Revisiting the Personal Satellite Assistant: Neuroevolution with a Modified Enforced Sub-Populations Algorithm

Boye A. Høverstad
Complex Adaptive Organically Inspired Systems Group (CAOS)
Department of Computer and Information Science
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
hoversta@idi.ntnu.no

ABSTRACT

This paper revisits the evolution of a neural controller for a simulated Personal Satellite Assistant (PSA) using the Enforced Sub-Populations (ESP) neuroevolutionary algorithm, as described by Sit et al. in 2005 [8]. ESP has previously been shown to be a very efficient algorithm for neuroevolution. As opposed to the original paper, we are not primarily concerned with the solutions discovered by the system, but rather with how ESP performs its evolutionary search; using the unstable PSA control task as a vehicle for fitness evaluation.

We propose several changes to the original ESP algorithm. Our experiments suggest that these improve both the internal consistency, and the success rate of the algorithm. We further analyze the ability of ESP to go beyond classic weight evolution. We compare our evolutionary results with those of a simple hill-climbing algorithm, and propose that improved heuristics for the modifications of network topology in ESP may be necessary to evolve increasingly complex and robust controllers.

Track: Real World Applications

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Learning; I.2.6 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search; I.2.9 [**Artificial Intelligence**]: Robotics

General Terms

Algorithms, Experimentation, Performance

Keywords

Neuroevolution, Enforced Sub-Populations, Coevolution

1. INTRODUCTION

In a recent paper, Sit and Miikkulainen described an experiment of applying the Enforced Sub-Populations (ESP) algorithm to the problem of evolving a neural network to control a rocket-driven robot in a simulated environment [8]. ESP is a highly efficient variation of evolutionary algorithms, dedicated to evolving neural networks.

The modelled robot was a Personal Satellite Assistant (PSA), designed to assist astronauts in space. This robot has very limited motor capabilities, and operates in a frictionless environment, making the control task very challenging. Sit et al. show how the evolutionary search is able to find several efficient and inventive solutions to basic control tasks.

This is a fascinating example of neuroevolution, showing how ESP can be used to successfully evolve a controller for a complex robotics task. However, the evolved neural controllers were evaluated using simplified, noise free simulation of the physics involved in guiding the PSA. This is in contrast to the guidelines for network evaluation previously laid out by Miikulainen et al. [1], and also greatly reduces the complexity of the search space in which the ESP operates.

This article repeats the experiment performed by Sit and Miikkulainen, using the freely available physics simulation environment *ODE* to model the behaviour of the PSA [5]. Experiments are carried out with and without noise added to the neural controller's sensors. The robustness and stability of the solutions found is then examined, by applying them to situations similar, but not necessarily identical, to those that were used during evolution. In general, it is found that the evolved solutions react poorly to even small amounts of noise.

ESP is a very creative and promising approach, but it needs some improvement. The main purpose of this paper is to propose certain changes to ESP, in order to make it more consistent and efficient. As will be described later, ESP evolves neurons, but evalutes networks, and thus needs a way to distribute the fitness of a network over its constituent neurons. An underlying motivation for investigating this aspect of the algorithm, is to improve our understanding of the dynamics of, and the interaction between, the neurons in artificial neural networks.

Although primarily a weight-evolving algorithm, ESP contains heuristics for adding and removing neurons to the currently evolved topology. Through repeated application of this heuristic, the algorithm is expected to find a near-optimal combination of network topology and weights. We

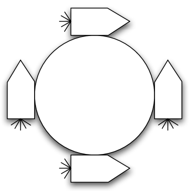

Figure 1: Schematic view of the two-dimensional model of the PSA. The thrusters are not drawn to scale, and are modelled with uniform mass distribution.

examine the details of this heuristic, and show that some modifications must be made for the system to work as intended.

Finally, the performance of the evolutionary system is compared to that of a simple hill-climbing methodology. ESP has previously compared favourably with several other neuroevolutionary techniques, including well-known methods such as SANE and Cellular Encoding [9],[1],[2]. It has subsequently been applied to evolve neural controllers for several non-linear control tasks in simulated environments. In our simulations, the original ESP algorithm shows surprisingly little performance gain over the hill-climbing algorithm.

2. EXPERIMENTAL SETUP

The task of the evolved neural networks is to control a model of a Personal Satellite Assistant (PSA) in a two-dimensional simulation, similar to that previously described by Sit et al [8]. The construction of the PSA is based on the descriptions given in the above paper, so only a rudimentary description will be given here.

The two-dimensional model of the PSA consist of a round body and four thrusters, mounted along the perimeter of the body, as shown in Figure 1. Two thrusters point toward the negative x-axis, and so may accelerate the PSA in the positive x-direction. Similarly, the two other thrusters point toward the negative y-axis, and may accelerate the PSA in the direction of increasing y.

By having only four thrusters, the PSA has effectively no brakes, and so it has no direct way of stopping its motion in certain directions. In order to stop a rightward motion, the controller must increase the output on one of its thrusters, in order to rotate the PSA, so that two of the thrusters point in the desired direction.

The design is deliberately limited, in order to minimize the risk of mechanical failure and the cost of maintenance. Having only 4 thrusters in a plane, rather than 8, requires a complex controller to successfully navigate the PSA.

The neural network is a sigmoidal feed-forward network with one hidden layer. It has 7 input units including a bias unit, and 4 output units. Input to the network consists of PSA position, velocity, heading and angular velocity. Each of the four output units controls the effect of one thruster.

2.1 Physical Simulation

Although an operational PSA would need a total of 8 thrusters, 4 in each plane, the evolutionary system searches for a simplified problem, that of controlling the PSA in two dimensions using 4 thrusters. Whereas the initial experiment by Sit et al was carried out with a custom physics model, the experiments described in this paper were carried out using a stock physical simulator, the Open Dynamics Engine (ODE) [5].

The PSA was modelled as a single physical body, with a mass distribution derived from the shape of the PSA. The thrusters and the body were assumed to have the same density, and each to have a uniform mass distribution. The mass distribution of the entire PSA was hence computed from the dimensions of the body and thrusters, as adapted from the Sit paper[1].

The action of the thrusters is modelled by adding a force proportional to the output of the corresponding thruster. The thruster is assumed to respond immediately to neural output, with maximum power of 1N.

A user-specified amount of Gaussian white noise may be added to input and output units separately. This models the thermal noise found in electrical conductors.

2.2 Control Tasks and Fitness Estimation

The controller is evolved to perform one of the three tasks specified by Sit et al:

1. Stop autorotation and maintain a given heading at an approximately constant position;
2. Turn 90 degrees and stop;
3. Given forward velocity, stop at a pre-defined position.

Fitness was initially assigned as the number of time steps in which certain constraints were satisfied, as suggested by Sit et al.

In task 1, the fitness of a network was the number of simulation time steps in which the PSA satisfied the following constraints:

- $|x| < 0.2$;
- $|y| < 0.2$;
- $|\theta| < 0.05$;

where x and y represent the coordinates of the PSA relative to its target position, and θ is the heading of the PSA. This fitness assignment would sporadically lead to good results, and to some controllers showing surprisingly efficient behaviour, but the evolutionary search more often stagnated at a very low fitness. Part of the reason for this may be the fact that the fitness assignment lacks a bias towards low angular velocity. Given zero *linear* velocity, and a satisfactory number of time steps, different *angular* velocities will produce the same distribution of sampled headings, given that the angular velocity is constant over the course of the simulation. In other words, as long as the position of the PSA does not change, lowering the angular velocity will not lead to a large increase in fitness; only a near zero angular velocity with the correct heading will have an effect. Given initial angular velocity, this means that there is a large local maximum found by networks that do nothing.

In order to overcome this problem, an alternative, incremental fitness assignment was used, in which the angular

[1]These mass distribution estimates are likely to be inaccurate, but as long as the mass distribution of the actual PSA is fairly regular, this is not likely to have a large impact on the search landscape.

velocity ω was taken into account. One point was awarded at every time step where the following conditions were met:

- $|x| < 0.2$;
- $|y| < 0.2$;
- $|\omega| < 0.05$ or $\omega_t < \omega_{t-1}$.

An additional three points were awarded whenever the last of the original three conditions was also met:

- $|\theta| < 0.05$.

3. NEUROEVOLUTION WITH ESP

ESP is a popular and relatively complex algorithm for evolving neural network weights and architectures, with a main focus on evolving neuron weights. It has been shown to be very efficient compared to several other neuroevolutionary techniques [1].

ESP works by defining a separate sub-population for each neuron in a neural network with a pre-defined architecture. In the simplest case, that of a feed-forward network with one hidden layer, a neuron is defined by its input and output weights, and a sub-population thus consist of a number of such neurons. Entire networks are then formed by sampling one neuron from each sub-population.

The evolutionary operators work at the neuronal level, whereas fitness estimation is performed on entire networks. Neurons are assigned a fitness based on the fitness of the networks in which they have taken part.

The motivation behind algorithms like ESP is the idea that good neural networks are made up of good neurons. The goal of the algorithm is therefore to evolve neurons that each perform a part of the desired operation well, and subsequently use these to form an efficient network. The structural division of each network neuron into separate sub-populations will presumably enforce a division of labour between the neurons, so that each sub-population will evolve neurons specializing in solving a separate part of the problem.

In this regard, ESP may be viewed as a cooperative coevolutionary algorithm, in which sub-populations are coevolved, and the neurons from each sub-population cooperate to form maximally efficient networks. ESP thus also exposes a problem present in several coevolutionary systems, namely that of estimating the fitness of each constituent in a system, given the fitness of the system as a whole. In this case, the task is to assign fitness to neurons based on fitness measured on the neural networks.

Since any permutation of the hidden neurons in a feed forward network will yield the same network functionality, a traditional weight-evolving algorithm which operates on the network level has to deal with a search space in which each neural architecture at the phenotypic level may be described by a vast amount of different genomes [6], [7]. This redundancy can to some extent be avoided with neuron-evolving methods such as ESP, since each sub-population may specialize on a specific type of neurons [4].

3.1 Fitness Evaluation

In order to evolve at the neuronal level, the fitness of each neuron needs to be estimated. Since evalutations can be performed on entire networks only, the fitness of a neuron must be derived from the fitness of the networks in which it participates. In ESP, this is done by repeatedly sampling neurons from each sub-population, thus creating a relatively large number of neural networks compared to the number of neurons in each sub-population, and assigning to each neuron the average fitness of the networks in which it has participated in the current generation.

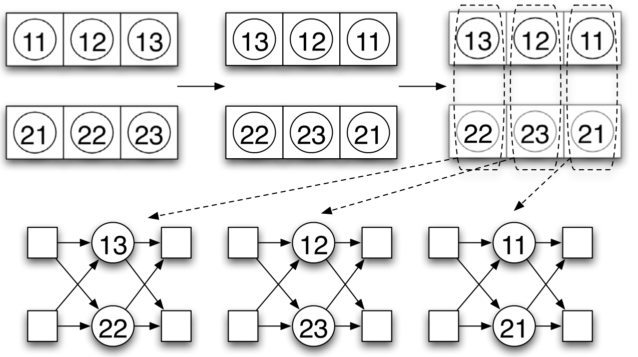

Figure 2: Overview of the alternative sampling method. The contents of the two sub-populations are shuffled, after which networks are constructed from neurons at corresponding positions in each sub-population.

In the current implementation, an alternative sampling method is proposed, as shown in Figure 2. Given a sub-population of size s, for each of the s networks to be generated, each sub-population is shuffled, so the order of the neurons is randomized, after which a network is generated by taking the nth neuron in each sub-population to form the nth network. This method creates networks by random combinations of one neuron from each sub-population, but at the same time guarantees that every neuron will be evaluated the same number of times.

A subsequent problem is the assignment of fitness to each neuron, given the fitnesses of the evaluated networks. In ESP, the fitness of a neuron is the average fitness of all the evaluated networks in which it has participated. This implicitly awards neurons that are able to perform well in many different network configurations. This strategy may disfavor neurons that specialize on a given subtask, since these may depend on cooperation with the remaining neurons in the network to achieve a high network fitness.

Assuming that a good neuron is one that performs a subtask of the problem well, it is still dependent on the remaining neurons to perform complimentary functions in order for the network as a whole to successfully fulfill its task. Optimally, the fitness of a neuron should hence reflect whether it participated in networks where this requirement was met.

In a similar vein, the ultimate goal of the search is to find an optimal network. This is by definition the network with the highest fitness, which is not necessarily identical to the network composed of the neurons with the highest average fitness.

Consequently, an alternative way of assigning network fitness to participating neurons has been tried, where the fitness of each neuron is the highest fitness achieved by any network in which it has participated.

3.2 Burst Mutation and Network Topology

The network topology may change in ESP during a simulation by a process of repeatedly adding or deleting sub-populations. This process is guided by the following heuristic. The fitness of the best network found during a generation is continuosly logged. If this maximum network fitness does not increase for a predefined number of generations, *stagnation* is said to occur. When the search stagnates, an operation called *burst mutation* is performed.

Burst mutation generates a new set of sub-populations based on the best network found so far, by adding noise drawn from a Cauchy distribution to each of the weights of the best network. The previous population is discarded in its entirety.

If, after a series of stagnations and burst mutations, maximum fitness still does not improve, a *lesion test* is carried out to determine whether the system should add or remove a sub-population [1].

A lesion test can be performed on a single network. The fitness of the network is evaluated repeatedly, each time with one of the neurons lesioned. A lesioned neuron is effectively disabled. By comparing the fitness of a lesioned network with that of the original, unlesioned network, the contribution of the neuron to the fitness of the network can be estimated.

If the fitness of a lesioned network does not drop below a certain threshold, the neuron is taken to be redundant, and so the corresponding sub-population is removed. On the other hand, if no neurons can be removed without a drastic drop in network fitness, a sub-population is added.

As pointed out by Gomez and Miikkulainen, burst mutation may be seen as "searching the space in a 'neighborhood' around the best previous solution." [1]. In the case where the current best is a good approximation of a local optimum, with no better proximal optima in terms of the mutation operator, there is only a small probability that the burst mutation will lead to fitness improvement. A fitness plot illustrating this situation is shown in Figure 3. Here, burst mutation has a transient negative impact on the overall fitness of the population, after which a network similar to the one previously used as the basis for the burst mutation is once again discovered. As an increasingly large fraction of the weight space is searched without improvement, it becomes likely that further improvements may instead require topological changes to the networks.

The original implementation of ESP is however dependent on quickly finding a new best solution after adding a sub-population. This is necessary for its topology modification heuristic of burst mutation and lesion testing to work correctly. In the case where the current best solution represents a local optimum, with no better proximal optima, neither in terms of network topology nor network weights, ESP is likely to get stuck.

This problem is most easily illustrated with an example. Suppose that the current best network contains 5 hidden neurons, with 5 corresponding sub-populations in ESP. After several rounds of burst mutation, best network fitness has not improved, and a lesion test is performed. Assume further that the lesion test finds no neuron to remove, so ESP decides a neuron should instead be added to the network. A new sub-population of randomly initialized neurons is created, so the system now contains 6 sub-populations, after which the 5 first sub-populations are wiped out and regenerated by burst mutation of the neurons in the best network. The discrepancy occurs if a better solution is still not found when the search again stagnates after adding a 6th neuron: The best network found so far contains only 5 neurons, but the system currently searches for 6-neuron networks. Since a lesion test can only be performed on a single network, the system has no way of estimating the utility of the 6th sub-population[2]. Given a deterministic fitness assignment, the next lesion test of the best network will give the same result, so another sub-population will be added. This process of adding sub-populations will continue indefinitely, unless a better network is found to replace the 5-neuron solution. At the same time, neither lesion testing nor burst mutation is applied to the additional sub-populations, since the best network found does not contain neurons from these sub-populations. Also, since the number of networks created is a function of sub-population size, not the number of sub-populations, as the number of sub-populations grows, each neuron will take part in a smaller fraction of the possible networks created by the neurons in the sub-populations, so the fitness evaluation of each neuron grows less and less reliable. This is likely to further decrease the probability of finding a new best solution.

A similar problem is present in the situation where a lesion test deems one of the neurons in the network superfluous, but a new best network is not found when searching the space of 4 sub-populations.

There are several possible solutions to this problem. One is to discard all sub-populations at each round of burst mutation. Referring to the example above, this amounts to erasing all 6 sub-populations after stagnation, after which a new lesion test is performed on the 5-neuron solution, a 6th sub-population is once again added, and burst mutation of the 5 first sub-populations is performed. This maintains pressure on all sub-populations in the system, but has the drawback that in order for ESP to search multiple topologies, for each number n of sub-populations, a network must

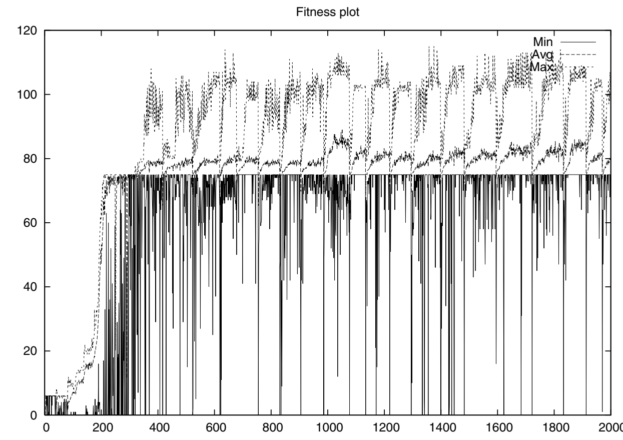

Figure 3: Characteristic example of repeated burst mutations without progress. Each burst mutation is characterized by a short-lived, drastic, drop in fitness, followed by a relatively rapid return to stagnation.

[2]Other than indirectly, in that adding it did not lead to an improved solution.

be found that has a better fitness than the best found with $n-1$ neurons. Unless noted otherwise, this alternative has been used for all the results obtained in the following section.

Another option is to discard the best network whenever a sub-population is added or removed. As a new best network will then necessarily be found, this guarantees that the best network always contains the same number of neurons as the ESP does sub-populations. This solution is somewhat contradictory to the philosophy of the burst mutation, where stagnation should be handled by modification of the best network found. On the other hand, it allows the ESP to search multiple topologies without the need for finding successively better solutions at each number of sub-populations. Further, disregarding topology modifications, it may allow the system to progressively move away from a local optimum, since the network used for burst mutation is continuously replaced.

4. HILL-CLIMBING SEARCH

As a comparison to the performance of the ESP, a simple, parallel, stochastic, hill-climbing search algorithm was designed. The algorithm searches the weight space of a neural network with a given topology.

The algorithm starts out by creating a single neural network with weights chosen randomly from the uniform range $(-6, 6)$. A set of candidate networks is created by repeated mutations of the initial network. The candidate networks are evaluated, and the best performing network is used as a basis for the next cycle of the algorithm[3].

Networks are mutated by stochastic application of two operations to each weight in the network. With a given probability, noise drawn from a Cauchy distribution of a specified range will be added to the weight. Then, with a given second probability, the sign of the weight will be changed. This second mutation operator has a marked effect on search efficiency.

Each of the the two mutation probabilities and the noise range described above may be annealed separately, by multiplying the current parameter value with a given constant at each cycle of the algorithm.

5. RESULTS

Several experiments were performed in order to compare and analyze the results of the different search methods. In addition to the three basic tasks, searches were performed with noise on the network input units. Further, for tasks 1 and 3, the initial conditions of the PSA were randomized at the beginning of each simulation.

Experiments were performed with combinations of the following variations on the original task:

- Noise on input signals;
- Randomized initial conditions of the PSA, where applicable.

Both search methods were applied, under similar conditions in terms of the number of network evaluations performed during a search. ESP was allowed to evolve for 2000

[3]An option similar to elitism was also tried, in which with the best of the candidates only replaced the base network if the former has a fitness better than or equal to that of the latter. This option had a negative impact on the search efficiency, and was not applied in the runs reported here.

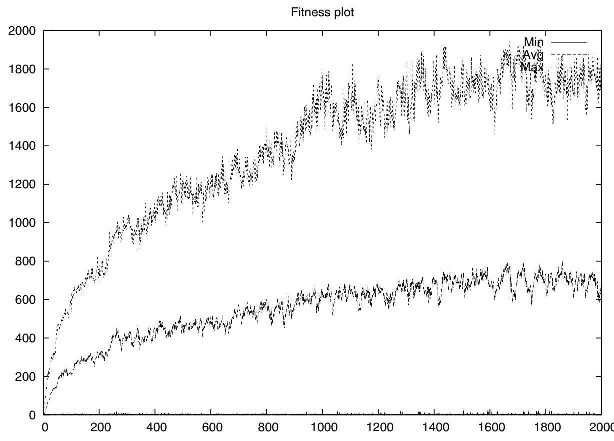

Figure 4: Fitness plot of ESP applied to task 1, showing minimum (barely above 0), average and maximum fitness, averaged over 30 runs.

Table 1: Average fitness scores of the best evolved controllers, in % of theoretical maximum, for tasks 1, 2 and 3, and averages across all three tasks.

Task	Average	Standard Deviation
1	42.4	12.8
2	44.0	20.7
3	11.9	8.1
All	**32.8**	**20.8**

generations, using 5 initial sub-populations, each of size 40. This amounts to 400 network evaluations per generation, not counting the lesion tests. Unless otherwise noted, all results refer to the average of 30 runs under identical conditions, but with different random seeds.

In order to investigate the quality of the solutions found by evolution the best networks were tested repeatedly under the various different experimental conditions described above.

The first results reported are from the original ESP implementation, except for the modification of the burst mutation, as described in section 3.2. In each of the following experiments, only the discussed parameter or aspect of the system is changed, in order to isolate the effect of the change relative to the initial settings.

5.1 General Search Efficiency

Figure 4 shows a typical fitness plot for the ESP algorithm, averaged over 30 repetitions of the same task with different random seeds. The search is quite noisy, but the overall fitness typically increases throughout the simulation. In Table 1, the average of the best networks found by the 30 repetitions of evolution on each task is reported. Fitness is given as the percentage of the theoretical maximum fitness assigned at each task. The table shows a large variation in the fitness achieved by the best solutions found in 2000 generations. In one case, the evolutionary search for a solution to task 2 has stagnated completely; the best network found in 2000 generations achieving only 1 fitness point out of 2000.

In order to estimate the robustness and generalization capabilities of the evolved controllers, each of the final net-

Table 2: Fitness scores, in % of theoretical maximum, for tasks 1, 2 and 3, with noise or random initialization.

Task	Noise	Average	Standard Deviation
1	Input	5.8	4.8
2	Input	2.5	3.1
3	Input	2.7	0.6
All	**Input**	**3.7**	**3.6**
1	Random init	3.9	5.9
3	Random init	7.8	6.1

Table 3: Fitness scores, in % of theoretical maximum, for tasks 1, 2 and 3, with noise or random initialization, after evolving under similar conditions.

Task	Noise	Average	Standard Deviation
1	Input	14.5	6.7
2	Input	10.0	6.1
3	Input	2.4	1.0
All	**Input**	**8.9**	**7.3**
1	Random init	10.0	9.9
3	Random init	6.2	3.0

works was re-evaluated under noisy conditions. First, each network was tested with noise added to the input signals to the network. Gaussian noise with a standard deviation of 0.05 was added to each input neuron. This produces noise levels about two orders of magnitude less than the sensory input levels of the network.

Each of the 30 best networks was tested 500 times, and its average fitness was recorded, for a total of 15000 evaluations.

Second, for tasks 1 and 3, generalization ability was tested by assigning the PSA a random initial position before each simulation. For task 1, this amounted to rotating the PSA by a random angle before simulation starts, whereas in task 3, the distance the PSA should travel before stopping was sampled uniformly from the range $[0.25 - 0.55]$ meters. Once again, each of the 30 best networks were given 500 tries, and their average fitness was recorded.

Table 2 shows the average and standard deviation of the fitnesses across all 30 networks of the above described evaluations. It is clear that the evolved controllers fail completely in the presence of even a small amount of noise. From the low fitness achieved when given random initialization, it appears that the evolved solutions are highly specialized; they do not generalize well to similar problems.

In order to assess the ability to evolve robust and general controllers, a second set of experiments was performed, in which noise and random initialization were present also during the fitness evaluation of each network during the evolutionary search. Each network was now evaluated three times during evolution, and the average fitness over the three runs was reported, so as to achieve a reasonable fitness estimate at a moderate runtime expence.

The results of testing robustness and generalization after evolving under similar are shown in Table 3. We see a marked improvement over the results shown in Table 2: For tasks 1 and 2, the average scores are roughly doubled, although the fitness values are still very low.

The solutions evolved for task 3, both with noise and random initialization, show the opposite effect. In general, the best evolved solutions had a very low fitness, indicating that when noise was present in the environment during evolution, the system was unable to make any progress. The quality of the best found controllers is therefore very low.

Visual inspection of the behaviour of the final networks reveals that evolution is able to find similar solutions to basic control tasks as those described in [8], in a different physics simulation environment. This is the case also for task 3, where the PSA stops its forward motion by first turning 180°.

At the same time, it is evident that although the desired basic navigational tasks are performed, their execution is not entirely perfect. After braking down to a near halt in each task, the PSA will inevitably maintain a small amount of linear or angular velocity, which the controller is not able to correct. Effectively, the controller collects fitness points until the PSA has slowly drifted out of the award area.

Given the unstability of this problem, with very little friction in the environment where the PSA is supposed to operate, this seems like the kind of behaviour one would expect also from the real robot.

In order to achieve a further improvement in fitness, and more robust controllers, a qualitatively different kind of behaviour is needed, in which corrective action is taken to bring the PSA back into the desired state whenever it has drifted too far.

The modifications proposed later in this paper did not improve robustness significantly. It is an open question whether a controller with this level of complexity could be found using the current methodology. A first step may be to further extend the capabilities of ESP to search not only network weights, but also network topology.

5.2 Suggested ESP Modifications

As previously described, certain changes to ESP are suggested, in order to improve the overall consistency and efficiency of the algorithm.

First, a more systematic selection of neurons for participation in neural networks is desirable in order to achieve a more reliable fitness estimate of each neuron. The effect of sampling each neuron the same number of times, while still creating networks by random combination of neurons, is shown in Table 4. $P(T \leq t)$ refers to the two-tailed probability of accepting the null hypothesis using a two-sample t-test with unequal variances.

With the current setup, this modification does not have a very significant impact on the efficiency of the search. However, it generally does not seem to have any negative impact on the search, and so should be implemented for improved consistency of the algorithm. Further, we hypothesize that, since a systematic sampling of neurons will give a more predictable neuron fitness estimate, this approach may allow the use of fewer network evaluations per neuron in the system, thus reducing runtime costs.

Second, an alternative way of assigning network fitness to participating neurons was tried. Rather than awarding each neuron the *average* fitness of each network in which it has taken part, each neuron is assigned the *highest* fitness achieved by any of its networks.

The results in Table 5 indicate that this change may indeed have a positive effect on the search. Compared to the original settings, shown in Table 1, there is an increase of al-

Table 4: Fitness scores, in % of theoretical maximum, for tasks 1, 2 and 3, when systematically sampling neurons during construction of networks. The scores from Table 1 are given in parenthesis for reference.

Task	Average	Standard Deviation	P(T≤t)
1	41.6	10.3	0.8
2	48.4	19.0	0.4
3	17.8	21.7	0.2
All	**35.9 (32.8)**	**21.9 (20.8)**	**0.3**

Table 5: Fitness scores, in % of theoretical maximum, for tasks 1, 2 and 3, when rewarding neurons for the best network in which they have taken part. The increased average fitness is clearly significant, compared to the values in Table 1. The scores from Table 1 are given in parenthesis for reference.

Task	Average	Standard Deviation	P(T≤t)
1	56.7	14.8	$1.8E^{-4}$
2	60.9	16.5	$9.2E^{-4}$
3	21.0	17.5	$1.4E^{-2}$
All	**46.2 (32.8)**	**24.2 (20.8)**	**$9.5E^{-5}$**

most 50% in the average fitness of the best evolved networks. Further, the total number of burst mutations performed during all 90 runs with each setting, dropped from 2448 to 1868. This may indicate that the algorithm was more resistant to stagnation with the new fitness assignment method.

5.3 Topology Modifications

A major strength of an evolutionary system compared to other search algorithms, especially gradient descent methods, is its versatility, and the way it can easily be adapted to evolve both the weights and the topology of a neural network [10].

ESP exemplifies this by adding a topology-modifying heuristic to a system that mainly evolves weights. However, as pointed out in an earlier section, the original implementation of the topology modification heuristic could lead to a very inefficient search.

As shown in Table 6, we saw no significant negative effect of disabling the topology modifying heuristic altogether, compared to our default implementation of the original heuristic. However, the alternative approach of erasing the best network at each burst mutation shows a significant improvement over both the two other scenarios, as shown by a 30% increase in average fitness in the experiments conducted here. Still, it seems that in order to fully harness the potential of evolution to develop both network weights and topology, further improvements of the heuristic are desirable.

Table 6: Fitness scores, in % of theoretical maximum, for tasks 1, 2 and 3, for different variations of the topology modification heuristic. Lesion: lesion tests are disabled. Erase: best network erased at each burst mutation. The scores from Table 1 are given in parenthesis for reference.

Task	Op.	Average	Std Deviation	P(T≤t)
1	Lesion	40.0	14.1	0.5
2	Lesion	44.8	18.9	0.9
3	Lesion	27.5	24.7	$0.2E^{-3}$
All	**Lesion**	**37.4 (32.8)**	**20.8 (20.8)**	**0.1**
1	Erase	51.3	13.8	$1.3E^{-2}$
2	Erase	53.9	20.8	$6.9E^{-2}$
3	Erase	17.6	22.7	0.2
All	**Erase**	**40.9 (32.8)**	**25.5 (20.8)**	**$2E^{-2}$**

Table 7: Average fitness scores of the best controllers found by hill-climbing, in % of theoretical maximum, for tasks 1, 2 and 3, and averages across all three tasks. The scores from Table 1 are given in parenthesis for reference.

Task	Average	Standard Deviation	P(T≤t)
1	38.2	18.4	0.3
2	52.2	22.2	0.2
3	15.5	16.4	0.3
All	**35.3 (32.8)**	**24.3 (20.8)**	**0.5**

5.4 Hill-climbing Comparison

As a way of estimating the overall efficiency of the ESP search, an alternative, hill-climbing search algorithm was also tried. Surprisingly, the performance of the hill-climbing algorithm compares favourably with that of the original ESP on the problem at hand, as can be seen from the figures in Table 7.

The hill-climbing algorithm has some similarities with a standard genetic algorithm for evolving network weights, with the lack of a crossover operator and an extremely high selection pressure, in which only the best network in each generation survives.

Part of the success of the hill-climbing algorithm seems to stem from the equivalent of the mutation operator, which with a certain probability changes the sign of a weight in the network. With a “mutation operator” that simply adds noise, the search is less efficient.

6. DISCUSSION

In this experiment, the basic ESP algorithm turned out to be only slightly more efficient than a very simple hill-climbing algorithm. This is consistent with the general belief that the main strength of evolutionary systems is not in the evolution of weights for neural networks [10], but still conflicts with several recent benchmarks of this particular algorithm.

It is our belief that this somewhat disappointing performance is, at least in part, caused by inaccurate fitness estimation for the evolved neurons. It seems that the overall fitness of a network may not be sufficient to judge the efficiency of each of its constituent neurons.

In future work, we intend to explore this by attempting to extract additional data from the running of each network, thus augmenting the algorithm to use additional information about the role of each neuron in a network. A promising approach is to apply information theoretical measures in addition to traditional fitness estimates. This line of research will be pursued in future work, as such quantitative measures of the fitness of each neuron in a network could be of use in a broad range of applications.

Partly as a result of the complexity of the ESP algorithm, there is a wealth of different parameters that must be specified by the experimenter, and their effect on the search efficiency can be very difficult to determine à priori. The small efficiency increase in this experiment therefore comes at the cost of a very intricate system.

On the other hand, a better understanding of how ESP traverses its search space may provide guidelines for choosing optimal parameter settings, which in turn may improve the efficiency of the algorithm.

Finally, from visual inspection of the performance of the best evolved controllers, and from the very low fitness achieved in the presence of a small amount of sensory noise in the simulations, it is clear that the evolved networks are far from representing usable controllers of the PSA. This can imply that a more complex network topology than the simple feed forward architecture used here is necessary, if neural controllers are to be used. As the complexity of the search space increases, evolutionary systems have performed increasingly better than other methodologies[3], so the difference between ESP and other systems may be larger in this case.

We have proposed certain changes to the ESP algorithm, in order to make it more consistent and streamlined. In experiments performed on the problem of controlling the PSA, the proposed changes have a significant positive effect on the efficiency and stability of the algorithm.

7. ACKNOWLEDGEMENTS

I would like to thank Haaken A. Moe for valuable discussions and good ideas, and Keith Downing and the rest of the CAOS group for their inputs and support.

8. REFERENCES

[1] J. Gomez, Faustino and R. Miikkulainen. Robust non-linear control through neuroevolution. Technical Report AI02-292, The University of Texas at Austin, Department of Computer Sciences, Oct. 1 2002.

[2] F. Gruau. Automatic definition of modular neural networks. *Adaptive Behaviour*, 3(2):151–183, 1995.

[3] D. J. Montana. Neural network weight selection using genetic algorithms. In S. Goonatilake and S. Khebbal, editors, *Intelligent Hybrid Systems*. John Wiley and Sons, 1995.

[4] D. E. Moriarty and R. Miikkulainen. Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, 5(4):373–399, 1998.

[5] Open dynamics engine (ode), 2007. `http://www.ode.org` [Online; accessed 16-January-2007].

[6] N. J. Radcliffe. Genetic neural networks on mimd computers. Ph.d. diss, Dept. of Theoretical Physics University of Edinburgh, Edinburgh, Scotland, 1990.

[7] N. J. Radcliffe. Genetic set recombination. In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 203–219. Morgan Kaufmann, San Mateo, CA, 1993.

[8] Y. F. Sit and R. Miikkulainen. Learning basic navigation for personal satellite assistant using neuroevolution. Technical report, Washington DC, USA, 25-29 June 2005.

[9] K. O. Stanley and R. Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 569–577, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

[10] D. L. Whitley. Genetic algorithms and neural networks. In G. Winter, J. Periaux, M. Galan, and P. Cuesta, editors, *Genetic Algorithms in Engineering and Computer Science*, pages 203–216. John Wiley and Sons, Ltd., Chichester, 1995.