedings of the 5th Conference on Parallel Problem Solving from Nature, LNCS 1498", A.E. Eiben, Th. Bäck, M. Schoenauer & H.-P. Schwefel, eds., Springer, Berlin, pp. 196-205.

C. M. Fonseca and P.J. Fleming (1995) *An Overview of Evolutionary Algorithms in Multiobjective Optimization*. In "Evolutionary Computation", 3(1):1-16.

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

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

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

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

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

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

P. Spronck, I.G. Sprinkhuizen-Kuyper and E.O. Postma (2001) Island-based Evolutionary Learning. In "BNAIC 2001 Proceedings of the 13th Dutch-Belgian Artificial Intelligence Conference", Ben Kröse, Maarten de Rijke, Guus Schreiber & Maarten van Someren, eds., Universiteit van Amsterdam, pp. 441-448.

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

Software availability

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