Countering Poisonous Inputs with Memetic Neuroevolution

Julian Togelius[1], Tom Schaul[1], Jürgen Schmidhuber[1,2] and Faustino Gomez[1]

[1] IDSIA, Galleria 2, 6298 Manno-Lugano, Switzerland
[2] TU Munich, Boltzmannstr. 3, 85748 Garching, München, Germany
{julian, tom, juergen, tino}@idsia.ch

Abstract. Applied to certain problems, neuroevolution frequently gets stuck in local optima with very low fitness; in particular, this is true for some reinforcement learning problems where the input to the controller is a high-dimensional and/or ill-chosen state description. Evidently, some controller inputs are "poisonous", and their inclusion induce such local optima. Previously, we proposed the memetic climber, which evolves neural network topology and weights at different timescales, as a solution to this problem. In this paper, we further explore the memetic climber, and introduce its population-based counterpart: the *memetic ES*. We also explore which types of inputs are poisonous for two different reinforcement learning problems.

1 Introduction

It stands to reason that when applying evolution methods to reinforcement learning problems, providing more information to the controller rather than less should make the problem easier rather than harder to solve. Intuitively, if those parts of the state description that are actually necessary to solve the problem (e.g. the position of the agent relative to the goal) were available, then a sensible learning algorithm ought to disregard any redundant information (e.g. the position of an unrelated agent relative to the goal, a random variable, or relevant aspects of the system state represented in the wrong scale or frame of reference). This assumption is not challenged by most existing reinforcement learning benchmarks, since they provide the controller with only a few well-chosen variables as inputs.

For problems where the best state representation is not immediately obvious, the above assumption is often wrong. In many cases, providing extra information to the controller results in lower fitness. For example, Lucas and Togelius [1] found that removing an input representing an angle to a way point was necessary for successful navigation to evolve for an holonomic agent; Igel [2] found that the CMA algorithm found good pole-balancing controllers much faster when a bias input was removed; and in the domain of helicopter control, De Nardi et al. [3] found that the network controlling yaw and the network controlling the pitch and roll could not share any inputs, lest evolution never found good controllers. In these examples, the presence of certain "poisonous" irrelevant inputs induces local minima in the fitness landscape—evolution exploits the poisonous inputs

to quickly find controllers that score better than random but cannot be built upon to find full solutions. For evolutionary reinforcement learning to be useful in real-world problems where the best state description is not known in advance, we need algorithms that can identify those state variables that should be ignored. Such algorithms will likely operate on more than one timescale, with one process learning what to ignore and another process learning the policy.

In a recent paper, we introduced the *memetic climber* [4], a variation of the simple hill-climber that searches for neural network topology and weights at different time scales; each topology mutation is accepted only if it is better than its predecessor after a brief period of local search in weight space. We found that on a version of a simulated car racing task which used a carefully selected set of inputs, the memetic climber performed slightly better than a standard, non-memetic hill-climber. However, when extra, potentially useful, but redundant inputs were added, the standard hill climber failed to find good controllers, while the memetic climbers performed almost as well as with the smaller set on inputs. In other words, the memetic climbers learned which inputs to ignore.

A number of algorithms have been proposed that evolve both topologies and weights of neural networks at the same time (see [5] for an overview). Most of these are not memetic algorithms, and treat topology search and weight search as a single search process, on a single time scale. An exception is the EANT2 algorithm [6], which treats topology and weight search as separate but interdependent processes: it evolves topologies with a simple ES and weights with the CMA-ES. Memetic algorithms have previously been used to efficiently search the space of neural network weights; see [7] for an example.

This paper continues our exploration of when and why extra inputs thwart the learning of effective control policies, and how memetic search in weight and topology space can counter this phenomenon. There are three main objectives: (1) to investigate the effects of changing the number of local search steps per global mutation in the memetic climber, (2) to compare the effects of redundant inputs with and without information content (irrelevant state descriptions versus pure noise), and (3) to introduce a population-based version of the memetic climber, the *memetic ES*, and compare it with other evolutionary algorithms.

2 Neural Memetic Search Algorithms

In this section, we describe five memetic search algorithms for neural network weights and topologies. The first two, originally presented in [4], use a single search point, while the other three are memetic extensions to evolutionary strategies. All of the algorithms are used to search the space of *masked networks*: in addition to the connection weights, the network chromosomes contain a bit-mask with a bit for each connection that determines whether or not the corresponding connection is active in the network.

There are two types of mutation operations that are applied to the masked network representations:

– ***weight mutation*** adds values drawn from a Gaussian distribution to all of weights.

Algorithm 1: Memetic Climber (n,m)

```
1  INITIALIZE (champion)
2  f_champ ← EVALUATE (champion)
3  for i=1 to n do
4      contender ← champion
5      TOPOLOGYMUTATE (contender)
6      for j=1 to m do
7          f_cntder ← EVALUATE (contender)
8          subcontender ← contender
9          WEIGHTMUTATE (subcontender)
10         f_subcnt ← EVALUATE (subcontender)
11         if f_subcnt >= f_cntdr then
12             contender ← subcontender
13         end
14     end
15     f_cntder ← EVALUATE (contender)
16     if f_cntder >= f_champ then
17         champion ← contender
18     end
19 end
```

– ***topology mutation*** iterates over all bits in the mask, flipping any bit with probability p.

When and how often these two operations are used relative to each other, is the key feature that distinguishes the algorithms presented here.

2.1 Memetic climber

The memetic climber, described in Algorithm 1, can be considered two nested hillclimbers operating at different timescales, and in different search spaces: a slow search in topology space (the outer loop, lines 3-19), and a fast search in weight space (the inner loop, lines 6-14). The algorithm maintains a single candidate solution, the *champion*. Each "generation", a copy of the champion, *the contender*, is topology-mutated and then local search is performed in for m steps. The contender replaces the champion only if its fitness after local search is higher than or equal to that of the champion. The intuition behind this algorithm is that by using local weight search to refine new topologies it might be possible to mitigate the disruptive effect of topology mutation. This is related to the NEAT algorithm, which affords new topologies "innovation protection" [8].

2.2 Inverse Memetic Climber

The inverse memetic climber works in the same way as the memetic climber except that the two types of mutation are interchanged (i.e. swapping lines 5 and 9 in Algorithm 1): for every weight mutation, local search is done in topology space.

2.3 Memetic ES

The *memetic ES*, described in Algorithm 2, is one possible combination of the memetic climber and evolution strategies. At each generation, a small amount of local search is conducted in weight space for each individual in the population. The population is then sorted by fitness, and the least fit λ are replaced by copies of the better fit μ of the population. Finally, all the newly copied individuals are topology-mutated.

2.4 Inverse Memetic ES

Just as with the inverse memetic climber, the inverse memetic ES is identical to memetic ES except that the weight and topology mutations (lines 8 and 21) are swapped.

2.5 Memetic CMA-ES

This algorithm is a version of the memetic climber where the local search (lines 6-14, algorithm 1) uses *Covariance Matrix Adaption Evolution Strategy* (CMA-ES) instead of the simple hillclimber. CMA-ES is a method that adapts the covariance matrix of the problem variables in order to model the fitness landscape as a multivariate normal distribution that used to generate new search points (see [9] for a complete description of this algorithm). Our Memetic CMA-ES is related to the more complex EANT2 algorithm [6] which uses CMA-ES for weight search and standard ES to search topology.

3 Experiments

Two very different domains were chosen as testbeds for the memetic algorithms described above: *simplerace* and *non-markovian double pole balancing*. The standard set of controller inputs normally used in these domains provide sufficient information for the controller to solve the task. In order to evaluate how well the algorithms learn to ignore irrelevant or redundant information, experiments were conducted using three different input representations: (1) the *standard inputs*, (2) an *extended input set* consisting of the standard inputs plus a number of inputs accurately describing redundant or irrelevant aspects of the state, and (3) a *noise input set* consisting of the standard inputs and a number of normally distributed random variables.

To compare the memetic to the non-memetics approaches, experiments we also run for standard (non-memetic) $(5 + 5)$ and $(50 + 50)$ evolution strategies, and as a baseline two versions of random search: one which randomly generates masks and weight vectors, and one which only generates random weight vectors, with all mask bits set. Both versions the memetic and inverse memetic ES have a population size of 10.

The weight mutation for all methods used a Gaussian distribution with mean 0 and standard deviation 0.1 applied to all weights, and the probability of having

Algorithm 2: Memetic ES(μ,λ,n,m)

```
1  INITIALIZE (Population, μ + λ individuals)
2  for i=1 to n do
3      for j=1 to (μ + λ) do
4          contender ← COPY (Population[j])
5          f_cntndr ← EVALUATE (contender)
6          for k=1 to m do
7              subcontender ← contender
8              WEIGHTMUTATE (subcontender)
9              f_subcnt ← EVALUATE (subcontender)
10             if f_subcnt >= f_cntdr then
11                 contender ← subcontender
12             end
13         end
14         Population[j] ← contender
15         EVALUATE (Population[j])
16     end
17     PERMUTE (Population)
18     SORTONFITNESS (Population)
19     for j=μ to (μ + λ) do
20         population ← COPY (Population[j-λ])
21         TOPOLOGYMUTATE (population[j])
22     end
23 end
```

a bit flipped, p, was set to 0.05 for the topology mutation operator. For the memetic algorithms, all mask bits are initially unset. These two operators are described in section 2.

3.1 Setup: Simulated Race Car Driving

The *simplerace* problem involves driving a car in a simple racing simulation in order to reach as many randomly placed waypoints as possible in a limited amount of time. In addition, the driver must decide which waypoint to target in order to beat an opponent car that tries to reach the same waypoints, giving the game a strategic element. The game has previously been used as a benchmark problem in several papers, and in two competitions associated with recent conferences[3].

For this domain, the standard input set consists of eight values: a bias term, the speed of the car, angle and distance to the current and next way point and to the next vehicle. The extended input set consists of the positions of both the controlled and the opponent car in Cartesian space, the speed of the opponent car, and the orientation and angular velocity of the controlled car in Cartesian coordinates. This information, while correct, should be significantly harder than the core inputs to interpret, due to the need for coordinate transformations. The

[3] A description of the problem is available in [10], and source code can be downloaded from http://julian.togelius.com/cec2007competition.

Table 1. Results for Memetic Climber with different numbers of local, weight search steps. Best fitness found after 20000 episodes for *simplerace*, averaged over 50 runs.

Local steps	Standard	Extra Cartesian	Extra random
2	13.83	12.81	13.27
5	13.79	12.71	13.12
10	**13.85**	**13.06**	**13.33**
25	13.51	12.90	13.15
50	13.86	11.66	12.96
100	13.46	9.50	12.76
250	13.25	7.80	11.74
500	11.19	6.99	10.65

noise input set are seven inputs that are set to independent values drawn from a Gaussian distribution with mean 0 and standard deviation 1.

Each algorithms was run 50 times, and 20,000 multilayer perceptrons (MLPs) with the *tanh* transfer function and six hidden units were evaluated in each run. For the memetic climber, eight different settings $(2, 5, 10, 25, 50, 100, 250, 500)$ for the number of local search steps, n, (weight mutations) per global (topology mutations) were tried.

3.2 Results: Simulated Race Car Driving

From Table 1, we can see that when the number of local steps per global mutation is low, the memetic climber finds good controllers under all three input conditions; for the *simplerace* problem, the best setting seems to be 10 local search steps, though everything under 50 is good. When using hundreds of search steps, worse solutions are found under all input conditions, though this effect is much more marked under the extra Cartesian input condition.

In Table 2 the results for the best memetic climber configuration are compared with a number of other search algorithms. The most striking result is that for every algorithm, the controllers found using standard inputs are better than those found using the extra random inputs, which in turn are better than those found using the extra Cartesian inputs. These differences can be minor, as for the memetic climber, or drastic, as for the $(50 + 50)$ ES.

None of the algorithms that only search weight space manage to find good controllers using the extra Cartesian inputs, e.g. the $(50 + 50)$ ES finds the best controllers (probably close to the optimum for reactive controllers) for the standard inputs, but performs extremely poorly with the extra Cartesian inputs. This effect can be seen even for random search, where random search in weight space performs worse than random search in topology and weight space under the standard condition, much worse under the extra random condition, and very much worse under the extra Cartesian condition.

Using the standard and extra random random, all the population-based algorithms outperform all non-population-based algorithms. With standard inputs, the ESs slightly outperform the memetic ESs, and under the extra random con-

Table 2. Results for *simplerace* task. Best fitness found after 20000 episodes for *simplerace*, averaged over 50 runs.

Algorithm	Standard	Extra Cartesian	Extra random
Random search (mask/weights)	13.44	9.08	11.15
Random search (weights only)	10.66	1.18	7.36
Hillclimber	12.82	0.56	10.78
Memetic climber 10-local	13.85	13.06	13.33
Inverse memetic climber 10-local	13.85	12.76	13.52
(5+5) ES	15.47	0.94	13.93
(50+50) ES	**16.23**	1.30	14.14
(5+5) Memetic-ES	15.36	14.02	14.90
(5+5) Inverse Memetic-ES	15.45	**14.51**	**15.18**

dition the opposite is the case; however, under the extra Cartesian condition the difference is dramatic. Overall, the memetic ESs are the best algorithms of those compared for finding *simplerace* controllers. (The differences in performance between standard and inverse versions are rather small and unsystematic.)

3.3 Setup: Non-Markovian Double Pole Balancing

In this task, two poles, sitting side by side, hinged to a wheeled cart must be balanced simultaneously by applying a scalar force at regular intervals such that they are balanced indefinitely and the cart stays within the track boundaries. Unlike the standard inverted pendulum problem which is nearly linear around the unstable equilibrium point, the double pole system is highly non-linear due to the interacting between the poles. In addition, in this non-Markovian version, the controller only recieves three of the six state variable as standard input: the distance of the cart from the center of the track, and the angle of each pole from vertical. Since the velocity of the cart, and the angular velocities of the poles are not provided to the controller, it must compute them from previous inputs usin internal state (memory) in order to balance the poles (see [11] for equations of motion and system parameters).

For this problem, the extended input set consists of the four Cartesian coordinates of the tips of both poles. The noise input set consist of four Gaussian noise sources as in the *simplerace* setup. In order to compute the velocities, the controllers were represented by Elman-style simple recurrent neural networks with sigmoid transfer functions. Each algorithm was run 30 times, and each run lasted until the best network could balance the pole for $50,000$ time steps, or until $30,000$ networks had been evaluated, whichever occured first. For this problem, we also compare our results to CMA-ES.

3.4 Results: Non-Markovian Double Pole Balancing

Table 3 summarizes the results for the pole balancing experiments. The variance in the results of each algorithms for different sets of inputs is more pronounced on this task compared to *simplerace*. In fact, many of the algorithms fail to find

Table 3. Comparison of methods using different input sets for non-Markovian pole balancing task. For each type of input: the first column (evals.) is the average number of pole balancing attempts required to solve the task; the second column (%solved) indicates the percentage of the runs that were able to solve the task; the third column (fail. fit.) is the average final fitness for the unsuccessful runs. The number of local search steps per global mutation is 100 for the memetic climbers, 50 for memetic-ESs and 500 for the memetic CMA-ES. Each method was run 50 times.

Algorithm	**Standard**			**Extra Cartesian**			**Extra random**		
	evals.	%solved	fail. fit.	evals.	%solved	fail. fit.	evals.	%solved	fail. fit.
Random search	—	0	60	—	0	60	—	0	43
Random masks	—	0	56	—	0	60	—	0	49
Hillclimber	—	0	47	—	0	45	—	0	38
(5+5) ES	—	0	46	—	0	43	—	0	39
(50+50) ES	—	0	40	—	0	40	—	0	38
CMA-ES	3658	**90**	**736**	**3421**	**90**	**523**	—	0	92
Memetic CMA-ES	2223	39	497	21733	39	109	**29000**	**5**	162
Memetic climber	**1736**	17	444	8005	2	324	—	0	64
Inverse memetic	2900	2	76	—	0	109	—	0	47
Memetic-ES	1950	5	240	20500	5	147	—	0	**862**
Inv. Memetic-ES	—	—	175	—	0	112	—	0	51

controllers that can balance both poles for the required $50,000$ time steps. This is partly because, for run time reasons, we have chosen to cut off the search very early, at $30,000$ evaluations. This is roughly ten times more than the best algorithm needs, but might be too short for some other algorithms.

One result that immediately stands out is that CMA-ES (along with memetic CMA-ES) is much better than all of the other algorithms using both the standard and extra Cartesian inputs. This is to be expected as CMA-ES is one of the most efficient algorithms for this particular problem to date [11]. However, the performance of CMA-ES drops sharply under the extra random condition–from 90% to 0% successful runs.

In fact, the performance of all algorithms degrades dramatically when the noisy inputs are present. The only algorithms that perform better than random search in this case are the memetic CMA-ES (which sometimes solves the problem) and the memetic ES (which does not solve the problem, but has a relatively good average fitness for failed runs).

Even using the standard and extra Cartesian inputs, the normal memetic algorithms clearly outperform the non-memetic algorithms, including the ES (but not CMA-ES). The inverse memetic algorithms perform worse than the normal memetic algorithms under all conditions.

4 Discussion

Pole balancing and *simplerace* are quite different problems with apparently very different search spaces, for all three input conditions. For the *simplerace* problem, random search does relatively well, and the observed performance differences between algorithms is at the upper end of the fitness range. For pole balancing,

almost the opposite is the case: most algorithms spend a long time trying very unfit solutions and some, including the standard ES, do not perform noticeably better than random search within the allotted number of evaluations.

The search spaces of the two problems are also transformed in different ways by the added inputs. The extra Cartesian inputs make the *simplerace* problem hard to solve for algorithms that search only for weights, but have little effect on pole balancing. This is possibly due to the fact that the Cartesian inputs make the *simplerace* search space deceptive: algorithms easily find modestly fit solutions that depend on the extra inputs, but are far from truly solving the task. No such effect seems to exist for the extra Cartesian inputs for pole balancing.

Adding random extra inputs has little effect on the *simplerace* problem, while it completely transforms the pole balancing problem, making the search much harder for all tested algorithms. A simple explanation for this is that noise is more deleterious for an unstable system like the double pole balancer, so that the connections from the random inputs have to be masked off or made very close to zero in order to prevent them from perturbing the system. The *simplerace* environment, however, does not exhibit this kind of instability, so that steering the car is more robust to disturbances (i.e. each action can, in principle, correct for each disturbance without the system diverging).

For both problems, memetic search of topologies and weights was not only competitive with algorithms that searched only for weights using standard inputs, but also significantly outperformed them when extra "distracting" inputs were provided.

The decrease in evolvability in the presence of poisonous inputs is not due to the increased dimensionality of the input or search space. This is clear from the fact that, for both problems, evolvability decreased only very slightly in the presence of non-poisonous inputs, whereas the input dimensionality is identical. Further experiments (omitted due to space constraints) have shown that the effects of poisonous inputs persist when replacing MLPs with linear filters of much lower dimensionality. See also the experiments in chapter 7.2 of [10], which suggest that increasing the dimensionality of the search space (through increasing the size of the hidden layer) actually *increases* evolvability in *simplerace*.

In *simplerace*, random weight and topology search reached lower fitness when subjected to the noise inputs than under the other two conditions, but performance did not decrease as drastically as it did for the other search algorithms. This suggests that poisonous inputs decrease evolvability both by reducing the portion of the search space which contains good solutions, and by making the search space more deceptive. The poor performance of random search on pole-balancing prevents us from drawing similar conclusions for that domain.

It is plausible that other topology and weight-evolving algorithms are equally capable of countering poisonous inputs. If so, this provides an important argument for the use of such algorithms on problems where the best input representation is not yet known.

5 Conclusions

We have shown that two rather different control learning problems can be made much harder by adding irrelevant information to the controller inputs, albeit

which sorts of extra inputs were poisonous depended on the problem. This effect seems to be independent of the type of neural network used. We have also shown that these effects can be mitigated to a large extent by using algorithms that search topology and weight space separately; such algorithms could solve both benchmark problems in the presence of irrelevant inputs that made them unsolvable by the tested non-memetic algorithms. In particular, we have shown that the memetic climber can be extended to a population-based algorithm in the form of the memetic ES. This algorithm is competitive with other population-based algorithms even when the state description is well-selected and well-represented, and performs better than the memetic climber when presented with problematic irrelevant inputs. We expect these findings to be highly relevant for cases where evolutionary reinforcement learning is applied to novel, unanalyzed problems, for which the correct state description is unknown, so that the best approach is to feed the controller any information that might or might not be relevant. In other words, the sort of problems where you would expect evolutionary computation to be at its greatest advantage.

Acknowledgments

This research was supported in part by the Swiss National Science Foundation (SNF) grant number 200021-113364/1. Thanks to Dan Ashlock for suggesting the term "poisonous inputs."

References

1. Lucas, S.M., Togelius, J.: Point-to-point car racing: an initial study of evolution versus temporal difference learning. In: Proceedings of the IEEE Symposium on Computational Intelligence and Games. (2007)
2. Igel, C.: Neuroevolution for reinforcement learning using evolution strategies. In: Proceedings of the Congress on Evolutionary Computation (CEC). (2003)
3. De Nardi, R., Togelius, J., Holland, O., Lucas, S.M.: Evolution of neural networks for helicopter control: Why modularity matters. In: Proceedings of the IEEE Congress on Evolutionary Computation. (2006)
4. Togelius, J., Gomez, F., Schmidhuber, J.: Learning what to ignore: memetic climbing in weight and topology space. In: To be presented at the Congress on Evolutionary Computation (CEC). (2008)
5. Yao, X.: Evolving artificial neural networks. Proceedings.1447 **87**(9) (1999)
6. Siebel, N.T., Sommer, G.: Evolutionary reinforcement learning of artificial neural networks. International Journal of Hybrid Intelligent Systems **4**(3) (2007) 171–183
7. N. Krasnogor, A.A., Pacheco, J.: Memetic algorithms. In: Metaheuristics in Neural Networks Learning, Springer (2006) 225–247
8. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation **10**(2) (2002) 99–127
9. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation **9**(2) (2001) 159–195
10. Togelius, J.: Optimization, Imitation and Innovation: Computational Intelligence and Games. PhD thesis, Department of Computing and Electronic Systems, University of Essex, Colchester, UK (2007)

11. Gomez, F., Schmidhuber, J., Miikkulainen, R.: Accelerated neural evolution through cooperatively coevolved synapses. Journal of Machine Learning Research **9**(May) (2008) 937–965