Combining Manual Training and Enforced Sub-Populations to Control Forest Fires

M.J.R. Hofland
Faculty of Computer Science
Utrecht University
The Netherlands
2007

MSc Thesis
INF/SCR-04-81

Supervisors

Dr. M.A. Wiering
Prof. Dr. J.-J.Ch. Meyer

Abstract

This thesis presents a further development of Bushfire, an experimental application that uses the neuroevolutionary algorithm Enforced Sub-Populations (ESP) to train Neural Network Controllers for agents in a forest fire simulation. The author of this thesis has extended Bushfire with new functionality, allowing a user to provide a solution directly. This solution can successively be used as a training set to train or pre-process the initial subpopulations used by ESP. This is achieved with an implementation of the supervised learning technique Back-Propagation. This thesis will show how combining manual training and enforced sub-populations will compare to the regular use of enforced subpopulations, when used to train (controllers for) agents in Bushfire.

Acknowledgements

I would like to thank *Dr. Marco Wiering* for his feedback and supervision during a very challenging assignment. A special thanks to my family and friends for their unconditional support throughout the years.

"Everybody who is incapable of learning
has taken to teaching"

Oscar Wilde

Contents

List of Figures

List of Tables

1 Introduction

This chapter functions as an introduction to this thesis. The first section provides a motivation for the research done, followed by the problem statement. The final section will give an overview of the remaining chapters.

1.1 Forest Fires

Forest fires are hazardous natural phenomena that can cause tremendous damage to the natural habitat. It occurs predominantly after long hot and dry periods, when plants are highly flammable. In dry and windy conditions, a forest fire can become uncontrollable fairly quickly and deadly for anyone caught unaware. Authorities will therefore try to respond as soon as possible and deal with a forest fire in its early stages. When a forest fire grows beyond a certain size, authorities will have to change their strategy to damage control. In a worst case scenario where urban areas are in the path of an approaching forest fire, building proper firebreaks becomes very important. The many factors that can influence the unpredictable propagation of a large forest fire make for a challenging control task.

1.2 Problem Statement

Previous research in [WMM05, WD98, WM03] describes the use of neuroevolution and more specifically Enforced Sub-Populations (ESP) as a good approach to training neural network controllers in a forest fire simulator (Bushfire). Agents controlled by the evolved neural networks are able to solve a variety of forest fire control problems. However, the number of generations needed by ESP to find a solution grows substantially with the increasing of the difficulty of the problem. And evolving the subpopulations from one generation to the next takes a nontrivial amount of time.

This thesis introduces the capability of performing a restricted form of supervised learning within Bushfire. This learning method is called 'manual training' throughout the thesis. It is proposed as a means to reduce the number of generations needed to find a good solution to a forest fire control problem. To this effect, running the ESP algorithm will be preceded by the manual training of its subpopulations.

This thesis will show that combining manual training and enforced subpopulations will decrease the amount of generations needed to generate neural networks that produce good solutions to forest fire control problems. In addition it will show that combined training shows better results than ESP without manual training, when solving forest fire control problems of higher complexity.

1.3 Thesis Outline

The next chapter explains how to use Bushfire, the application used and partly developed for this thesis. The third chapter will provide the reader with a basic understanding of the techniques used for this thesis. The concepts of neural networks, supervised learning and neuroevolution will be described. Chapter four describes and discusses the concept of

combining manual training with ESP. The fifth chapter describes the experiments that have been done and the results they have produced. The fifth chapter analyses these results and the final chapter presents the conclusion of this thesis and makes proposals for future work.

2 Bushfire

Bushfire is an application, designed to simulate forest fires and experiment with the neuroevolutionary algorithm Enforced Sub-Populations (ESP). It evolves neural network controllers used to control multiple cooperating agents with the objective of containing the fire. The neural network controllers are feed-forward multi-layer perceptrons which generate subgoals around the area where the fire occurs. The collection of all subgoals equals the solution to the forest fire problem. Agents that act as bulldozers use these subgoals to contain a spreading fire by digging trenches between these points through which the fire can not propagate. By evolving these controllers to find better subgoals the damage done to the virtual environment is being minimised.

New functionality has been added to Bushfire allowing a user to specify a solution and use supervised learning to train neural networks. Mixing the initial subpopulations with the neurons from these networks provides a better starting point for ESP when it is used to further evolve the subpopulations.

2.1 Installation

As Bushfire is a java program it will need a Java Runtime Environment to be able to execute. This distribution has been tested with version 1.6 under the Windows OS. Executables can be found in the 'bin' folder. Included with the distribution is a project file named 'default.bfp' and a map called 'pixmaps' which should both be placed in the Bushfire root folder for the program to work properly. Also included are 5 project files that can be used to load the settings used for the 5 experiments done for this thesis. When running for the first time, make sure that the following windows have been opened: World Builder, Trainer, Simulation and Output. All four windows can be opened from the Window menu. Storing (and loading) the project settings can be done from the File menu. Projects will be saved in files with a 'bfp' extension.

2.2 World Builder

Starting a new project begins with changing the settings in the 'World Builder' tab. Selecting the nodes in the explorer view to the left of the screen opens the corresponding settings in the main view. The catalogue is a compilation of all the different items that can be added to the world (or map). It is not required to change anything in the catalogue. Selecting the Catalogue folder in the explorer view allows for the creation or deletion of catalogue items in the main view. Figure 1 shows the settings available for each catalogue item, in this case grass. Changing the settings will affect all grass cells on the map. The first editable setting is the colour, which is the colour of grass in the graphical view of the world. The 'threshold' of a cell needs to be exceeded by a neighbouring burning cell's 'fire intensity' before it will start burning. A burning cell will increase its fire intensity and decrease its amount of 'fuel' each step of the simulator until it has no more fuel, after which it stops burning. The cost of the cell is used in the fitness evaluation. It defines the value of a cell relative to the value of cells of different types. If the catalogue item allows for driving through or digging of by agents, then the amount of time (simulation steps) needed per cell can be set.

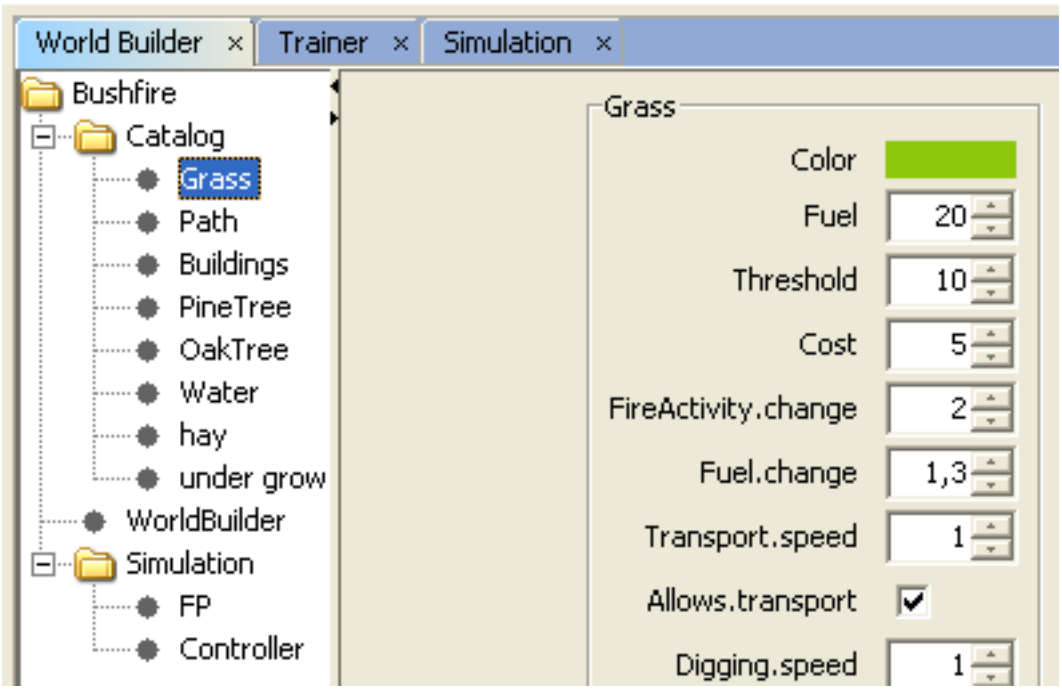


Figure 1: Each catalogue item has several settings that define its interaction with the fire and agents.

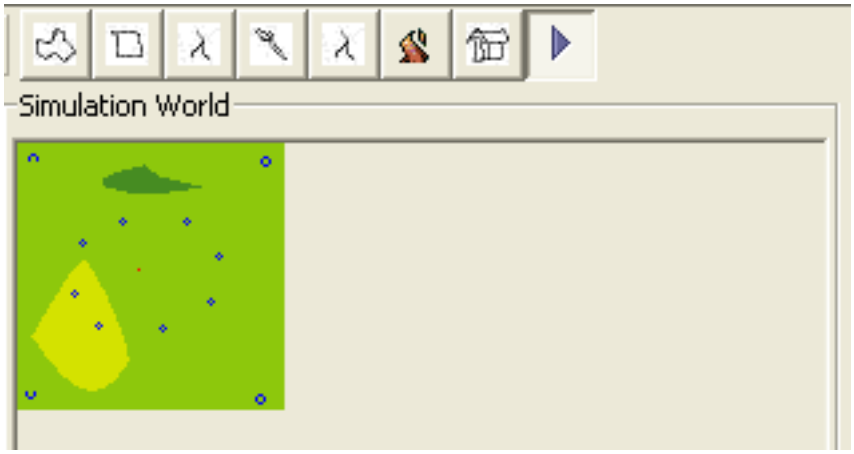


Figure 2: In the 'World Builder', the user can add fires, depots, subgoals and different types of terrain in customizable shapes.

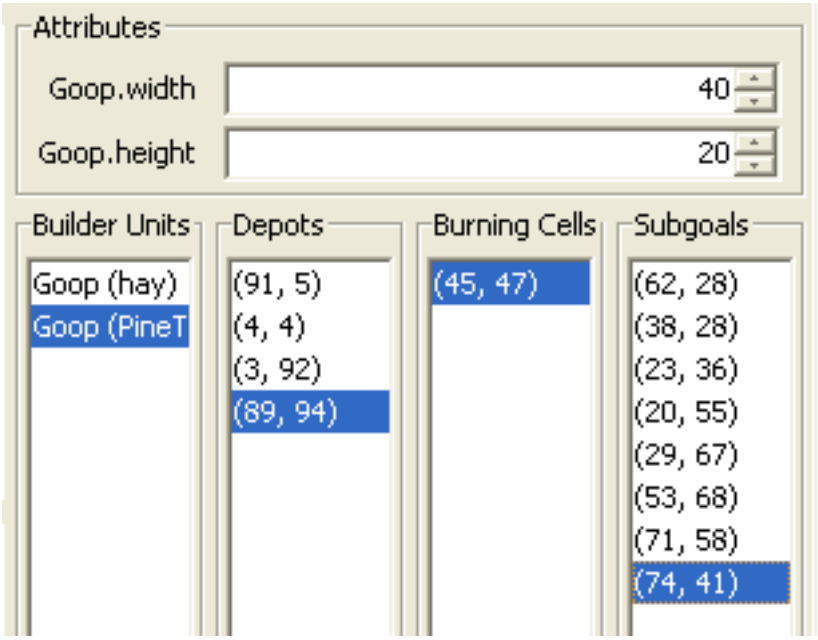


Figure 3: Everything added to the world can be removed by removing the corresponding link from the lists shown.

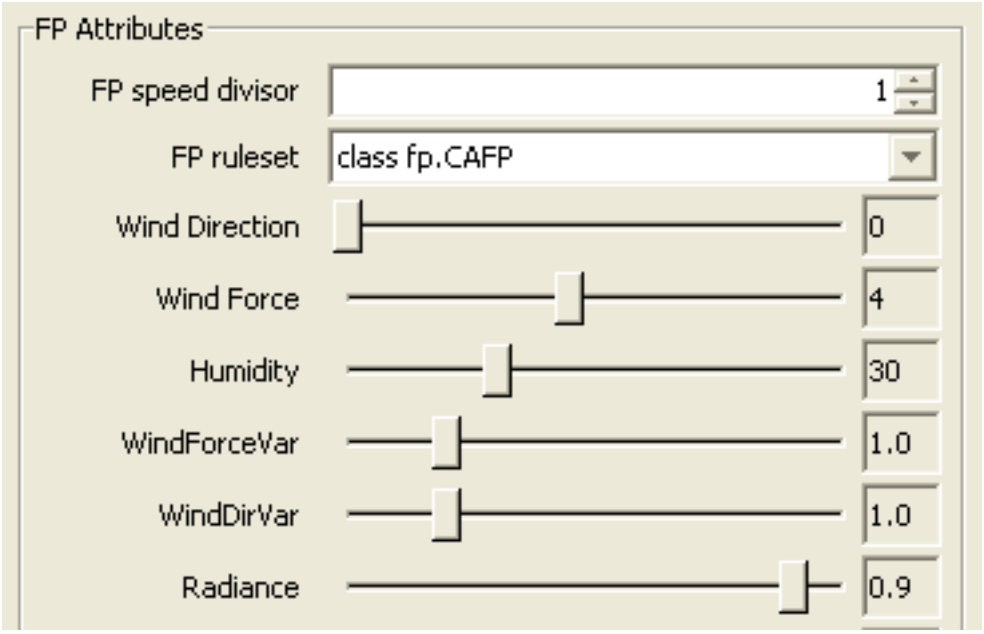


Figure 4: Increasing the propagation of fire will increase the difficulty of finding a good solution.

When all the catalogue items have the desired settings, select the 'WorldBuilder' node in the explorer to open the map editor. In the editor shown in figure 2, setting the background sets a default catalogue type for all cells. Setting 'Current Catalogue Item' and selecting a drawing tool from the toolbar, allows the user to draw that item on the map with the mouse. Changing the 'Random Seed' affects the shapes created by some of the drawing tools. The final three buttons on the toolbar can be used to draw a fire, a depot or a subgoal on the map. A depot can be used as a starting point for an agent and subgoals are the points that agents dig between. All additions to the map are tracked in lists (shown in figure 3), visible to the right of the map. These lists allow for an easy removal of anything unwanted. It is required to add at least one source of fire and one depot on the map.

2.3 Simulation Settings

The FP node represents the Fire Propagation settings (shown in figure 4), where the behaviour of fire in the simulation can be configured. The first attribute is the 'FP speed divisor'. It divides the speed of fire propagation with the entered value. More specifically, it defines how many simulation steps the agents can make for each step of fire propagation. Setting this value to 1 ensures the fastest possible fire propagation. This is the only setting that has been modified in the experiments. The second attribute sets the module that defines the behaviour of fire in the simulator. Experiments were done with the CAFP module, which takes into account a lot of environmental factors such as wind and humidity. The next attributes are used by the selected module to specify the simulator weather conditions.

After a map has been created with at least 1 depot, the user can select the 'Controller' node to start creating agents. Enter the name of the first agent in the text field shown in figure 5 and press the 'add' button. The freshly created agent will now appear in the list shown in figure 6. Repeat the above procedure to add more agents. Selecting one of the agents from the list shown in figure 6 allows for the changing of that agents settings (shown in figure 5). These settings allow the assignment of different modules that control the agent and also the assignment of a depot as a starting location.

The first module is the 'Subgoal generator' module and is set globally for all agents. It is responsible for generating 8 subgoals around the burning area. The 'SGG2' module is the most advanced subgoal generator module. It builds a neural network controller from a population of genepools (or subpopulations) and works fully in conjunction with the ESP and backpropagation algorithm implementation. Or in other words, it supports neuroevolution and manual training. When selecting the SGG2 module, a popup window will appear that allows the creation of new genepools (subpopulations) or the loading of existing ones. The popup shown in figure 7 lists the number of generations the subpopulations have been evolving and the highest average fitness score obtained in a generation. Note that storing subpopulations for each controller is done automatically during training whenever a generation shows improvement.

The enhance module is also set for all agents simultaneously and is responsible for optimising the positioning of the subgoals. Currently it is not working as intended and should be set to the 'EnhanceEmpty' module to disable it.

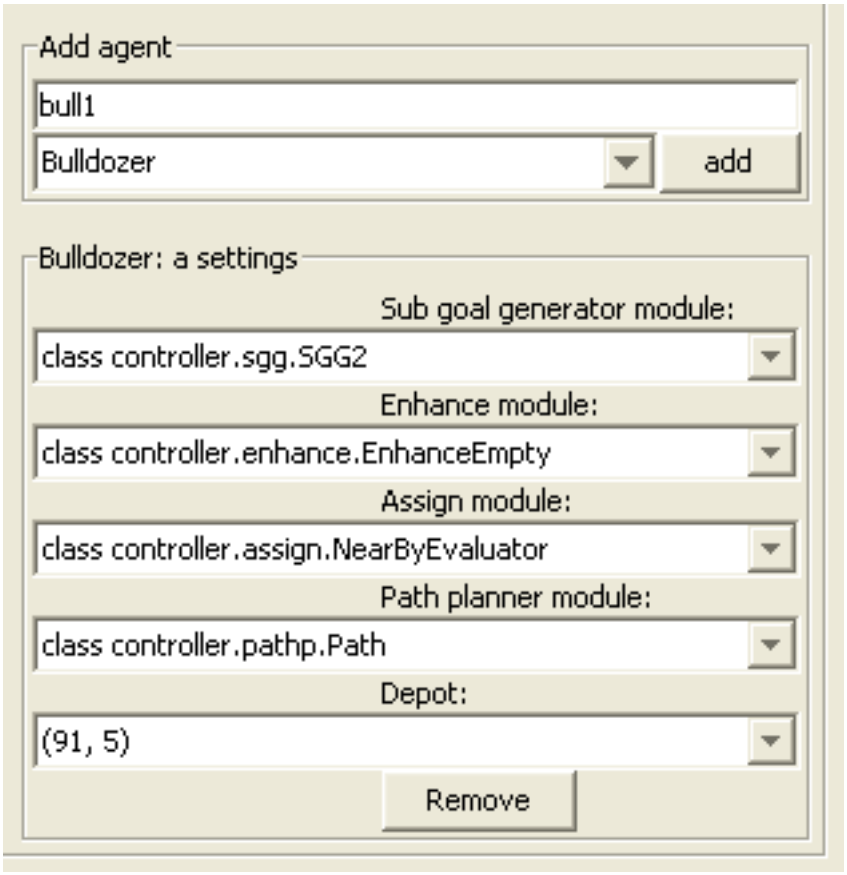


Figure 5: Create an agent by assigning controller modules and a depot. Assigning a Neural Network Controller module shows the popup shown in figure 7.

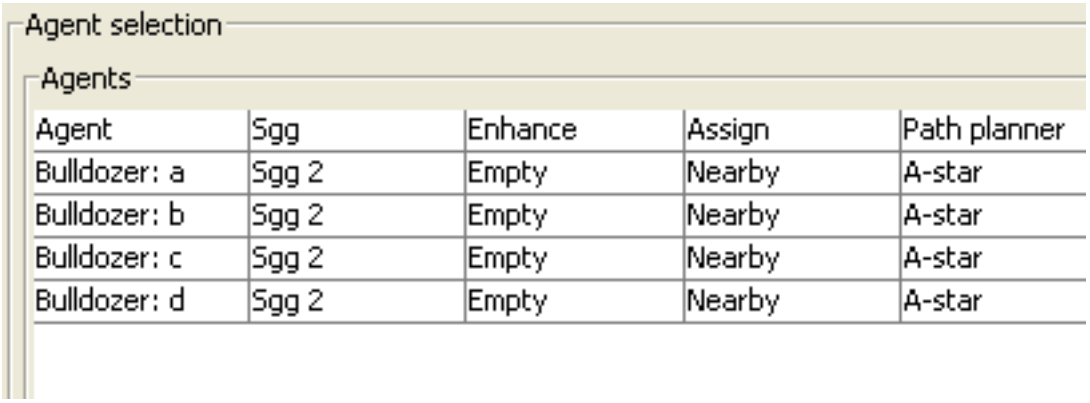
The assign module assigns the subgoals to the available agents. The 'Nearby' assign module, assigns subgoals to whatever agent is closest at the time. This module was used throughout the experiments done for this thesis. A more complex assign module is available that supports the training of and assigning by a neural network.

The path planner module calculates the path an agent takes from one point to another (usually a subgoal). The module named 'Path' implements the A* algorithm as a path planner. [Wik07b] describes the mechanics of this algorithm. It is the best and most advanced path planner available within Bushfire.

When the agents are configured, the user can optionally go back to the editor to add eight subgoals to the map to make a manual solution.

2.4 Trainer

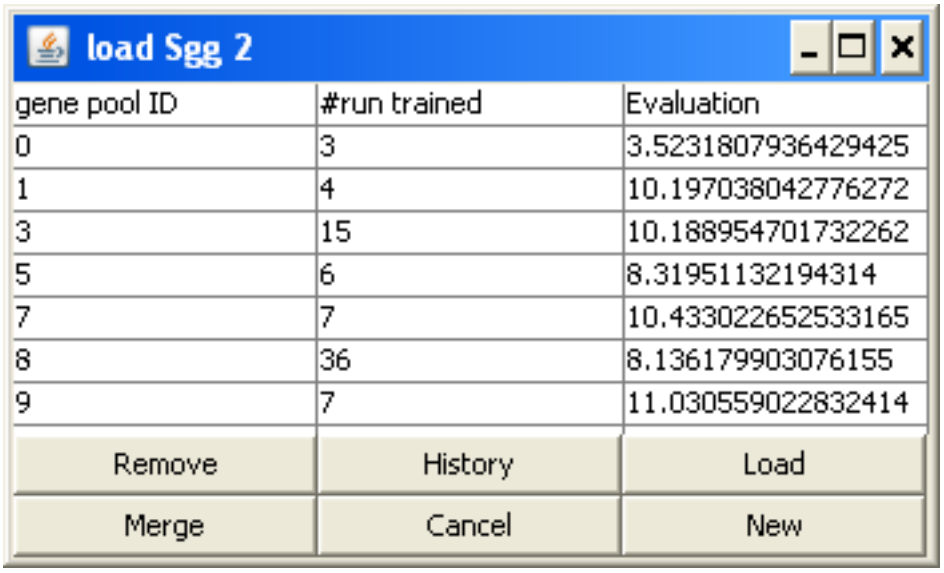
Select the Trainer tab or open it from the Window menu. This window consists of settings to control the training process and statistics to show training progress. The list in figure 8 shows an overview of all the controllers that could be a neural network controller.


Agent selection

Agents

Agent	Sgg	Enhance	Assign	Path planner
Bulldozer: a	Sgg 2	Empty	Nearby	A-star
Bulldozer: b	Sgg 2	Empty	Nearby	A-star
Bulldozer: c	Sgg 2	Empty	Nearby	A-star
Bulldozer: d	Sgg 2	Empty	Nearby	A-star

Figure 6: A list of all the created agents with their respective settings.


load Sgg 2

gene pool ID	#run trained	Evaluation
0	3	3.5231807936429425
1	4	10.197038042776272
3	15	10.188954701732262
5	6	8.31951132194314
7	7	10.433022652533165
8	36	8.136179903076155
9	7	11.030559022832414

Remove | History | Load

Merge | Cancel | New

Figure 7: Create new or load existing subpopulations and assign them.

Networks

parent	module	network id	train?
Bulldozer: a	Nearby	None	Train
Bulldozer: b	Nearby	None	Train
Bulldozer: c	Nearby	None	Train
Bulldozer: d	Nearby	None	Train
Sgg	Sgg 2	1	Continue
Enhance	Empty	None	Train

Figure 8: To continue training previously trained subpopulations, set the desired module on 'Continue'. This is a requirement for both subpopulations loaded from disk and newly trained subpopulations within the same session.

Figure 9: Progression chart with each bar representing the average fitness of a generation.

Setting a neural network controller to 'Continue' allows the user to continue a training session. When subpopulations were loaded during controller selection as discussed in 2.3, the corresponding module in figure 8 should be set to continue. If manual training has been used to train the subgoal generator subpopulations, the Sgg module should be set to continue if training with ESP is the next step.

The remaining settings control the conditions for the ESP algorithm to keep evolving. The first requires a minimum amount of improvement of the next generation's average fitness value. The second sets a maximum number of generations that can be evolved without improving before it stops. And the third sets the average fitness value a generation needs to exceed before it is done. A combination of these settings can be saved by adding a strategy step. For each new strategy step, these settings can be modified. The algorithm will work through the list of steps from top to bottom.

At the bottom of the screen, the statistics keep track of the number of generations that have been evolved when running the ESP algorithm, the highest average fitness a generation has obtained, a progress bar counting the number of simulation runs for each generation and a graph showing the average fitness scores for all generations trained thus far (see figure 9).

2.5 Simulator

The simulator shows the changing of the world state caused by environmental conditions (fire propagation) and the work done by the agents. The simulation window shown in figure 10 can be used to view the running of a simulation. During training the view can be switched freely to and from the progress information shown on the trainer tab.

The main toolbar shows a number of coloured buttons that can be used to control the various training options. Pressing the green play button starts the ESP algorithm, while the red stop button stops all training activity. The blue play button performs one simulation run. For each neural network controller, the neurons used in the neural network are randomly picked from the subpopulations, each time the simulation runs. This will result in different neural networks creating different solutions every run. Running a simulation once can be used to test a sample from the subpopulations, or to test a manually created solution. Information relevant to the training process will be printed in the output pane.

The backpropagation algorithm is started with the pressing of the blue 'skip backward' button. This will train part of the subpopulations to 'fit' the solution followed by storing them. To continue training the subpopulations with the ESP algorithm (as was done in the experiments), the subgoals need to be removed from the map in the world builder and the subgoal generator module needs to be set to continue on the trainer tab.

When the subpopulations are trained and a neural network has been set (use the blue play button to build a neural network), the generated solution can be viewed repeatedly with help of the control buttons on the lower toolbar. Use the blue play, stop and reset buttons in the lower toolbar to respectively start, pause and reset a simulation. It is also possible to adjust the playback speed. Figure 11 shows the simulator in action.

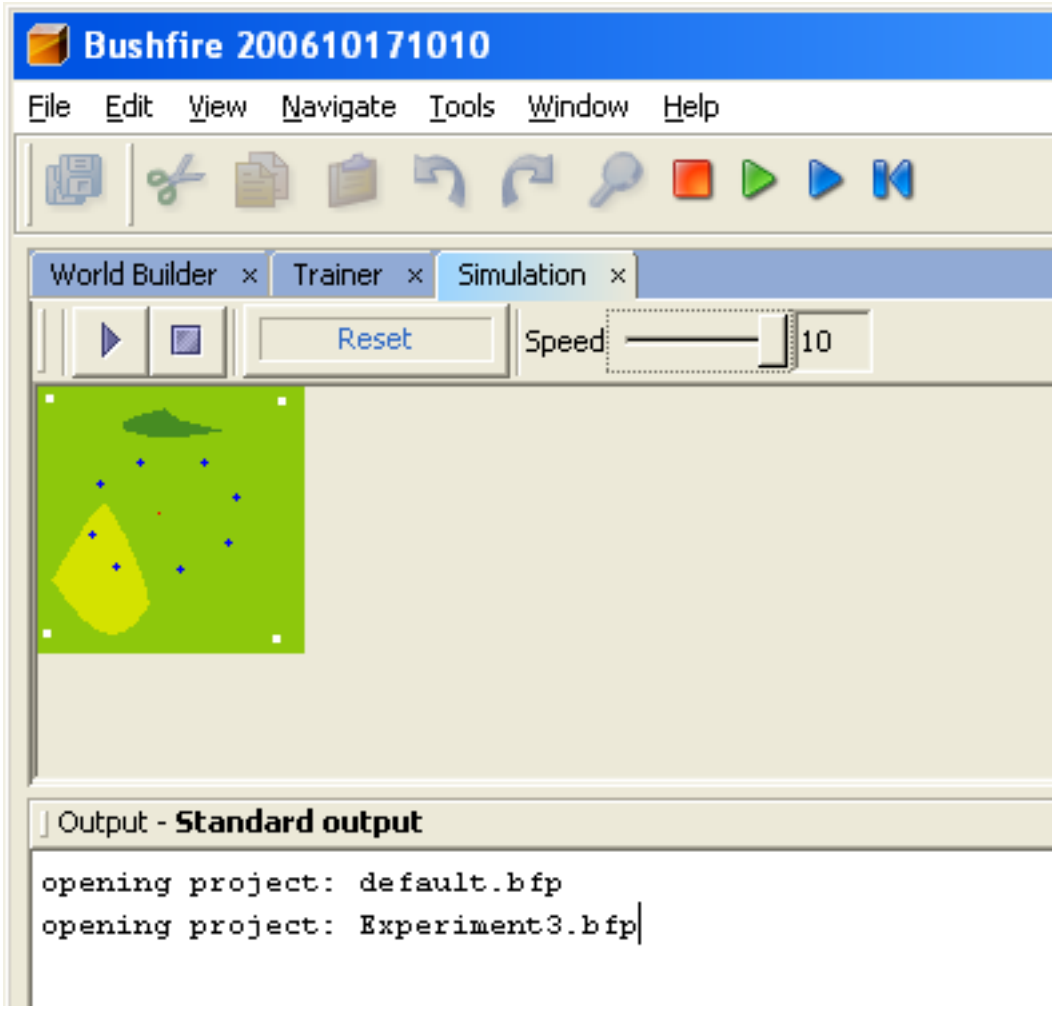


Figure 10: The 'Simulation' view shows the actual running of the simulator. The control buttons in the upper toolbar control the training of the subpopulations. The control buttons in the lower toolbar can be used to watch the last generated solution at different speeds.

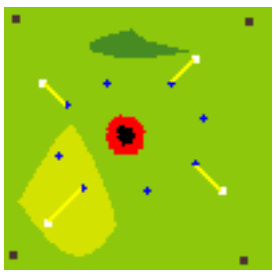


Figure 11: Playing the simulator shows the bulldozer agents in action.

2.6 Hard Coded Settings

All the relevant settings for using Bushfire can be managed through the user interface. However it might be desirable at some point to change some of the hard coded settings. Settings that define the architecture of the neural network, set the size of the subpopulations or the pace of the evolutionary process are examples of some of these settings. For the sake of completeness, code fragments of classes with key attributes are listed at the end of this chapter.

Program 1 SGG2 module settings in class SGG2.

```
class SGG2Settings {
// Nr of genes in each subpopulation
genePoolSize = 30;
..
// Evolution settings:
..
selectPerc=.25; // 25% selection
linMutation=.33; // 33% linear mutation
..
// Neural network values:
neurontype=2; // sigmoid hidden neurons
inputSize = 5; // size of input vector
outputSize = 1; // nr of output units
hiddenUnitSize[0] = 8; // nr of hidden units
}
```

Program 2 ESP settings in class MultiAgentESP.

```
class MultiAgentESP {
..
// the number of simulation runs that are processed each generation
// each simulation run will evaluate a newly constructed neural network
aantalRuns = 250;
..
}
```

Program 3 Backpropagation settings in class BackProp.

```
class BackProp {
..
// 30 genes in a subpopulation
// to get 50% of them trained, we need 15 runs
public static int nrOfNeuralNetworksToTrain = 15 ;
public static int numEpochs = 15; // number of training cycles

// input = {windForce,windVector,humility,fuel,threshold}
public static int numInputs  = 5; // size of input vector
public static int numHidden  = 8; // number of hidden units
public static double LR_IH = 0.17; // learning rate hidden unit
public static double LR_HO = 0.07; // learning rate output unit
..
}
```

3 Underlying Mechanics

This chapter will inform of the techniques used in this project. The core of the Bushfire project entails its use of neural network based controllers and the two provided methods to train these. The first part of this chapter focuses on the basic aspect of neural networks. [HD99] provides additional information on using neural networks as a controller. Training can be done in a supervised and reinforced way by use of the Backpropagation algorithm and the neuroevolutionary algorithm: Enforced Sub-Populations (ESP) respectively. Both will be discussed separately and will provide a general description of its principles.

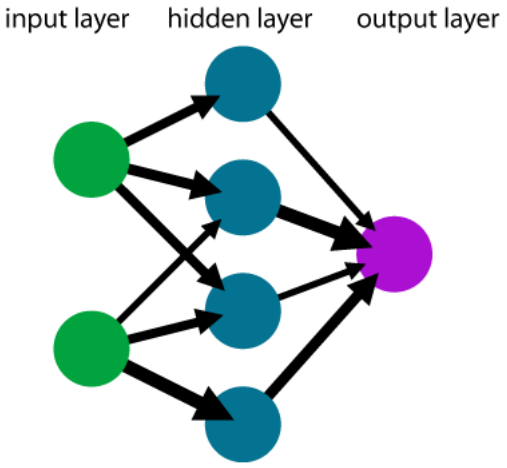


Figure 12: A neural network is an interconnected group of nodes, akin to the vast network of neurons in the human brain. Image from [Wik07a]

3.1 Artificial Neural Networks

Artificial neural networks consist of simple processing elements (the neurons) connected together to exhibit complex behaviour. The idea (and name) is inspired by the way the central nervous system works. An example of an artificial neural network is depicted in figure 1. Note that for this particular neural network the information flows one-way through the network from input to output, this attribute is called feed-forward.

Definition 1 *'An artificial neural network (ANN), often just called a "neural network" (NN), is an interconnected group of artificial neurons that uses a mathematical model or computational model for information processing based on a connectionist approach to computation. In most cases an NN is an adaptive system that changes its structure based on external or internal information that flows through the network.' [Wik07a]*

Neural networks can be viewed as an extension of conventional techniques in statistical pattern recognition [Bis95]. The goal of pattern recognition systems is to make good predictions for new data based on the observations that have been made. A neural network improves its

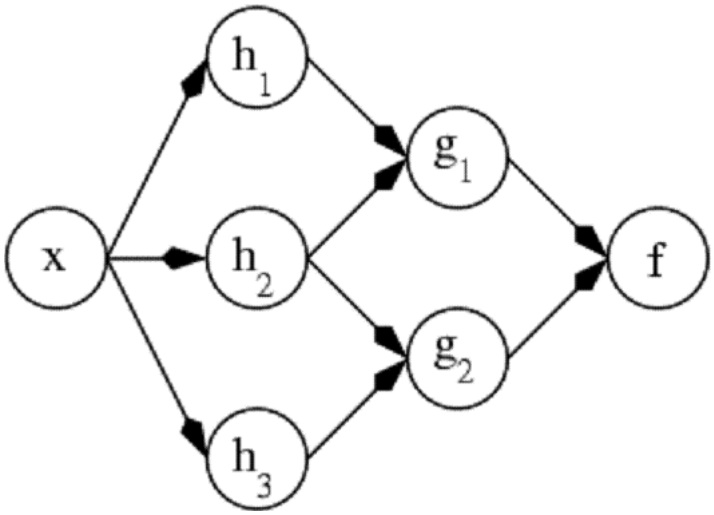


Figure 13: The arrows depict the dependence of a neuron in a neural network. A neuron can only calculate its output if it knows the result of the neurons it depends on. Image from [Wik07a]

performance not unlike polynomial curve fitting. Fitting a polynomial function to a set of data points (the training set) by minimising an error function. Neural networks can be used to find patterns in data or to model complex relationships between inputs and outputs. Therefore in general a neural network models a function:

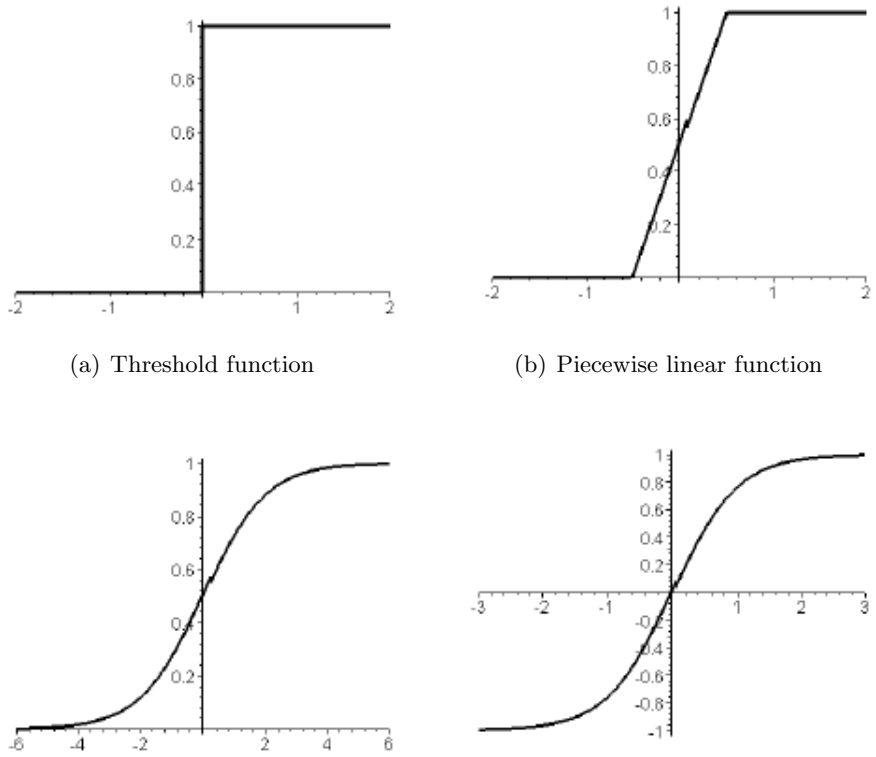
$$f : X \to Y \tag{1}$$

which maps input X to output Y. This function is defined as a composition of other functions, which can further be defined as a composition of other functions. This can be conveniently represented as a network structure, with arrows depicting the dependencies between variables (see figure 13).

Each connection in the network has a value associated with it called a weight. Each node (or neuron) in the network represents a function that calculates its weighted sum by summing up the product of the inputs and weights of the incoming connections. This result is then evaluated by use of a so-called activation or transfer function. The evaluated value is then passed on to the succeeding node(s). In formal writing a neuron calculates its weighted sum y_k:

$$y_k = \phi \left(\sum_{j=0}^{m} w_{kj} x_j \right) \tag{2}$$

with m the number of inputs with output x and weight w associated with neuron k and ϕ being the activation function.



(a) Threshold function

(b) Piecewise linear function

(c) Sigmoid function

(d) Hyperbolic Tangent function

Figure 14: Activation functions.

3.1.1 Activation Functions

Each neuron uses an activation function to calculate the value it passes on to the next neuron(s). The use of activation functions in hidden neurons is essential to have a neural network represent nonlinear functions. They can be divided into the following types [Csa01]:

Threshold (Step) function:

$$\phi(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{3}$$

The Piecewise linear function:

$$\phi(x) = \begin{cases} 1 & x \geq \frac{1}{2} \\ x + \frac{1}{2} & \frac{1}{2} \geq x \geq -\frac{1}{2} \\ 0 & -\frac{1}{2} \geq x \end{cases} \tag{4}$$

Both the Threshold and the Piecewise functions are discrete and produce a limited set of outputs. In case of the threshold function it is often said that the neuron fires when the result is 1. The continuous and non-linear Sigmoid function is the most commonly used activation function.

Sigmoid function:

$$\phi(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

All these functions calculate values ranging from 0 to 1. Sometimes it is desirable to have a range from -1 to 1. In this case the hyperbolic tangent function can be used:

Hyperbolic Tangent function:

$$\phi(x) = tanh(x) \tag{6}$$

Both the sigmoid and the hyperbolic tangent functions have the desirable property that they have a continuous derivative which allows the use of algorithms that use gradient descent as a learning technique (like backpropagation). Figure 14 shows the functions graphically.

3.1.2 Feed-forward Neural Networks

The feed-forward neural network is perhaps the most popular network architecture and is discussed at length in numerous articles [Sta03, Bis95]. In the feed-forward neural network, the information moves in only one direction, from the input nodes, through the hidden nodes (if any) to the output nodes. There are no cycles or loops in the network. Figure 12 and 15 both show examples of feed-forward neural networks. Note that the neurons within a layer are independent of each other. The least complex form of this kind of neural network is a singlelayer perceptron network, which consists of a single layer of output nodes. The sum of the products of the weights and the inputs is calculated in each node, and if the value is above some threshold (typically 0) the neuron fires and takes the activated value (typically 1), otherwise it takes the deactivated value (typically -1). The multilayer perceptron is a feed-forward neural network consisting of more than 1 layer. The additional layers are called hidden layers. A multilayer perceptron with a nonpolynomial activation function can approximate any continuous function with real numbers as input and output. Multilayer networks can be trained in many ways. A commonly used training or learning algorithm is backpropagation.

3.2 Supervised Learning

Supervised learning is a technique for creating a function from training data. The training data is a set of matching input/output pairs. Generalising over this set can be used to predict output values for any given input. In the case of neural networks, creating a function equals adjusting the neuron weights. When using supervised learning, special care needs to be taken to not overfit the training data. Overfitting leads to a function that predicts well within the training set, but does a poor job of predicting output outside the training set.

3.2.1 Backpropagation

So far we have seen how neural networks can calculate some value by processing given input through a network structure of neurons which multiply a weight with their input(s) and apply an activation function to the result. But for the neural network to approximate a desired function we need to find the correct weights. If we have a training set available with some input and the matching correct output we can use supervised learning to train the neural

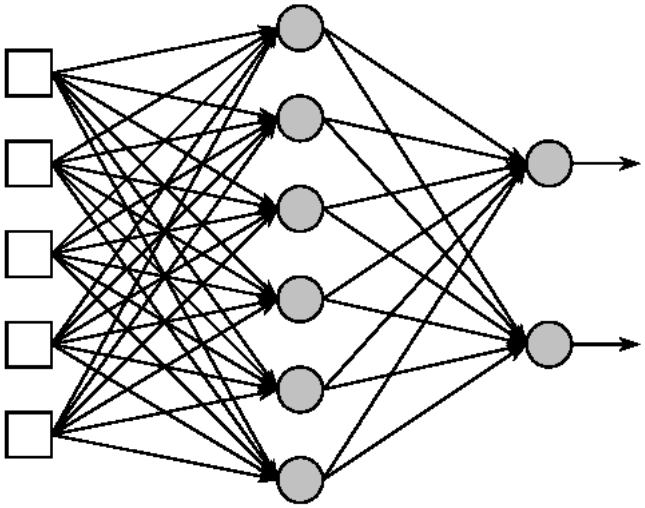


Figure 15: Feed-forward multilayer perceptron with 1 hidden layer.

network. Backpropagation is a supervised learning technique that is classified as a gradient descent algorithm. The algorithm follows the following basic steps.

Backpropagation algorithm [Wik07c]:

1. Present a training sample to the neural network.
2. Compare the network's output to the desired output from that sample. Calculate the error in each output neuron.
3. For each neuron, calculate what the output should have been, and a scaling factor, how much lower or higher the output must be adjusted to match the desired output. This is the local error.
4. Adjust the weights of each neuron to lower the local error.
5. Assign 'blame' for the local error to neurons at the previous level, giving greater responsibility to neurons connected by stronger weights.
6. Repeat the steps above on the neurons at the previous level, using each one's 'blame' as its error.

Backpropagation propagates the error back through the network and for each neuron an error function is calculated. Backpropagation adjusts each weight by descending the slope (or gradient) of the error function. [Hea05] shows examples of how the algorithm could be implemented. Backpropagation is mainly used on multilayer neural networks with nonlinear activation functions in the hidden layer. Multilayer neural networks with linear activation functions are equivalent to singlelayer neural networks which can be trained by the delta rule. Backpropagation is computationally efficient and has a runtime that is linearly dependent on the number of weights [Bis95].

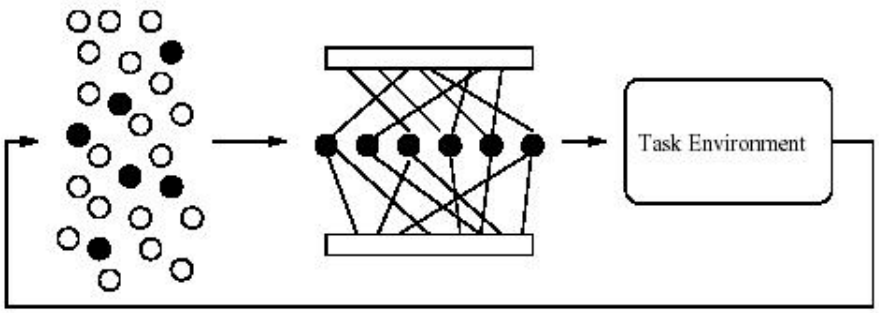


Figure 16: Symbiotic, Adaptive NeuroEvolution (SANE) [MM96]. The population consists of hidden neurons, which are randomly chosen to form networks. Networks are then evaluated and fitness is distributed among all participating neurons. After all neurons are evaluated this way, recombination is performed in the neuron population.

3.3 Neuroevolution

There are many real-world learning tasks that are hard or impossible to solve with the use of supervised learning methods like backpropagation. The problem lies in the absence of a good training set. Controlling a robot in a dynamic environment or playing games against opponents are both examples in which a given training set will not train a neural network to act desirably outside this small domain. Measuring the performance of the neural network is in general very easy for these kind of problems. The field of reinforcement learning takes a learning by trial and error approach, by performing actions and rewarding them according to the perceived result. Evolutionary Reinforcement Learning or Neuroevolution is a subfield hereof that applies genetic algorithms to train artificial neural networks. In general it constructs a neural network by selecting genes from a population and these genes act as the neurons in the network and their chromosomes encode the weights of their connections. Depending on how well the network has performed, a fitness value is given to each of the participating genes before throwing them back in their population. This process is repeated a number of times to ensure each gene had a chance to prove itself. Finally the population is evolved by use of some genetic operations like selection, crossover and mutation creating a new generation of fitter building blocks for the neural network. After a number of generations the population will converge to some (sub)optimal level in which the algorithm measures no more improvements and stops. An example of a neuroevolutionary algorithm whose method closely resembles the one described above is SANE [MM96], which stands for Symbiotic, Adaptive NeuroEvolution and can be considered the predecessor of the neuroevolutionary algorithm ESP. A short description of SANE is shown in figure 16. Other neuroevolutionary algorithms distinguish themselves by evolving the topology of the neural network in addition to the weights, like NEAT (NeuroEvolution for Augmenting Topologies) [SM02] which can be considered as a successor to ESP in some aspects. These algorithms are sometimes referred to as TWEANNs (Topology and Weight Evolving Artificial Neural Networks).

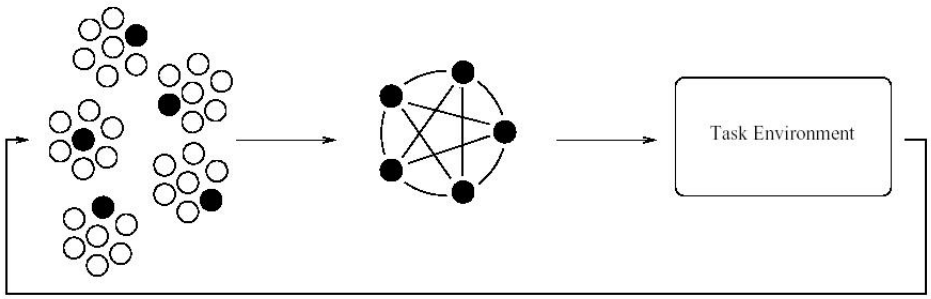


Figure 17: The Enforced Sub-Populations Method (ESP) [GM99]. The population of neurons is segregated into subpopulations shown here as clusters of circles. The network is formed by randomly selecting one neuron from each subpopulation.

3.3.1 Enforced Sub-Populations (ESP)

ESP is an algorithm based on the aforementioned SANE [GM99]. The neurons in a neural network are again represented by genes. Evaluation of the neural network on some given problem results in the genes to be assigned fitness appropriate to their combined performance. But where in SANE the genes were still united in one big population, in ESP the genes are divided among subpopulations (see figure 17). Each subpopulation has a fixed place in the neural network appointed to it and evolution is restricted to the confines of its genepool. This will structurally force each subpopulation to specialise for a place in the neural network. The following procedure taken from [GM99, GM98] describes in detail the workings of the ESP algorithm:

1. Initialisation. The number of hidden units u in the networks that will be formed is specified and u subpopulations of neuron chromosomes are created. Each chromosome encodes the input and output connection weights of a neuron with a random string of floating point numbers.

2. Evaluation. A neuron is selected from each subpopulation at random to form a network. The network is submitted to a trial in which it is evaluated on the task and awarded a fitness score. The score is added to the cumulative fitness of each neuron that participated in the network. This process is repeated until each neuron has participated in an average of e.g. 10 trials.

3. Recombination. The average fitness of each neuron is calculated by dividing its cumulative fitness by the number of trials in which it participated. For each subpopulation, the neurons are then ranked by average fitness. Each neuron in the top quartile of its subpopulation is recombined with a higher-ranking neuron in the same subpopulation using crossover and low-probability mutation. The offspring is used to replace the lowest-ranking half of the population.

4. Goto 2.

ESP has been successfully used to solve many interesting control tasks. In [GM98, GM99] ESP was successful in training controllers that could balance a pole in a 2D and 3D plane indefinitely. In [BM03] ESP was used to evolve an Adaptive Team of Agents (ATA), a system of homogeneous agents that learned to fulfil specific roles depending on their environment, in a sense learning to work as a team. In [WMM05,WM03] it was used to train multiple controllers in a multi-agent forest fire simulation. Evolving these controllers led to the successful solving of numerous forest fire scenarios by the firefighting agents. And in [GM03] ESP was shown to be capable of controlling an unstable (but aerodynamic) finless rocket accurate enough for it to outperform unguided rockets with fins in a detailed simulation model. ESP has proven itself as a great tool for solving real-world nonlinear control tasks on numerous occasions.

4 Combining Manual Training and Enforced Sub-Populations

In this chapter we will first discuss the introduction of manual training in Bushfire and how it is implemented to work in support of ESP. [WMM05, YM07] stated that the strength in ESP lies in its ability to find an initial good solution and less so for improving afterwards. This chapter discusses the effect of preprocessing the genepools on ESP. It will also be used to express some thoughts on mixing different learning approaches in other domains.

4.1 Putting Expert Knowledge in the Initial Subpopulations

In Bushfire a solution consists of eight points, surrounding a fire. These are the subgoals that agents use to plan their digging path. To calculate the location of a subgoal, the neural network calculates an offset from the fire front by processing an input vector with information from the simulation. Every subgoal direction has a specific input vector associated with it and it provides information about wind (force and direction), humidity and local cell data (threshold, fuel). Summarising, a neural network calculates eight distances from the propagating fire front for eight directions around the fire.

When an end-user of the application creates a solution by placing eight subgoals around the fire, it provides the application with eight offsets from the fire front. Since in addition to these outputs, the corresponding inputs can be extracted from the simulation, it allows for supervised learning of the neural network.

Activating backpropagation creates a new neural network from the randomly initialised subpopulations and starts the Backpropagation algorithm to teach the neural network to approximate the given subgoals. When the difference between the provided and calculated output (the error) of the neural network is sufficiently small the neural network assigns the fitness of the solution to the genes that are participating in the network as neurons. The genes are thrown back in their respective subpopulations and new genes are randomly selected to create a new neural network. The process of backpropagation is repeated until a part of the gene population has been trained to represent the user-provided solution. We now have enriched subpopulations and can continue evolving it with the enforced subpopulations algorithm.

4.2 Incremental Learning

Promising results were shown in [WMM05] using incremental learning. Tasks that were unsolvable by ESP when training started with an untrained population of genes became solvable when using a set of genes that had previously solved an easier task. Good results were obtained by training the same population on increasingly difficult tasks. Tasks were made more difficult through the lowering of the cell threshold i.e. the amount of fire activity required to start a cell burning or by increasing the speed of the fire propagation. If the shape of the burning area and the shape of the solution does not change, ESP needs only to scale the outline of the solution to contain a faster spreading fire. Solutions found for easier tasks can therefore be quite fitting for a harder task especially if the former solution did not contain the fire too tight. Considering such a solution that works for a more difficult task, it is much harder to find it in the solution space by ESP when the fire propagation is faster since all solutions that are too tight are now assigned a low fitness, because it fails to stop the

spreading of fire. This makes for a smaller set in the solution space. Even when a population is trained towards a solution unfit for containing a faster spreading fire, evolving from that locale in the solution space is a good strategy to find a solution that succeeds. Experiments for this thesis researches the effect of incremental training using populations trained by the user.

4.3 Balancing the Amount of Training

Depending on the number of trained genes in the enriched subpopulations, the result of successive training with ESP differs significantly from training a random untrained population of genes. In fact, care must be taken to leave enough untrained genes in the initial populations to ensure reasonable chance to cover the solution space. Populations with great majority of their genes trained to approximate one solution will transform ESP from a problem-solving system to a problem-optimising one, converging directly to the user-provided solution and tweaking it to gain a slightly higher fitness. Ideally the introduction of trained genes results in quicker convergence to a good solution and still be able to find other good solutions in the process. With the right balance between trained and untrained genes, ESP should be able to cover enough solution space to retain its usefulness as a problem solver. The information encoded in the trained genes is essentially defining the direction ESP is searching for a solution.

4.4 Biased Solutions Lead to Suboptimal Solutions

In contrast to solutions found through evolving an initial random population, the solutions provided manually are always biased. Human expertise can be driven by many factors like common sense, experience with similar tasks or a developed strategy. The main side effect is that experts discard a lot of approaches that appear counter intuitive. Sometimes the optimal solution is structurally different from the suboptimal ones created by the user. Chances become smaller for a neuroevolutionary algorithm like ESP to find this optimal solution when evolution starts from a population that has been trained in the direction of another suboptimal solution.

In addition the main strength of neuroevolution, the ability to evolve from scratch a working strategy, is being sacrificed in favour of speed and optimisation. Depending on how biased a population is before evolving it, the magic of neuroevolution will diminish when the results become predictable. Of course, predictable behaviour is not always equal to unintelligent behaviour. And if some measure of creativity remains in the form of a nontrivial amount of pruning of the solution space, then the end result may be a healthy combination of common sense and innovation.

4.5 Decision Support Systems

The weight the expert provided knowledge should have in the process of creating a neural network that solves a problem in a certain domain, depends on the role of the system and the nature of the problem. [WD98] proposed Bushfire as a decision support system. Results obtained from solving scenarios could be used as a tool for experts to help decide the right course of action. In the role of a decision support system the aspect of biased results has

different consequences. The bias introduced into the population is already a part of the decision making process, because of its presence in the human expert. Consulting Bushfire can either lead to confirmation or small improvements if the user provided solutions were good and biased towards a certain strategy or to suggesting a strategy if the expert provided diverse and unbiased solutions.

4.6 Games

When using neuroevolution to evolve an intelligent system that has to make decisions on its own in a complex dynamic environment, the introduction of bias in the process should depend on the domain of the problem. For if we want a system to behave intelligently and in the process perhaps outsmart other systems or humans it may have to act open-minded concerning low fitness solutions or strategies.

In computer games there is often the need to have the computer controlled players act intelligently. In some cases this requires the computer to be able to develop a complex strategy e.g. winning a game of Othello. Intelligence in this case could be perceived as the level of difficulty for the player to beat the computer. In other cases it requires the computer to express human-like behaviour e.g. in a first person shooter like Quake, players prefer a computer controlled opponent (bot) that tries to outsmart the player through clever choices and unpredictable navigation opposed to being outgunned by a mechanically moving bot with perfect aim. Different types of games require different types of intelligence. In the next sections the different demands that exist for computer games will be elaborated with research examples.

4.6.1 Othello

Othello is a Japanese board game derived from the better known Go. It has the nice game property of being easy to learn but hard to master. Two players play the game by placing white and black pieces on an 8x8 board. Each placement of a piece requires the conformation of all pieces that are bracketed in any of the three directions, to the colour of the placed piece. So a line-up of black piece, white piece, black piece would turn into three black pieces if any of the black pieces were placed. The winner has the most pieces of his colour on the board when no more pieces can be placed i.e. the board is full.

The standard approach to developing a computer controlled Othello player, would use search and tree pruning e.g. alpha-beta search, which evaluates a large number of game scenarios and decides its action based on this evaluation. It depends on the notion that short term gain will lead to long term gain. A good short term gain strategy in Othello exists and is known as the 'positional' strategy. Unfortunately for alpha-beta search, in Othello the most successful known strategy named 'mobility' focusses on controlling the centre of the board and postpones capturing pieces to the last stages of the game. Alpha-beta search would need to search very deep to find such a strategy and because of the exponential growth of the search time relative to the depth of the search, this is not efficient.

In [ASM02] the Neuro-Evolution through Augmenting Topologies algorithm (NEAT) is studied as an neuroevolutionary alternative to the search and tree pruning alpha-beta search approach. NEAT trained its subpopulations against a random mover and quickly developed

the positional strategy. Against alpha-beta search a weak form of the mobility strategy was eventually developed winning matches 22% of the time. It required more than 500 generations of evolution steps before the mobility strategy was found. Proposals made for future work in [ASM02] indicated the importance of seeking out the mobility strategy and protect it until it has fully evolved. Some form of manual training could be helpful in this endeavour. It would be interesting to see how subpopulations made familiar with the mobility strategy through use of some form of manual training would evolve in the same experiment.

4.6.2 Quake

Quake I, II and III are well known and popular games in the first person shooter genre. In a first person shooter, the player views a 3D world through the eyes of a soldier in a first person perspective. This world is divided into different parts called maps. Goal of the game is to navigate through each map, collect items distributed throughout the map that augment the player's abilities (like weapons and power-ups) and ultimately kill the computer controller opponent (bot). The bot has a similar goal and will also move through the map, collect items and seek out to reduce the player's health to nothing.

The bot requires three controllers that each control independently the navigation, aiming and strategy of the bot. The strategy controller decides what actions the bot should take at any given time e.g. move to a certain point on the map, select another weapon, etc. Using a good strategy allows a player or bot to effectively take control of a map and its resources. Players regard the use of a good strategy the main quality of a good player. It is also the quality that is most lacking in current bot behaviour. Bots executing a clever strategy and capable of changing their strategy in realtime would appear very human-like.

Research had shown progress with bots showing human-like navigation throughout a map [TBS04, TBS03b, TBS03a]. Neural network controllers were trained using supervised learning based on human-generated training data. Final results showed bots behaving human-like and imitating actions from the training data performed by human players. Combining this supervised learning method with reinforcement learning could lead to new behaviour. This would allow the discovery of neural networks expressing strategies outside the scope of the training data.

4.6.3 Legion I

Legion-I is a turn-based strategy game or simulator developed by Bryant and Miikkulainen as a testcase for their 'Adaptive Team of Agents (ATA)' [BM03]. Figure 18 shows the Legion-I game. ATA is a system of homogeneous agents with identical control policies that adopt heterogeneous roles appropriate for their environment. In Legion-I the ATA controlled legions of roman soldiers that were assigned the task to defend cities and surrounding land from hordes of barbarians. Each turn a legion could decide to move a square on the map or stay stationary. Barbarians would spawn regularly on the map and were driven by the desire to pillage the cities and surrounding land but repelled from the legions. The ATA controller was evolved using ESP with the shared fitness evaluation depending on the amount of pillaging that took place with more pillaging meaning lower fitness. After 250 generations the legions had adopted to their roles and had learned to cooperate in defending the cities and the surrounding land.

While some legions took to the role of city garrison, remaining legions scoured the land for barbarian hordes.

Legion-I has many similarities with Bushfire. It uses a map with cells of varying values, the agents and their opposition (barbarians or forest fire) move turn-based, agents share their control policy and both use ESP to evolve their controllers. Extending Legion-I with manual training could bring down the number of generations needed to evolve the controller. This could be desirable if ATA controllers would be applied to even larger control problems.

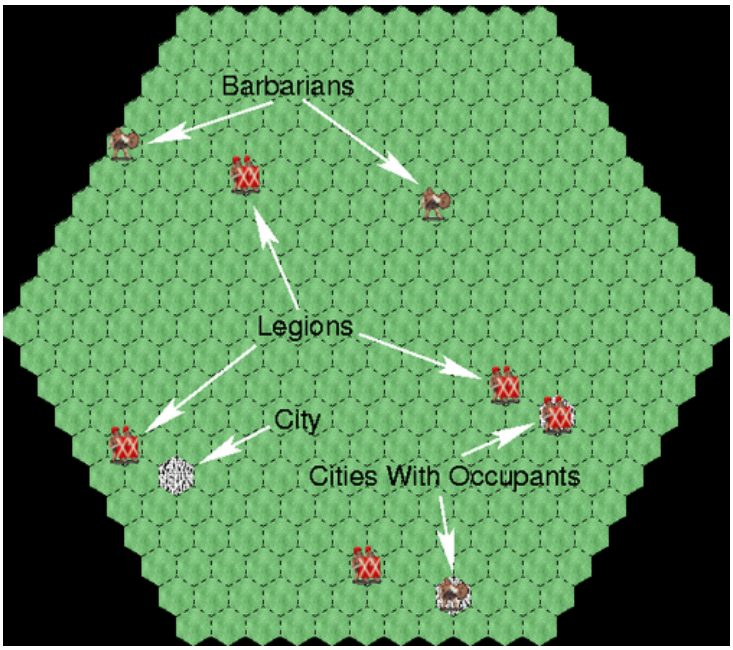


Figure 18: The Legion-I strategy game [BM03].

5 Experiments and Results

This chapter will describe the experiments that have been done and the results that have been observed. Each experiment will focus on a different aspect of the possible scenarios in the forest fire simulator. Each experiment is divided into three parts. First the Enforced SubPopulations algorithm is tested using initial random subpopulations. Points of interest for each experiment are:

- Capability of ESP to find at least one good solution.
- The generation that finds the first good solution.
- The last generation that shows improvement
- The quality of the solutions found and their fitness.

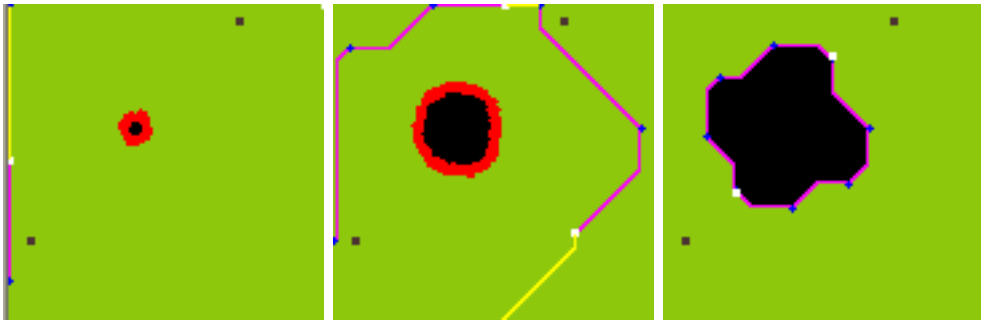
The second part consists of creating a solution by manually placing eight subgoals around the fire. Followed by running the simulation and reporting the fitness.

Thirdly the initial random subpopulations are taught how to make this particular solution. This is done through use of the backpropagation algorithm. From the provided solution, the input (a vector of simulation conditions for each subgoal) and the output (the distance from the fire to each subgoal) are taken for the training set. Genes are taken randomly from their subpopulations (one gene per subpopulation), encoded as neurons and connected to generate a neural network. The input for the neural network is taken from the training set and the neural network produces an output. Based on the difference (error of the neural network) between this output and the desired output from the training set the backpropagation algorithm adjusts the weights of the neurons/genes.

The algorithm needs to run at least 15 times (number of epochs) for each neural network to reduce the error to approximately 0.3. There is no need to decrease the error more under the assumption that the provided solution is most likely not a local optimal solution. In the case of these experiment this was not so. These 'educated' genes are returned to their respective subpopulations. Genes are again randomly selected and the process is repeated. For these experiments the backpropagation algorithm was used on 15 different neural networks, which statistically results in a maximum of 50% of the genes in each subpopulation to be trained. ESP was then started again, but now with the trained subpopulations.

Table 1: Hard coded settings used in the experiments.

Neural Network	Single hidden layer network with 8 hidden and 1 output unit
Activation function	Sigmoid function for hidden units, linear for output unit
Population size	30 neurons
Tests per neuron	10
Fitness assignment	Maximum score obtained
Crossover	1-point crossover at a rate of 50%
Mutation	25% Linear mutation and 33% Gaussian mutation



(a) Initial solution in the first generation.

(b) Solution in the 3rd generation.

(c) Good solution found in the 6th generation.

Figure 19: Solutions found in different generations during the first experiment.

For each experiment, the results of these three approaches will be compared and reviewed. Each experiment is repeated six to ten times and the final listed results are an average hereof.

Some attributes that affect the difficulty of the experiment that are essential for the workings of the algorithms, are constant for all experiments. Each experiment starts with nine randomly initialised subpopulations (also one for the output unit) having thirty neurons each. Neural networks are single hidden layer neural networks with eight hidden units and one output unit. The hidden neurons have a sigmoid activation function and the output neuron has a linear activation function. Every neuron gets on average ten evaluations before evolving to the next generation. This is achieved by the 250 simulation runs between each evolution step. The highest fitness from the evaluations a neuron is part of is assigned to that neuron.

Evolution consists of selection, recombination and mutation. The top 25% of the subpopulations are selected for creating offspring using recombination. One-point crossover and Gaussian mutation with a rate of 33% are the genetic operators used. In addition there is a 25% linear mutation function that runs through the subpopulations when a new generation has marginal improvement. All settings mentioned are summarised in table 1.

The fitness evaluator or function, calculates the fitness of a simulation run based on the relative damage done to the world and the number of edges that have successfully been dug around the fire. A bonus is awarded on top of the fitness if the agents were able to dig 8 edges around the fire. The fitness function shown in 7 calculates a fitness value in the range of [0,12].

$$F = 6\alpha + \frac{\beta}{2} + 2\gamma \tag{7}$$

with $\alpha \in [0, 1]$ the quotient between the sum of cell values before and after the simulation run, $\beta \in [0, 8]$ the number of edges dug and $\gamma \in \{0, 1\}$ the bonus if all edges are dug.

After the first experiment, the fire propagation was kept at the highest level for the remaining experiments.

5.1 Experiment 1

In the first experiment we will train 2 agents to solve a forest fire scenario in a homogeneous environment with different speeds for fire propagation. This experiment is similar to the first experiment done in [WMM05]. The cell threshold, which defines the time it takes for a cell to ignite, is kept low to allow for a fast fire propagation. The time it takes to let an agent drive or dig through a cell is also set at the lowest value to decrease the overall time of running a simulation. The fire propagation speed is set slow at first, 1 step for every 10 steps an agent takes. Each setup is run until the average fitness of the subpopulations stops increasing and converges or until the average fitness rises above 10. This is repeated ten times to get an accurate measurement. Then fire propagation is increased to create a more difficult experiment. Note that in the program, the number of steps fire propagation waits is set and not a speed multiplier. Setting it to 10 corresponds to a fire propagation speed of 0,1 compared to the speed of the agents.

5.1.1 ESP

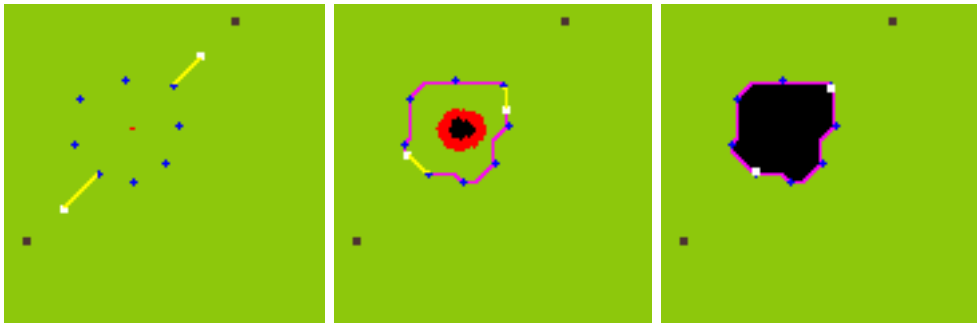
The neural networks from the first generation consist of neurons with randomly chosen weights and as a result perform poorly. These neural networks typically draw their subgoals at the edges of the map, resulting in all of the cells getting burned and agents digging perhaps a few edges at the rim of the map. Solutions with subgoals closer to the fire are rare at first. But after a few generations their number grows and increasingly better solutions are produced. Average fitness of the subpopulations grows with each generation and a very good solution with fitness > 10 is usually first monitored in the 2nd or 3rd generation. For the lowest fire propagation setting, the subpopulations reach a limit with an average fitness of 7,31 around the 6th generation. Figure 19 shows typical solutions observed in different generations. Medium fire propagation speed results in comparable fitness values within the subpopulations, but takes a few generations longer to achieve. Maximum fitness values observed are in both cases near the optimum of 12. Setting the speed of fire propagation to high, lowers the average fitness of the subpopulations that ESP obtains considerably and it takes an additional 2 generations to get to that point. Now, even the best solutions can not prevent the fire from destroying a large portion of the world resulting in a highest fitness score of only 8,65. See table 2 for an overview of the results.

Why won't the average fitness of the subpopulations converge to the maximum fitness?

In some cases the subpopulations do evolve to the point where they can only produce the same neural network over and over again. There is convergence to one solution that produces a high fitness score. In other cases, the final subpopulations produce a few neural networks of different quality and fitness values. The question remains, why the neurons that are part

Table 2: Results for experiment 1, ESP with initial subpopulations.

Fire Propagation	First Good Solution	Best Generation	Avg Fitness	Max Fitness
0,1	2-3	6th	7,31	11,88
0,5	2-3	7-8	7,71	11,96
1	2-3	10-11	5,09	8,65



(a) Agents driving towards the subgoals. (b) Agents digging around the fire. (c) Fire has been stopped.

Figure 20: A manually created solution for experiment 1 with fire spreading slowly.

of a low-fitness neural network are kept alive. A possible explanation could be that the same neurons are also part of a high-fitness neural network when combined with other neurons. These final neural networks created from the subpopulations, usually end up creating solutions a little sparser and a little tighter around the optimal solution. The first results in high fitness values, but the latter results in the agents burning to death and consequently a very low fitness score. This could explain why the average fitness of the subpopulations is so much lower than the maximum fitness. In combination with some local search heuristic, ESP will most likely finish with subpopulations with an average fitness closer to the best results obtained.

5.1.2 Manual Solution

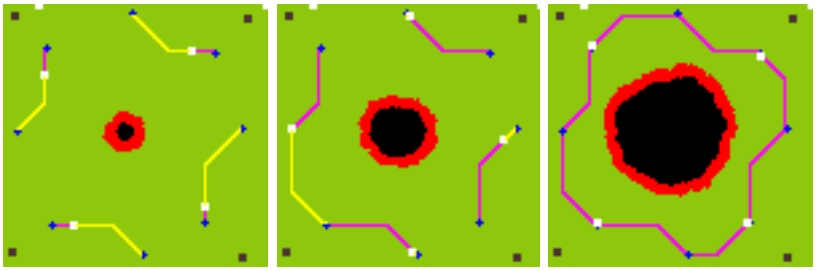
A user can create a solution by drawing 8 subgoals around the fire. Figure 20 shows the solution that has been drawn for this experiment using the lowest possible fire propagation speed. Running the simulation computes the fitness for this solution to 11,55. For medium and fast fire propagation speeds the subgoals had to be moved farther apart resulting in solutions with fitness scores 10,89 and 9,29.

5.1.3 Combined Training

With the use of the backpropagation algorithm these manually created solutions were used to train three initial subpopulations. ESP was subsequently used to further train them for all three corresponding fire propagation settings. The results can be seen in table 3. Again for the low and medium fire propagation speed the results were similar. Good solutions

Table 3: Results for experiment 1, ESP with trained subpopulations.

Fire Propagation	First Good Solution	Best Generation	Avg Fitness	Max Fitness
0,1	1st	5-6	11,07	11,98
0,5	1st	7-8	10,31	11,94
1	1-2	8-9	6,78	7,57



(a) 4 Agents driving towards their subgoals.

(b) Agents digging around the fire

(c) Fire has been contained. Note that the solution is far from optimal.

Figure 21: A manually created solution for experiment 2 with fire propagation set high. This solution has fitness 9,12.

were almost always found in the first generation, one generation faster compared to running regular ESP. The subpopulations obtained an average fitness value > 10 as soon as the 6th and 7th generation, substantially more than the average fitness seen in table 2. Maximum fitness observed in each test was again near optimum. With a high fire propagation a good solution is seen in the 2nd generation, one generation faster with help of the trained genes. Convergence takes place in the 8th or 9th generation, faster than the ESP runs with untrained subpopulations, and again with a higher average fitness. The maximum fitness observed in these runs however, was on average quite a bit lower.

5.2 Experiment 2

The second experiment was designed to pick up where the first experiment would end. A homogeneous environment with low threshold cells and high fire propagation. With the addition of 2 more agents and depots, the time invested in digging will be halved. This should lead to new and better solutions.

5.2.1 ESP

Tests were first done using ESP and untrained subpopulations. Comparing the top results from table 4 with the bottom results of table 2 confirms the assumption that the addition of 2 more agents leads to better solutions. It does take more time (one generation) with a 4 agent setup to find a first good solution and for the subpopulations to converge. The first is most likely due to the fact that independent of the scenario, a solution with fitness 8 or

Table 4: Results for experiment 2, ESP with initial and trained subpopulations.

Test	First Good Solution	Best Generation	Avg Fitness	Max Fitness
ESP	4	11-12	6,61	11,25
Combined Training	1-2	8-9	9,13	10,92

9 will almost always be found before a solution with fitness 10 or 11 when started with an untrained population. The latter is a result of the better solutions that are now obtainable. Average fitness after convergence takes place is above 6 on average and as stated, the main reason are the better solutions that are now possible, with maximum fitness reaching above 11.

5.2.2 Manual Solution

Figure 21 shows the solution that has been drawn for this experiment using the highest possible fire propagation speed. Running the simulation shows that this solution has fitness 9,12. As is visible in the figure, the subgoals could have been placed closer to the fire which would have resulted in a better solution.

5.2.3 Combined Training

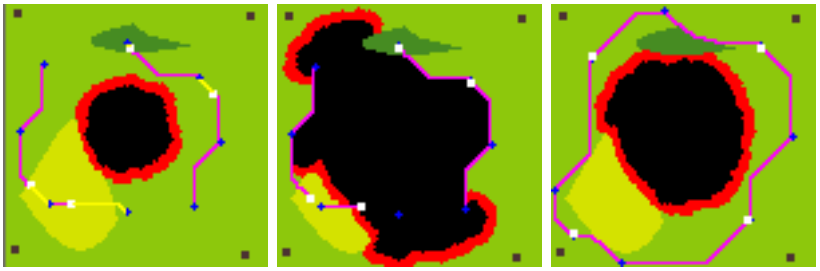
The genes in the subpopulations were trained again using the created solution shown in figure 21 before running the ESP algorithm. The first solutions with high fitness values were seen in the first 2 generations during evolution. Improvement stopped in the 8th or 9th generation, sometimes resulting in an average fitness score > 10, but the average of all tests was 9,13, almost 40% higher than starting with untrained subpopulations. The average of all maximum fitness scores observed were slightly lower, 10,92 compared to 11,25.

5.3 Experiment 3

The third experiment introduces other terrain types, making the environment heterogeneous. The main terrain type remains grass with the same attributes as in the previous experiments, low threshold and fast travelling speed for the agents. The threshold for the hay is kept low to ensure it will be ignited by the neighbouring grass, but driving and digging for the agents will take longer. The trees at the north of the map will have a very high threshold and will remain unaffected by the spreading fire. Driving and digging will take longer through trees as is the case with hay.

5.3.1 ESP

Because the terrain acts like obstacles that slow the agents down, the agents will have to dig around the terrain. Every run that sends the agents through the rougher terrain is destined to fail. Therefore it takes ESP far longer to start generating acceptable solutions than before. At least 7 generations of evolution have to pass before the agents start saving some land from destruction. It takes another 6 generations before the subpopulations stop improving, and only an average fitness of 4,65 is reached. The average fitness is very low compared to the maximum fitness obtained. Table 5 shows the results.



(a) (b) (c)

Figure 22: Experiment 3 uses a heterogeneous environment with grass, hay (light green) and trees (dark green). The images shown in (a) and (b) show that movement through hay and trees is slower and that the trees will not ignite. The manual solution used in this experiment and shown in (c) has fitness 8,77.

5.3.2 Manual Solution

As shown in figure 22 (c) the manual solution has been drawn around all the rough terrain resulting in a fitness value of 8,77. Any attempts to make a better solution resulted in the agents dying.

5.3.3 Combined Training

After training the subpopulations with the backpropagation algorithm and the user solution we continue training with ESP. The third generation shows the first solutions similar to the one used as a training set by the backpropagation algorithm. The results in table 5 show that manual training (i.e. training the subpopulations with the help of a user provided solution) decreases the amount of time ESP needs to evolve good subpopulations.

5.4 Experiment 4

This experiment is based on the different values of cells in a heterogeneous world and the effect it will have on the calculated fitness value of a simulation run. Buildings are created with 1000x the cell value of a normal (grass) cell. Finding solutions that prefer saving buildings in favour of large grassy areas is the goal of this experiment.

Table 5: Results for experiment 3, ESP with initial and trained subpopulations.

Test	First Good Solution	Best Generation	Avg Fitness	Max Fitness
ESP	7th	13-14	4,65	8,65
Combined Training	3-4	8-9	7,08	8,33

5.4.1 Manual Solution

The solution shown in figure 23 was used as the manual solution. It has no preference concerning the type of cells it wants to protect.

5.4.2 Results

The results written in table 6 show again that manual training before using ESP results in good subpopulations in less time. But also, the best solutions (with maximum fitness) seem to be getting worse. The goal of this experiment was to obtain solutions that had agents primarily protecting the buildings. This did not happen. Although solutions that protected buildings from the fire, did result in high fitness scores. These solutions went hand in hand with protecting large areas of grass. So the solutions generated by the fully evolved subpopulations, were similar to the ones from experiment 2 (see figure 21). Solutions that had agents protect separated areas at the edges of the map were monitored, but regular solutions outperformed them. A logical consequence from formula 7 that calculates the fitness value, the result partly depending on the number of edges dug.

5.5 Experiment 5

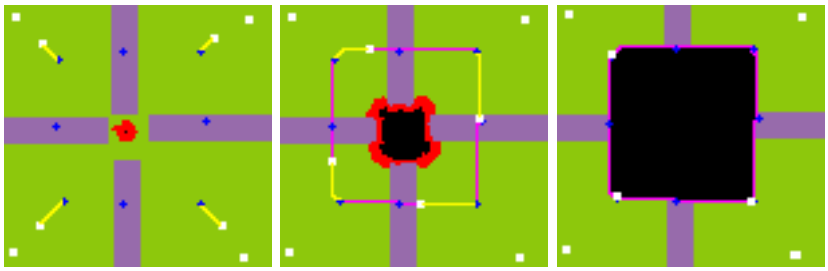
The final experiment was designed to discover the limits of the system. The world consists of grass with a residential area (buildings) in the upper right corner. One agent starts at a depot near the buildings at the top right of the world and multiple sources of fire are started in the other corners. An ideal outcome would have the agent dig a path around the buildings. Figure 24 shows the setup of this experiment, the handmade solution and the solution that a trained neural network produced.

5.5.1 Results

A manual solution was easily created with the drawing of subgoals around the buildings. Because of the high value of building cells relative to grass cells, the protection of this small part of the map lead to a high fitness score of 11,58. Unfortunately, the attempts to train a neural network to reproduce these subgoals were unsuccessful. Neural networks in Bushfire calculate the offset or distance from the fire, where the subgoals should be placed. The direction of the subgoals relative to the fire however are constant. More specifically, there will always be one subgoal to the north of the centre of the burning area, one to the northeast, etc. Therefore training agents in scenarios with multiple fires starting close to an edge of the map, will not result in satisfactory solutions.

Table 6: Results for experiment 4, ESP with initial and trained subpopulations.

Test	First Good Solution	Best Generation	Avg Fitness	Max Fitness
ESP	4th	11-12	5,59	9,84
Combined Training	2-3	8th	7,92	9,49

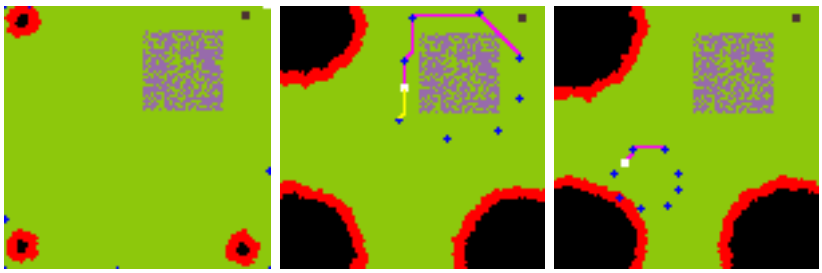


(a) Agents driving towards their subgoals.

(b) Agents are digging a path.

(c) Forest fire has been stopped.

Figure 23: Experiment 4 uses a map with grass and high valued residential areas (in purple). This solution has fitness 9,17.



(a) (b) (c)

Figure 24: (a) The fifth experiment starts fires in three corners of the map and places a town in the remaining corner. (b) The successful execution of a manually generated solution protects the town with a fitness value of 11,58. (c) Distributing the subgoals around the centre of the fires, leads to bad performance by the evolved neural network controllers.

6 Analysis

The first two experiments were designed to test in a homogeneous environment. All cells having equal values has the advantage of having a constant fire propagation that behaves predictably. Increasing the speed of the fire propagation was the only defining factor influencing the difficulty of the problem. The first experiment was repeated three times, increasing the fire propagation each time. The second experiment used four agents instead of two. Both experiments did not, even at the most challenging settings, produce a result that could be described as failure to find a good solution. The best possible solutions were always found in a homogeneous environment.

However, there were differences between the two training methods under inspection. Combined training (i.e. using ESP after manual training) finds its first good solution faster, usually in one or two generations of training. This works as a 'jumpstart' for the evolution process. Resulting in faster convergence of the subpopulations. In many cases the subpopulations specialised to producing the same neural network every time with near optimal solutions.

Regular ESP (i.e. using Enforced SubPopulations with initial random genepools) needed to evolve more generations to find similar solutions, and experienced more problems in converging to the best solutions found. The subpopulations from the best generation still had a chance of generating neural networks producing bad solutions, but additional training did not improve the average fitness or converge the subpopulations. Possible reasons were stated in section 5.1, and the introduction of a local search heuristic could perhaps improve the poor performance of regular ESP when it comes to the reliable production of good solutions. However, regular ESP was better in finding a solution with the highest fitness in both experiments. Even though the maximum fitness score found by both methods did not lie far apart, it does show that the extensive exploration done by regular ESP translates to a bigger chance of finding the best possible solution.

The third experiment was the first setup that used a heterogeneous map. It featured some terrain that had to be circumvented to find a good solution. Even the best possible solutions would have most of the map burned, and would have a fitness value below ten. ESP had a hard time finding solutions to this particular challenge on its own. Taking a minimum of six generations to find a decent solution and improving at a slow pace. The best generation produced solutions with an average fitness almost half of the maximum fitness value found. If the measuring bar is defined as: 'A reliable production of good solutions.' then regular ESP has failed to solve the control problem of this experiment. If the objective is to find at least one good solution, it has succeeded in solving the problem.

Combined training showed better results. It found solutions as soon as the third generation and within nine generations was capable of producing neural networks with high fitness solutions most of the time. Even the highest fitness evaluations were now observed with combined training instead of with regular ESP. In fact the third experiment is the best showcase for combined training. As was shown for incremental learning [WM03], having ESP start with pretrained subpopulations, improves its performance when trying to solve more complex control tasks.

The fourth experiment was designed to test if solutions would be found that discriminate between different kind of cells. To this end, one type of cell (grass) was awarded a very low

value and the other type (buildings) a very high value. Solutions that preferred the protection of 'building' cells would receive a high fitness reward. Initially, interesting solutions that had agents protecting areas with buildings were produced and rewarded properly. But eventually, solutions that stopped the fire altogether would become the dominant strategy. Both regular ESP and combined training showed results comparable to the second experiment and did not provide additional insight.

The fifth experiment shows one of the major limitations in the current design. Namely, subgoals are always placed around the source of the fire, i.e. the direction from a subgoal relative to the fire is fixed. First, this denies the generation of strategies focussed on damage control i.e. agents protecting their own territory. And second, placement of multiple fire sources will result in erratic behaviour by the subgoal generator, denying the design of even more challenging experiments.

7 Conclusion and Future Development

This chapter presents the conclusion of this thesis and some proposals for future work.

7.1 Conclusion

The choice of using manual training before running the ESP algorithm is connected to the 'Exploration vs Exploitation' dilemma. Using the enforced subpopulations algorithm with an untouched random set of genes will allow it to explore the solution space more thoroughly. It will take more time, but it has a better chance of finding that perfect solution with maximum fitness. Using combined training on the other hand favours the exploitation approach. It is guaranteed to find a good solution in a short amount of time, but it does so at the cost of excluding a large part of the solution space.

Where the regular use of ESP fails to solve a problem, combined training could prove to be successful. The third experiment was the best example of a hard control task that showed combined training performing well and had regular ESP struggling. Future experimentation with even harder control problems could prove that combined training is a credible and fast alternative to incremental learning as a tool to solve increasingly difficult problems.

Overall the results obtained from the experiments speak in favour of the use of manual training before starting the ESP algorithm. Running the algorithm to solve a problem takes less time and produces higher quality neural network controllers.

7.2 Future Development

This section will be used to make some proposals for future development and research.

- Applying the combined training approach in other domains. It would be interesting to see how the combined training approach can be used in other domains where a neuroevolutionary problem solving strategy only finds solutions after evolving for relatively many generations.
- Fixing the enhance module and have it work as a local subgoal optimiser in conjunction with the subgoal generator module. This could have a positive effect on the convergence of the subpopulations.
- Redesigning the subgoal generator module to remove the fixed directional placement of subgoals around the source of the fire. This will allow the designer of a forest fire scenario to bring diversity in the problems, e.g. multiple fires could be started from opposite edges of the map or subgoals could be generated everywhere.
- In the current setup eight subgoals are generated and assigned to the available agents. The generation of more subgoals could be desirable if the simulator world would become larger and if goal placement becomes more flexible.
- Multi-agent behaviour could be researched by letting the agents generate their own subgoals and see if at some point they start cooperating by digging towards each other.

- Implementing and experimenting with other algorithms. When the number of subgoals is flexible, the topology of a neural network may need to change with it. Implementing and experimenting with a TWEANN like NEAT [GM99] could be an interesting research topic.
- Modifying the manual training module to allow the user to supply multiple solutions for the training set. This would make manual training more flexible.

7.2.1 Bushfire

Modifying and extending a big application like Bushfire becomes problematic if it is not developed using good design principles [GHJV95]. The author of this thesis has migrated the Bushfire code into Netbeans modules. Netbeans modules are restricted to one-way dependence enforcing a better design and implementation. When changing the old code e.g. rewriting the simulator, an effort should be made to use the module-based approach and improve overall design.

7.2.2 Simulator

Replacing the simulator with a better one will significantly improve the capabilities of Bushfire. The current design relies on heavy memory usage and requires a lot of computational power. A new simulator should be able to cope with thousands of cell operations to allow the creation of advanced maps. Core design principles should be:

- Efficient memory management.
- Lightweight cell based operations.
- User interface independence.

In other words the simulator should become scalable, fast and encapsulated from other parts of the application.

A Design

Rich-client application development strives to relieve the developer of writing code for the requirements that all desktop applications have in common. Such requirements are menus, document management, settings and so forth. The Netbeans platform is a generic desktop application, that can be used as a basis for an application. Development starts with a fully functional desktop application, that can be extended by writing modules and bundling them with the platform.

Bushfire has been migrated on top of the Netbeans platform. It now functions as a module and some of the GUI functionality has been surrendered to the platform. Ideally all the packages that were part of the original Bushfire program would each become their own module, but Netbeans modules require one-way dependency, and the Bushfire packages do not meet that requirement. The new package containing the backpropagation algorithm has been written in a module. It depends on the 'model' module, which contains the original Bushfire packages, but not the other way around. Building new functionality in modules enforces better design. Diagram 25 shows how the modules relate to one another. Relations between the original Bushfire packages, and dependencies between the classes within each package can be found in the original technical documentation. For more information on building on top of the Netbeans platform, visit [Net07a, Net07b].

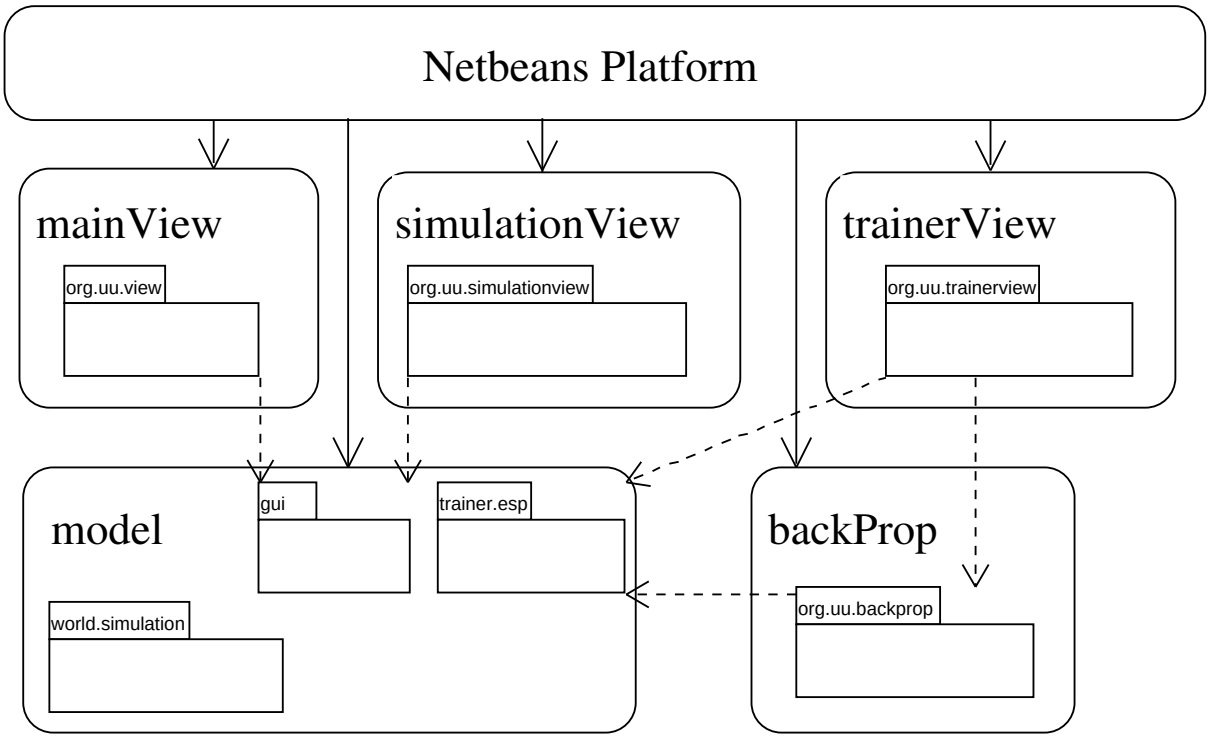


Figure 25: Bushfire is built on top of the Netbeans platform. This diagram shows the dependencies between the different modules.

B Data Experiments

The following tables accumulate all the results obtained during the experiments. Results listed per test are the generation wherein a good solution was generated for the first time (depending on the experiment, the fitness had to be above a value ranging from 8 to 10), the last generation that improved its population, the average fitness of this population and the fitness of the best solution that has been produced.

Table 7: Test results from the first experiment with a low fire propagation set at 10. ESP started with initial untrained subpopulations.

	Generation with First Good Solution	Last Generation	Avg Fitness	Max Fitness
1	3	6	8,32	11,69
2	2	8	6,65	11,83
3	2	8	8,38	11,88
4	2	7	7,08	11,90
5	2	4	6,61	11,99
6	3	4	6,84	11,99

Table 8: Test results from the first experiment with a low fire propagation set at 10. ESP started with trained subpopulations.

	Generation with First Good Solution	Last Generation	Avg Fitness	Max Fitness
1	1	6	10,23	11,98
2	1	4	10,94	11,98
3	1	9	10,45	11,99
4	1	5	11,24	11,97
5	1	5	11,95	11,98
6	1	5	11,59	11,97

Table 9: Test results from the first experiment with a medium fire propagation set at 5. ESP started with initial untrained subpopulations.

	Generation with First Good Solution	Last Generation	Avg Fitness	Max Fitness
1	3	5	7,38	11,94
2	2	8	7,67	11,97
3	2	8	8,45	11,96
4	2	12	7,94	11,98
5	2	5	6,21	11,95
6	2	7	8,58	11,97

Table 10: Test results from the first experiment with a medium fire propagation set at 5. ESP started with trained subpopulations.

	Generation with First Good Solution	Last Generation	Avg Fitness	Max Fitness
1	1	4	10,16	11,94
2	1	6	10,89	11,93
3	1	9	9,44	11,95
4	1	4	10,92	11,95
5	1	12	9,33	11,95
6	1	8	11,14	11,89

Table 11: Test results from the first experiment with a high fire propagation set at 1. ESP started with initial untrained subpopulations.

	Generation with First Good Solution	Last Generation	Avg Fitness	Max Fitness
1	1	13	4,76	8,58
2	4	8	4,93	8,64
3	4	9	4,23	8,63
4	5	10	4,69	8,53
5	3	8	5,02	8,81
6	3	11	5,34	8,64
7	2	17	5,65	8,69
8	2	10	5,33	8,67
9	2	10	5,63	8,63
10	1	7	5,31	8,67

Table 12: Test results from the first experiment with a high fire propagation set at 1. ESP started with trained subpopulations.

	Generation with First Good Solution	Last Generation	Avg Fitness	Max Fitness
1	1	10	6,94	7,74
2	4	6	5,43	7,75
3	1	8	6,78	7,77
4	1	7	6,91	7,98
5	2	12	6,74	7,99
6	1	4	6,61	6,79
7	1	7	6,94	7,52
8	1	10	7,21	7,23
9	1	10	6,97	7,44
10	1	9	7,23	7,46

Table 13: Test results from the second experiment with a high fire propagation set at 1. ESP started with initial untrained subpopulations.

	Generation with First Good Solution	Last Generation	Avg Fitness	Max Fitness
1	4	7	6,31	11,21
2	3	12	6,48	11,26
3	4	12	5,36	11,27
4	3	9	5,86	11,07
5	6	14	6,84	11,22
6	4	13	7,17	11,37
7	3	7	6,93	11,42
8	3	17	6,59	11,36
9	5	12	7,35	11,30
10	5	10	7,16	11,00

Table 14: Test results from the second experiment with a high fire propagation set at 1. ESP started with trained subpopulations.

	Generation with First Good Solution	Last Generation	Avg Fitness	Max Fitness
1	1	6	10,02	10,76
2	2	8	8,96	10,65
3	1	12	7,51	11,42
4	2	14	10,04	10,61
5	2	8	10,17	10,94
6	2	8	10,10	10,67
7	1	4	5,92	10,94
8	2	5	10,10	10,71
9	2	12	10,21	11,15
10	2	7	8,26	11,37

Table 15: Test results from the third experiment with a high fire propagation set at 1. ESP started with initial untrained subpopulations.

	Generation with First Good Solution	Last Generation	Avg Fitness	Max Fitness
1	7	13	6,58	8,76
2	6	9	3,81	8,56
3	6	17	5,03	8,85
4	8	14	4,02	8,45
5	10	11	4,52	8,74
6	9	25	5,89	8,60
7	10	10	3,51	8,57
8	4	13	3,96	8,56
9	6	10	4,54	8,94
10	5	13	4,64	8,49

Table 16: Test results from the third experiment with a high fire propagation set at 1. ESP started with trained subpopulations.

	Generation with First Good Solution	Last Generation	Avg Fitness	Max Fitness
1	3	5	6,98	9,18
2	3	7	7,36	9,19
3	3	12	8,53	8,53
4	3	8	7,08	8,39
5	3	10	8,43	8,97
6	5	5	6,25	8,45
7	1	19	6,27	8,47
8	2	8	7,95	8,03
9	6	8	6,30	8,45
10	-	6	5,62	5,62

Table 17: Test results from the fourth experiment with a high fire propagation set at 1. ESP started with initial untrained subpopulations.

	Generation with First Good Solution	Last Generation	Avg Fitness	Max Fitness
1	3	11	5,96	9,65
2	2	7	6,3	10,52
3	3	17	7,12	10,55
4	3	9	5,55	10,08
5	4	14	6,17	10,48
6	-	14	3,40	7,78
7	5	12	4,33	8,42
8	7	13	5,66	10,64
9	4	8	5,01	9,77
10	4	11	6,44	10,48

Table 18: Test results from the fourth experiment with a high fire propagation set at 1. ESP started with trained subpopulations.

	Generation with First Good Solution	Last Generation	Avg Fitness	Max Fitness
1	3	12	9,1	9,72
2	2	13	8,4	8,4
3	2	4	5,17	9,72
4	2	13	9	9,21
5	1	4	6,75	10,15
6	2	6	8,39	8,92
7	2	6	9,24	9,43
8	-	7	8,47	8,99
9	2	9	6,54	10,41
10	4	7	8,18	9,93

References

[ASM02] T. Andersen, K.O. Stanley, and R. Miikkulainen. Neuro-evolution through augmenting topologies applied to evolving neural networks to play othello, 2002. Honors thesis project.

[Bis95] C.M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[BM03] B.D. Bryant and R. Miikkulainen. Neuroevolution for adaptive teams. In *Proceedings of the 2003 Congress on Evolutionary Computation (CEC'03)*, volume 3, pages 2194–2201. IEEE Press, 2003.

[Csa01] B.C. Csaji. *Approximation with Artificial Neural Networks*. PhD thesis, Eotvos Lorand University, 2001.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[GM98] F.J. Gomez and R. Miikkulainen. 2-d pole balancing with recurrent evolutionary networks. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN-98, Skovde, Sweden)*, pages 425–430. Elsevier, 1998.

[GM99] F.J. Gomez and R. Miikkulainen. Solving non-markovian control tasks with neuroevolution. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99, Stockholm, Sweden)*, pages 1356–1361. Morgan Kaufmann, 1999.

[GM03] F.J. Gomez and R. Miikkulainen. Active guidance for a finless rocket using neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003)*, pages 2084–2095, 2003.

[HD99] M.T. Hagan and H.B. Demuth. Neural networks for control. In *American Control Conference, 1999. Proceedings of the 1999*, pages 1642–1656, 1999.

[Hea05] J Heaton. Website, 2005. `http://www.heatonresearch.com/articles/5/page4.html`.

[MM96] D.E. Moriarty and R. Miikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22:11–33, 1996.

[Net07a] Netbeans. Website, 2007. `http://platform.netbeans.org/`.

[Net07b] Netbeans. Website, 2007. `http://www.netbeans.org/products/platform/`.

[SM02] K.O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[Sta03] StatSoft. Website, 2003. `http://www.statsoft.com/textbook/stneunet.html#multilayer`.

[TBS03a] C. Thurau, C. Bauckhage, and G. Sagerer. Combining self organizing maps and multilayer perceptrons to learn bot-behavior for a commercial computer game. In *Proc. GAME-ON*, pages 119–123, 2003.

[TBS03b] C. Thurau, C. Bauckhage, and G. Sagerer. Learning human-like opponent behavior for interactive computer games. In B. Michaelis and G. Krell, editors, *Pattern Recognition*, Lecturenotes in Computer Science 2781, pages 148–155. Springer-Verlag, 2003.

[TBS04] C. Thurau, C. Bauckhage, and G. Sagerer. Learning human-like movement behavior for computer games. In *Proc. 8th Int. Conf. on the Simulation of Adaptive Behavior (SAB'04)*, 2004.

[WD98] M.A. Wiering and M. Dorigo. Learning to control forest fires. In H.-D. Haasis and K.C. Ranze, editors, *Proceedings of the 12th International Symposium on Computer Science for Environmental Protection (UI'98)*, volume 18 of *Umweltinformatik Aktuell Series*, pages 378–388. Metropolis-Verlag, 1998.

[Wik07a] Wikipedia. Website, 2007. `http://en.wikipedia.org/wiki/Artificial_neural_network`.

[Wik07b] Wikipedia. Website, 2007. `http://en.wikipedia.org/wiki/A*_search_algorithm`.

[Wik07c] Wikipedia. Website, 2007. `http://en.wikipedia.org/wiki/Backpropagation`.

[WM03] M.A. Wiering and F. Mignogma. Learning to control forest fires with esp. In A. Dutech and O. Buffet, editors, *Proceedings of the Sixth European Workshop on Reinforcement Learning*, pages 22–23, 2003.

[WMM05] M.A. Wiering, F. Mignogma, and B. Maassen. Evolving neural networks for forest fire control. In M. van Otterlo, M. Poel, and A. Nijholt, editors, *Benelearn'05: Proceedings of the 14th Belgian-Dutch Conference on Machine Learning*, pages 113–120, 2005.

[YM07] C. Yong and R. Miikkulainen. Coevolution of role-based cooperation in multi-agent systems. Ai07-338, The University of Texas at Austin, 2007.