Evolving Neural Networks for Forest Fire Control

Marco Wiering MARCO@CS.UU.NL
Intelligent Systems Group, Utrecht University, Padualaan 14, 3508TB, Utrecht

Filippo Mignogna FMIGNOGN@CS.UU.NL
Presidio Siemens, I.C.P., via Daverio 6. 20100 Milano

Bernard Maassen BMAASSEN@CS.UU.NL
Computer Science Department, Utrecht University, Padualaan 14, 3508TB, Utrecht

Abstract

Forest fire control is a challenging research problem involving a non-stationary environment and multiple cooperating agents. In this paper we describe the application of enforced sub-populations (ESP) to evolve neural network controllers that solve different instances of the forest fire control problem. Our system works by initially generating subgoals and assigning subgoals to the different agents. The subgoal generator and task assignment module are modelled as multi-layer perceptrons which are evolved to minimize the damage done by the spreading fire. The experiments show that agents learn to stop forest fires and that incremental learning can be used to solve more complex problems.

1. Introduction

Forests play a crucial role for sustaining the human environment and because forest fires are among the largest dangers for forest preservation, it is not a surprise to see increasing state expenditures for forest fire control. Despite of this, annually millions of hectares of forests are still destroyed by fires. If we look at the way current forest fires are attacked, then usually the fire boss makes an initial attack plan to stop the spread of the fire. This plan consists of a number of fire-lines that break the fire-propagation. Then he allocates resources from neighboring resource bases to fulfil all subplans. After this the field-commander is in control and reevaluates the plans constantly based on a stream of online information (Wiering & Dorigo, 1998). In case of very large fires, a first attack plan usually does not work. Therefore in this case, no-one knows how to deal with the problem and the forest fire control team usually waits until the weather changes which can last as long as three weeks. This of course results in large forest areas being destroyed. Therefore we want to study the application of intelligent algorithms for dealing with large forest fires.

Forest fires as expanding processes resemble disease epidemics and volcanic eruptions, and for all these catastrophes propagation lines should be removed. E.g., to control disease epidemics roads leading away from diseased areas should be severely controlled so that the disease cannot spread itself further. In forest fire control there are three ways to deal with the problem; removing fuel, removing oxygen, and decreasing temperature. Removing fuel can be done by bulldozers or other ground agents that cut away trees or grass and this is the most effective way for dealing with large forest fires. For removing oxygen or decreasing temperature, airborne agents can be used that throw waterbombs containing chemicals, but without cutting fire-lines this method is not sufficiently effective to stop a large forest fire (although airborne agents are helpful to decrease the propagation speed of the fire). Therefore, there are basically three types of attacks: airborne attack, a ground attack, and a mixed attack employing both air- and ground-forces. In this paper we concentrate on ground-attacks, although our simulator also allows for mixed attacks.

In previous work, forest fire control is done using planning algorithms. One of the first attempts was Phoenix (Cohen et al., 1989), a simulated environment modelling forest fires in Yellowstone National Park. Agents, which included watchtowers, fuel trucks, helicopters, bulldozers, and a coordinating fire boss, were used to fight the fire using planning techniques. The CHARADE project (Ricci et al., 1994; Avesani et al., 1997; Avesani et al., 2000) is a working environmental decision support system for managing first intervention in forest fires. The planning system integrates case-based reasoning and constraint satisfaction (Avesani et al. 1997) and is integrated with a geographic information system (GIS). The case-based reasoning system uses a database of previous plans to deal with forest fires in the south of France to select appropriate plans for the current fire. The problem of this approach is that there are no plans available for very large forest fires, which have never been controlled successfully. Instead, our approach relies on simulation and learning, where a large number of plans is simulated and after each simulation the plan-generating policy is adapted based on results of previous plans.

For learning to control forest fires, we use enforced sub-populations (Gomez & Miikkulainen, 1998) which is a promising evolutionary algorithm for evolving neural network controllers. Basically, enforced sub-populations (ESP) is an approach to solving reinforcement learning (RL) problems using direct policy search instead of learning a value function. The disadvantages and advantages of direct policy search versus value function based RL are still being debated, but we think that direct policy search based on evolutionary algorithms makes it faster to find initially good controllers, although the fine tuning of the controllers is more difficult (due to the small amount of policies that work better instead of worse compared to the best previous policy). Nevertheless, they might also be combined in some fruitful way, and we intended to research first direct policy search since the environment is composed of multiple agents and due to the highly non-stationary forest fire dynamics, learning accurate value functions is very difficult.

In the next section we will describe the forest fire simulator called “Bushfire”. In section 3, we explain how we use the ESP algorithm for evolving neural network controllers implementing forest fire control policies. In sections 4 and 5 we present experimental results. Finally in section 6 we discuss the obtained results and describe possibilities for future research.

2. Forest Fire Simulator: Bushfire

We study forest fire control by a learning multi-agent system. For this we developed a forest fire simulator, named Bushfire, based on a stochastic cellular automaton where single cells may contain different kinds of trees, grass, water, digged paths, and cells may be on fire or not. The fire starts at some place and then propagates itself according to wind strength and direction, and humidity.

The basic variable entity of a cell is its fire activity. If a cell is ignited the cell starts to release fire activity to its neighboring cells according to its type, wind direction and speed, and humidity. After a cell has received more fire activity than a specific threshold for this cell-type (e.g. the threshold for trees is larger than for grass) the cell starts to burn as well and releases fire activity to its neighbours. After some time the fuel of a cell becomes depleted and the cell enters a burned state that is not able to release fire activity any longer. By setting the different parameters, we can construct forest fires which are moving slowly or very fast. This enables us to deal with a large number of different difficulty levels for controlling them. One of the most important parameters here is the trade-off between the number of steps a bulldozer can make compared to the number of steps the fire is able to propagate itself. Setting this parameter to a high value means that it is easy to perform many steps to control the fire, thereby making the success ratio much larger and the size of the burned area much smaller.

The goal of the multi-agent system is to control the propagation of the forest fire. This they can do by cutting fire-lines around the fire. Therefore the question becomes: where should the agents cut fire-lines to minimize the damage done by the forest fire?

The problem can in principle be described by a Markov decision process consisting of states, actions for the agents, transition rules to go from one state to the next when the agents have executed their actions, and a reward function. The number of states is huge; first of all there are at least 10,000 discrete cells in our simulations which have their own properties such as consisting of grass, trees, digged paths, etc. Furthermore the properties of a cell also consist of the fire activity and the remaining unburned fuel which are continuous numbers. Thus, it is clear that even if we could measure all these properties, we cannot take the whole state information into account when selecting an action. The actions are much simpler, basically there are 8 of them; drive north, south, east, west, and dig north, south, east, and west. The transition rules are complicated and depend on the cellular automaton and its parameters. Finally, the reward function should handle the case that an agent is burned in the fire, that the fire is contained by the fire-lines, that special cells are put on fire (e.g. houses), and the size of the area being burned.

To solve the problem, we propose an algorithm that generates subgoals after which the bulldozers cut fire-lines between subgoals. The planning between subgoals is currently done using a simple linear path-planner, but can also be done using the A* algorithm that takes into account that cutting fire-lines over grass can be done much faster than cutting away trees. The main problem is to generate optimal subgoals. This is a difficult control problem, since each forest fire looks different and the state space is huge.

The method we devised for controlling forest fires consists of four steps:

- Generate initial subgoals around the fire
- Enhance subgoals using local information
- Assign each bulldozer to a specific path from one subgoal to another subgoal
- Dig lines between subgoals using a path-planner

Some replanning of subgoals is done automatically if a straight line from an agent to a subgoal goes through the fire. We explain these steps in more detail below.

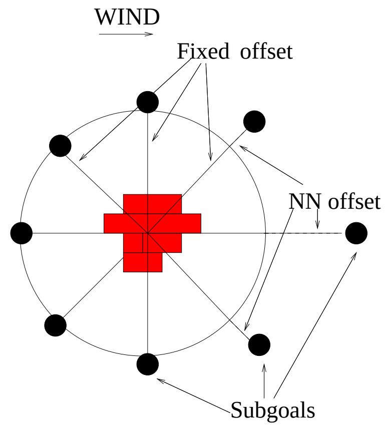

Figure 1. Around the fire we first construct 8 subgoals using a fixed offset. After this, the neural networks receive information from each subgoal and learn to displace it from the fire. The 8 subgoals will then be used for the bulldozers, although replanning may change them afterwards.

2.1. Generating Initial Subgoals

For controlling forest fires, we developed a representation using 8 subgoals in all wind-directions around the fire. If the agents are able to cut fire-lines around the fire by going through and connecting all subgoals, the fire has been contained. For generating the subgoals we use 8 lines in all 8 directions to place each subgoal. First we use a fixed offset from the centre of the fire (this can be seen as a non-learning approach) and then we learn with a neural network how much the offsets should be displaced in all 8 directions. This neural network can receive as inputs the distance and angle to the fire front, the wind and speed direction, and some measures of the average humidity, fuel, and the threshold of the fire front neighbouring cells. Figure 1 shows how initial subgoals are generated.

2.2. Subgoal Enhancer

The previously described subgoal generator only used global information such as wind speed and direction and distance to the fire front as inputs in the neural network. Sometimes it is necessary to look at local characteristics as well for generating subgoals. E.g. if there is a house or river nearby, this may change the optimal plan. The task of the subgoal enhancer module is to learn to map local information to a small change of the position of the subgoal. The enhancer neural network looks at all initially generated subgoals and receives local information given the place of the subgoal to compute an evaluation. Then the subgoal may be changed some squares in the cellular automaton and using the new input generated by the new relative local surroundings, the neural network computes a new evaluation. If some neighboring square has a higher evaluation than the current square, we continue this local hillclimbing strategy with the new square and otherwise we stop.

2.3. Task Assignment

Although we assume that the number of resources is fixed before a simulation (the number of resources can be chosen by the user), we still have to assign (allocate) the subgoals to each individual agent. This is also done using neural networks. The neural networks obtain information such as the distance from a subgoal to the agent, the wind speed and direction, the distance from an agent to the fire front, and which subgoals have already been chosen before, to choose for each agent a single subgoal to go to. If there are multiple agents, all agents examine all subgoals from which a list of evaluations is obtained by propagating subgoal and agent information through the neural networks, after which first the highest evaluation is taken and that agent is allocated to that subgoal. Then, the fact that this agent is allocated to a subgoal is used as input in the neural network for computing the subgoals of the next agents, after which the second agent is allocated to its highest subgoal etc. In this way the task assignment module is responsible for learning to coordinate the multi agent team. There is no need for communication etc., since we alternatively assign each agent to a subgoal and this information is used by the task assignment module to assign the other agents one by one.

2.4. Path Planning

Once subgoals are generated and allocated to each agent, the agent uses a path-planner to dig a fire-line from its current position to the subgoal. In case the simulation just started, the agents first go to a subgoal before digging a fire-line thereby driving with higher speed to reach a subgoal. We implemented a simple path-planner that constructs more or less a direct line to the subgoal. If it is detected that this straight line goes through the fire, the subgoal's location is changed by the subgoal generation module now taking into account different input information. The other path-planner is A* and can take into account that going close to the fire is dangerous, and that digging through grass goes faster than digging through trees, but since A* is much more time consuming, we only used the simple path-planner in our current experiments.

[t]

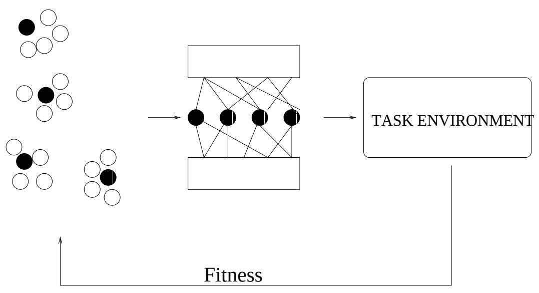

Figure 2. Enforced Sub-Populations (ESP) constructs a neural network by taking one neuron from each subpopulation. The resulting neural network is tested and the evaluation is used to evolve novel sub-populations of neurons.

3. Enforced Sub-Populations for Evolving Neural Networks

For learning to generate subgoals and assign agents to them, we use Enforced Sub-Populations (ESP). ESP (Gomez & Miikkulainen, 1998) is an evolutionary method for evolving neural networks. It works by keeping different subpopulations containing neurons that have weighted connections to inputs and outputs (see Figure 2). To generate a neural network, one neuron is selected from each subpopulation and these neurons then form a feedforward neural network. In order to train the system, each neuron from each subpopulation is combined a number of times (we use 10 times on average) with neurons from different subpopulations, and the neuron is assigned the average (or maximal) fitness of the networks in which it took part. Then crossover and mutation are used within the subpopulations to generate new neurons. In this way, neurons that can collaborate well with other neurons will receive higher fitness values, and will be used to evolve novel neurons. ESP has already been used for particular difficult reinforcement learning problems such as double pole balancing with hidden state (Gomez & Miikkulainen, 1998) and active guidance of a rocket (Gomez & Miikkulainen, 2003) and obtained good results.

The reason we used ESP for evolving neural networks is that they do not suffer from the permutation problem when using crossover on complete neural networks. This permutation problem is caused by different orderings of neurons in a network, so that offspring can easily have the same functional units multiple times and loose an important different unit. Since we want to use crossover for finding solutions, we use a symbiotic algorithm with specialized neuron subpopulations. A difference with SANE (Moriarty & Miikkulainen, 1996) which keeps a single population and no sub-populations is that SANE requires a metastable population in which neurons with different functions stay in the population at the same time, whereas in ESP each sub-population may converge to a unique neuron. Furthermore, recombination of neurons implementing different functions is usually not very effective, and therefore ESP only recombines neurons in the same sub-population. In some aspects, ESP resembles the cooperative co-evolution method from Potter and de Jong (2003) and SEAM from Watson and Pollack (2000) which are also evolutionary algorithms based on symbiotic combination instead of sexual reproduction, but ESP is specifically tailored for evolving neural networks.

We use ESP for evolving the different modules; the subgoal generator, the subgoal enhancer, and the task assignment (although in our current experiments we did not use the subgoal enhancer module). These different modules can be evolved at the same time. An advantage of ESP is that it is easy to use for multi-agent learning. In multi-agent learning, issues arise about credit assignment to individual agents given a team reward. In ESP these issues are solved using the same mechanism as with single agent learning; each agent uses its own neurons and each neuron is again evaluated by how well the resulting combinations of neurons (and multiple neural networks) work. The fitness of such a combination of neural networks is computed by testing the system in a forest fire simulation. The fitness function takes into account the burned area, whether the fire-propagation was stopped, how much subgoals were successfully digged, and how often it was necessary to recompute subgoals. By evolving neurons which have higher and higher fitness values, the resulting neural networks also become better and after a while we can save the best found complete neural network modules.

4. Experiments with Learning the Subgoal Generation Module

In this section we perform two different experiments to study the ability of the system to learn to control forest fires using only the subgoal generation module. The enhancer was not necessary, since the local regions look alike (there are no houses, rivers etc.). Furthermore, we use the fixed nearby assign module for allocating agents to subgoals. Therefore in the experiments we describe in this section we only used a learning subgoal generation neural network.

In the **first experiment**, we study a virtual pine forest environment in which three agents are located in different parts of the world that have to cooperate to stop the fire propagation. In this environment we gradually lower the cells' thresholds to increase the fire propagation speed. In our **second experiment**, we show the *Incremental Learning* skill of ESP. If we train a population to solve a determined task, we can use the same population to solve a similar, but more difficult task. In previous work (Gomez & Miikkulainen, 1998), it was shown that starting the evolution from a trained population in an easier task resulted much faster in good performing individuals for a more complex task than starting evolution from a randomly initialised population.

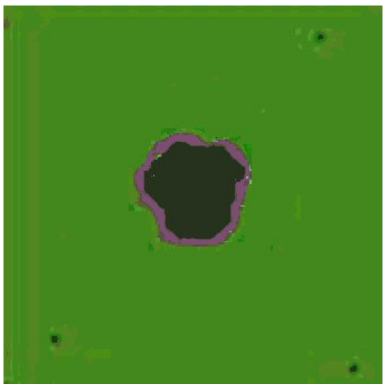

Figure 3. Pine forest environment: in the centre the fire has spread as far as the fire-line has been digged. In the angles north-east, south-west, and south-east of the map we can see the bulldozers' depots.

4.1. Three Bulldozers in a Pine Forest

The first set of experiments are done on a virtual pine forest by an agency composed of three bulldozers. Trees are more difficult to ignite than grass, however once ignited they release much more heat thus propagation is still fast. For this reason pine-tree cells present bigger fuel and threshold than grass. Controllers operate in much more difficult circumstances in a pine trees forest than in a grassy terrain. We model this by decreasing the digging and moving speed in a pine forest environment. We experimented in a world of 120×120 cells. Experiments with a single agent were not satisfactory. One single bulldozer was not able to dig a complete line around the fire in a good time. Therefore we used a system composed of three bulldozers initially located in three different depots. In Figure 3 we show a virtual pine forest in which fire has been contained in the line digged by the agents.

Setup. We tried different kinds of neural networks with different architectures and activation functions. We decided to use a single hidden layer network with 8 hidden units and 1 linear output unit, because we did not find meaningful improvements using more layers and neurons. The sigmoid function has been proved to work well in the hidden layer but not in the output unit. The setting of the ESP parameters applied to our architecture are as follows: Population size is 30. We assign the maximal (and not the average) fitness obtained in one of the (on average ten) tests of a neuron (individual). The crossover rate is 50%. We used linear mutation with 25% probability and Gaussian mutation with 33% probability. We designed a quite complex fitness function including information such as the time of spreading, the overall cost of the cells burnt, the number of generated subgoals and the number of goals reached.

Threshold	Burnspeed	Gen.	Fitness	Efficiency
330	1.11	1	11.438	52.4%
305	2.35	2	11.375	53.5 %
280	2,66	2-3	11.279	55.9 %
255	2,75	3-4	11.044	58.8 %
230	3,16	5-6	10.201	64.8 %
205	3,33	-	3.787	0

Table 1. ESP performances using a team of three agents in a virtual pine forest environment.

Results. In Table 1 we can see the results. The task is solved if the system finds a solution with fitness bigger than 10 (in this case the fire propagation has been stopped effectively). We always stopped the simulation after the twentieth generation. The simulations were repeated 10 times for each problem and we store the average of the maximum fitness received in each simulation in the table. We also store the generation in which the system finds the solution in the majority of the simulations. Efficiency is the ratio between the time of the controllers to dig a complete line surrounding the fire and the time of the fire to burn the cell inside that line. When the system finds a solution with a low efficiency ($\leq 35\%$) it means that the task was too easy or that the solution found was not so useful, because the agents could have digged a smaller line to reduce the area destroyed by the fire. However, if this value is too big ($\geq 90\%$) the solution computed is not robust, because if there is a slight change of the environmental conditions the system cannot anymore deal with the problem.

For all the tasks except for the last one we found a solution in each simulation; in the last task we tried with a threshold of 205 we never found a solution. The fastest fire propagation the system could successfully deal with is characterised by a threshold of 230 and an average propagation speed of about 3.16 cells/step.

We can also see from Table 1 that the fitness obtained decreases when lowering the threshold. However, the efficiency of the system increases. The explanation lies in the fact that for faster propagation the system has to generate a larger digged line to surround the fire. Thus, more cells burn and the fitness is lower. However, in the same situation the system has proved to be more efficient in terms of time to dig the line versus the time for the cells to burn. Figure 4 shows a typical learning evolution of ESP with a threshold of 230.

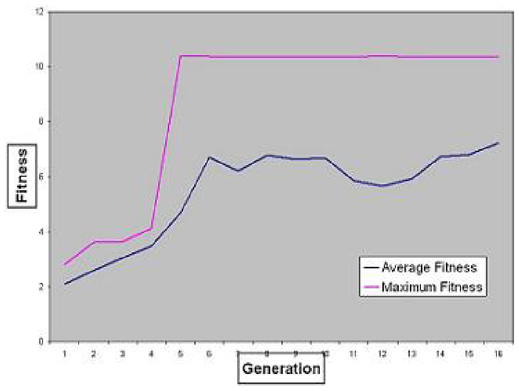

Figure 4. ESP applied to contain flames in a pine forest environment with a threshold of 230.

ESP works well even if at the first generations the maximum and the average fitness are very low. In these cases the fire propagation was not stopped. After the fifth generation we can see that the maximum fitness increases a lot. In that case a solution has been found

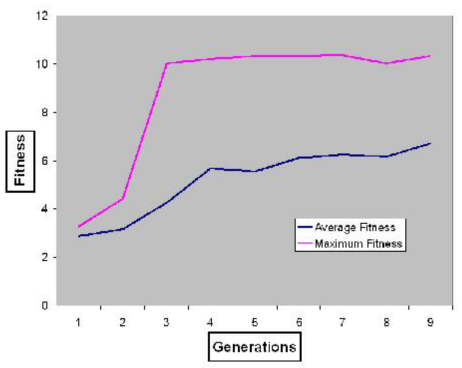

Figure 5. Incremental ESP applied to contain flames in a pine forest environment with a threshold of 230.

with an efficiency of 65%. After that individuals start to converge to the best performing ones. We can see ESP's principal strength: the fast search for a satisfactory solution. ESP is not so good in improving the solution found: when a satisfactory result is obtained ESP rarely improves it. In this experiment we found that ranking the neurons according to the maximal fitness received in the tests worked better than taking the average fitness.

4.2. Incremental Learning

We repeat the previous experiments on the virtual pine forest with the same neural network and ESP settings. However, we do not start from a randomly generated population, but we use a population already trained in an easier task. In our experiments we start to solve the task with a threshold of 280. Once we have found a solution we use the population trained to search a solution for the task with a threshold of 255. Then we repeat the same operation until we reach the task with a threshold of 205. In Figure 5 we can see the fitness evolution in solving the task with a threshold of 230, starting with a population already trained to solve the easier tasks with a threshold of 280 and 255. With respect to the fitness evolution of the same task starting with a random population we can observe that the system finds a solution in the 3rd generation instead of after 5 or 6 generations. Then, we used this trained population to solve the task with a threshold of 205. We repeated this last part of the experiment 10 times and we found a solution in 80% of the trials. The generations in which we found the solutions range from the eighth generation to the twelfth generation. We note that without incremental learning, we never found a solution for this difficult problem. Thus, starting from a population already evolved to solve an

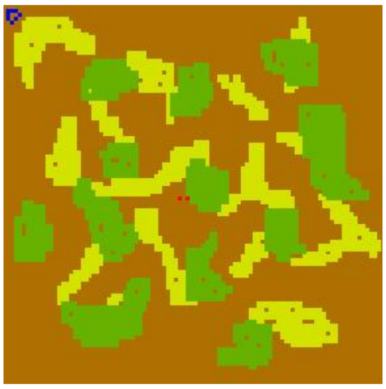

Figure 6. Heterogeneous environment consisting of hay, small brushes, and oak trees. In the north-west we can see the bulldozer's depot.

easier task has been proved effective in this case.

5. Experiments with Learning Subgoals and Task Assignment

In the previous experiments, we only trained the subgoal generator module (SGG), since the nearby assign module works well for homogeneous terrain types. However, for heterogeneous terrain types, learning subgoals and the assignment may prove worthwhile.

The map used for these experiments consists of 3 terrain types, one is fast burning but does not generate much heat (hay), one is slow burning but generates lots of heat (oak trees) and we use a terrain type that is between the two (small brushes). The terrains are setup in such a way that the hay will ignite the small brushes in time, but has big problems igniting the oak trees. The small brushes can ignite the hay with ease and will ignite the oak trees in time. The oak trees can ignite both with ease, and they all can ignite themselves. With these terrain types we have built a map that has corridors of fast burning hay and slow burning plains. In this way the fire does not spread in a circle form but has a far more grim form, making it much more difficult to extinguish. In Figure 6 we show the map.

All experiments are done on the same map under the same conditions. We used only 1 agent to extinguish the fire. The map had a dimension of 100 by 100 cells. Each simulation went on until the fire was extinguished or the program had done 2000 steps. These 2000 steps are more than enough to burn almost the whole map giving a very low evaluation value. The agent started in the upper left corner and the fire started in the middle of the map.

We used the map and let the fire spread with different speeds. Instead of adjusting the threshold values, we used the Fire Propagation (FP) step divider that regulates how many steps FP has to wait before it can do one step. By making this each time 1 less we can find a value for which the system cannot find a solution.

5.1. Experiment 1: Without Learning

As a baseline for the results, we first examined the performance when learning was not used at all in this map. In this experiment the subgoals are placed at a specified distance from the fire front, and the agent will drive/dig towards the closest free subgoal. The results of this experiment are shown in Table 2.

FP step divider	Number of Successes
10	10/10
9	9/10
8	3/10
7	0/10

Table 2. Results without learning. A lower FP step divider increases the propagation of the fire relative to the number of steps a bulldozer can make, thereby making the problem more difficult.

5.2. Experiment 2: Learning only SGG

In this experiment we only learn to generate subgoals and use the fixed nearby assignment module. The neural network for generating subgoals only used two inputs; the wind vector, and the distance from the fire centre to the fire-front, where wind vector means the difference between the wind direction and the line angle used to find the direction. Since we used static weather for the experiments, it was not necessary to include the wind speed. The results of this experiment are shown in Table 3.

FP step divider	Nr. Generations
10	5
9	6
8	8
7	20
6	-

Table 3. Results with learning only subgoals.

5.3. Experiment 3: Learning Assign

In this experiment we used a non learning SGG module that placed the subgoals at a fixed distance from the fire front. But now we used a learning assign module. The neural network for learning the assignment task has the following inputs: the distance from the fire to the subgoal, the distance from the fire to the agent, the distance from agent to the subgoal, the distance from the subgoal to the middle of the fire, the distance from the agent to the north, east, south and west of the map, the wind direction and force, and for each subgoal if it is free or already assigned.

The results of learning assign are given in Table 4.

FP step divider	Nr. of Generations
10	8
9	15
8	19
7	-

Table 4. Results with learning only the assign module.

5.4. Experiment 4: Learning SGG and Assign

In this experiment we used the learning SGG module and the learning assign module. We compare two approaches: learning both modules at the same time, and learning the modules alternatively. We started with training both modules at the same time, but this did not work at all. Even when the FP step divider was set upon 10, the program was not able to learn a solution within 50 generations, hence we aborted this.

The second experiment went a lot better. Here we first trained the SGG module while using the nearby assign module. Then when this reached an acceptable level we stopped training the SGG module and evolved the Assign module. Then when Assign showed intelligent behavior we stopped training the Assign module and started training the SGG module again. This process was repeated until no significant improvements were shown. In this way, we obtained successful controllers even when the FP step divider was set to 5. A very good solution to this difficult problem was obtained after 40 generations, but after 25 generations we already had some good results.

6. Discussion

In the experiments we have seen that cooperative agent policies are evolved to solve quite large forest fires. By using incremental learning the system was able to find solutions to even harder tasks with a larger fire propagation speed. This means that incremental learning can be effective to solve even more complicated problems. We have also studied the evolution of different modules (the subgoal generator and the task assignment). It turned out that evolving them synchronously did not work well. The reason of this is that if both modules are constantly changing and initially random it takes a lot of time to progress to a solution. In other experiments, when we evolve only one module at a time, evolution is much more steady and although in the initial generations no solutions were found, the system finally learned good solutions. One way to cope with multiple adaptive modules is to train them alternatively as was done in the experiments in the last section. By optimizing both modules, but not at the same time, solutions could be found even for environments in which the fire propagation was very fast. In the future, we want to research even larger and more complex forest fires. The Bushfire simulator allows us to generate a learning environment in an easy way, and therefore it is possible to perform much more experiments. For this we want to focus on incremental learning and training all modules in an asynchronous way and test the enhancer module by generating environments containing houses or rivers.

References

Avesani, P., Perini, A., & Ricci, F. (1997). CBET: A case base exploration tool. *Proceedings of the 5th congress on the Italian Asscociation for Artificial Intelligence on Advances in Artificial Intelligence* (pp. 405–416). Spinger-Verlag, London.

Avesani, P., Perini, A., & Ricci, F. (2000). Interactive case-based planning for forest fire management. *Applied Intelligence*, *13(1)*, 41–57.

Cohen, P., Greenberg, M., Hart, D., & Howe, A. (1989). Trial by Fire: Understanding the design requirements for agents in complex environments. *AI Magazine*, *10(3)*, 32–48.

Gomez, F., & Miikkulainen, R. (1998). 2-d pole balancing with recurrent evolutionary networks. *International Conference on Artificial Neural Networks* (pp. 425–430). Berlin, New York: Springer.

Gomez, F., & Miikkulainen, R. (2003). Active guidance for a finless rocket through neuroevolution. *Genetic and Evolutionary Computation Conference (Gecco-03)* (pp. 2085–2095).

Moriarty, D. E., & Miikkulainen, R. (1996). Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, *22*, 11–32.

Potter, M., & Jong, K. D. (2003). Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, *8(1)*, 1–29.

Ricci, F., Mam, S., Marti, P., Normand, P., & Olmo, P. (1994). *CHARADE: a platform for emergencies management systems* (Technical Report 94094-07). IRST, Trento, Italy.

Watson, R., & Pollack, J. (2000). Symbiotic combination as an alternative to sexual recombination in genetic algorithms. *Proceedings of Parallel Problem Solving from Nature (PPSNVI)* (pp. 425–434).

Wiering, M. A., & Dorigo, M. (1998). Learning to control forest fires. *Proceedings of the 12th international Symposium on "Computer Science for Environmental Protection"* (pp. 378–388). Marburg: Metropolis Verlag.