university of groningen / faculty of science and engineering

Adapting CoSyNE to fight forest fires

Ivo de Jong, ivopascal@gmail.com
Supervisor: Dr M.A. Wiering

Abstract: Limiting the area burned in a forest fire can be done by digging out a circle where the fuel is removed so the fire cannot spread. Determining the size of this circle has previously been solved in simulations where the circle is determined by 8 points, set at varying distances with the neuroevolutionary algorithm ESP and having an agent navigate along these points by following the shortest path. This paper instead controls the distance of the points by an adaptation of CoSyNE, but the main focus lies on the navigation between these points. The performance is compared between shortest path navigation, navigation with an adaptation of CoSyNE, and the same adaption with the addition of dropconnect. These CoSyNE adaptations were given the input of shortest path navigation as a recommendation. While the CoSyNE navigation, particularly with the dropconnect addition, was able to find better solutions for each map than the shortest path, the exploration required for this gave both algorithms a worse average performance on each map.

1 Introduction

Due to the ongoing climate change, the number of forest fires is inevitably going to increase (Stocks et al., 1998). While the fires in and of themselves are dangerous, even the air pollution caused by large areas of forest burning away has been shown to seriously affect the health of local populations (Sastry, 2002). Moreover, the fires release large amounts of stored CO_2 back into the atmosphere, reinforcing global warming. To minimize this problem, and to minimize the number of homes lost, we need strong systems to combat these forest fires.

Forest fires, like all fires, depend on fuel, heat, and oxygen. While many different reactive and proactive approaches are used by different countries (Calabri, 1982), a particularly interesting one is the use of so-called "Fire Lines". These Fire Lines are lines of forest which are cleared of vegetation, which acts as the fuel, so that the fire cannot spread across this line. This can be used as a proactive measure, as done by the Spanish I.C.O.N.A. (Calabri, 1982), but also as a reactive measure. In the latter case, when a fire is detected, lines are mapped out at varying distances around the fire. Each line is assigned to a group of workers who clear the line of fuel. This can be done with for example bulldozers, tractors, or chainsaws. When all the fire lines are successfully cleared, the fire is contained, and so the damage that it can cause is minimized. The tricky aspect of these fire lines is that it can be challenging to determine where these lines must be drawn. Ideally you want to confine the fire to a small area, so less land has to be evacuated. On the other hand, you also need to keep in mind that the fire can spread fast, and that clearing the land takes time. If the latter is neglected and the fire is able to reach the fire line before it is successfully drawn, it puts the firefighters at risk and allows the fire to escape the encapsulation.

While the fire brigades are extensively trained and have a good amount of experience and expertise in fighting these forest fires, the problem does pose an interesting challenge for Artificial Intelligence. It has been proposed that this problem can be solved using Reinforcement Learning (Wiering and Dorigo, 1998), but the problem has also been solved using an evolutionary algorithm called Enforced Sub-Populations (ESP) (Wiering et al., 2005). Since this research, a new evolutionary algorithm called Cooperative Synapse Neuroevolution (CoSyNE) has been introduced (Gomez et al., 2006), and shown to be more effective than many other techniques for both simple and complex problems (Gomez et al., 2008). Since CoSyNE has been shown to be more effective than ESP, this paper will

be a reproduction of Wiering et al. (2005) where, next to other changes, the evolutionary algorithm is swapped out for two variations of CoSyNE.

The previous work introduced the idea of ensuring encirclement by placing 8 sub-goals around the fire in cardinal and inter-cardinal directions, where the distance to the center of the fire is to be determined by the evolutionary algorithm. The agent (for example, a bulldozer) would then travel over the map to each goal, forming a circle around the fire. While the determining of these goals is most influential on the performance of the agents, the behavior asked from the evolutionary algorithm can be considered somewhat simple. To expand on the task, this paper also applies two variations of CoSyNE to attempt to improve on the navigation between the sub-goals. While the previous work computed the shortest path to the next goal, this paper will employ CoSyNE, so that it may make smart decisions about the navigation, where it focuses on the goal of minimizing fire damages. In particular it may look at nearby houses, so that it may try to swerve in order to protect the house, rather than have the house be encased with the fire.

This paper considers two variations of CoSyNE, since it not only tests the possibility of CoSyNE as a way to perform more conscious navigation, but also investigates whether dropconnect (Wan et al., 2013) can give beneficial effects. Dropconnect is a technique that has been proven effective in backpropagating neural networks (Wan et al., 2013), but it has not been applied to networks under control of neuroevolution. Therefore, this research investigates whether similar benefits can be achieved for this evolutionary algorithm.

Unfortunately, researching an accurate simulation of forest fires is beyond the investigative and computational scope of this research. Instead we designed a simple cellular automaton to form a simulation environment for the evolutionary algorithm. Since the simulation is not environmentally realistic, no lessons can be drawn about how to fight forest fires specifically, but it will be possible to draw interesting conclusions about the navigational problem.

Specifically, this research investigates whether an improvement in the number of houses saved can be achieved by applying CoSyNE as a navigation compared to the shortest path, and an improvement on that again when adding dropconnect to the CoSyNE network.

2 Methods

As previously introduced, the experiment will be run on a non-realistic cellular automaton simulation. This keeps the computational cost of running experiments low, as a lot of trials are needed for the algorithms to be able to learn complex behavior. A single simulation has been developed with challenging, but possible fire and agent speed. Over these simulations, the different algorithms were used to fight the fires, and their performance was evaluated on the ability to minimize the number of houses lost to the fire.

2.1 Simulation

To minimize computational cost, the simulation works in discrete time steps. At every time step the cells that are on fire will lose some of their fuel. The cell stops burning when the fuel runs out. As a cell burns through its fuel it gives off heat to the adjacent cells. A cell ignites when the heat given to it exceeds the ignition threshold, which is determined by the material. This cell will then also proceed to burn through its fuel and expel heat towards the adjacent cells. In each time step the agent has a certain amount of energy it can expend. This can be spent on either moving or clearing a tile of fuel.

The maps of the simulation are procedurally generated, where the generation of the evolutionary algorithm determines the outcome of the map. This ensures that the varying algorithms are exposed to the same sets of maps. The maps are fairly simple, consisting of mainly grass, some areas of forest, and a lot of randomly placed houses, as can be seen in figure 2.1. It also shows that the fire is always placed in the center, while the agent is randomly placed around the map. This is done to make the placing of the sub-goals easier, without affecting the performance of the algorithms.

The different materials and how they act in the simulation are a key feature of making the problem of navigation interesting. Movement speed and clearing speed over grass is faster than through trees. This makes using a technique of determining the shortest path a lot better than simple things like moving in a straight line. Naturally, grass also

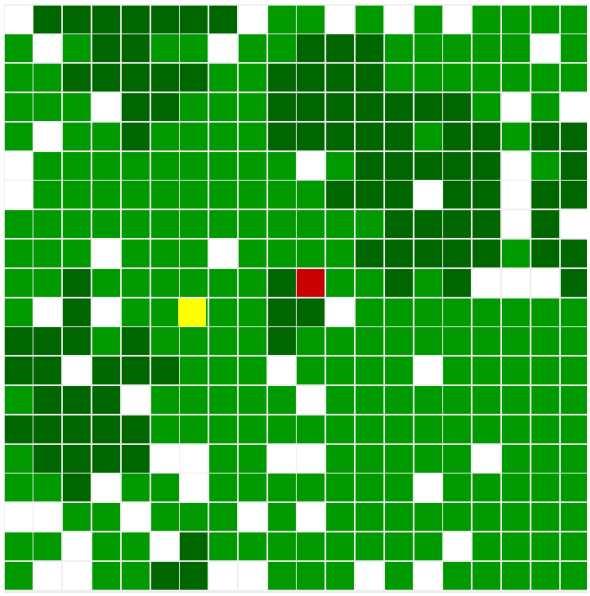

Figure 2.1: An example of the procedurally generated maps. White represents houses, yellow the agent, red is the fire, dark green is forest, and light green is grass.

ignites faster than trees, so fire will spread faster over this area. Grass also has less fuel than trees, so the fire will die down faster as well.

The agent cannot clear tiles which have a house, since the destroying of large brick structures would take an absurd amount of time. To help ensure that the navigation techniques do not try to go over these houses, the movement cost on houses is set to infinite. This ensures that the shortest path will always be around a house, rather than over it.

Another important thing to note about the simulation is that the agent can burn as well. Like the other tiles, when an adjacent tile is on fire and expels enough heat to the agent, it will die. At this point, the agent is not able to do any more work, so it is a given that the rest of the map will burn down if the circle has not been completed.

The map was specifically chosen to be simple, without rivers or roads, so that the maximum influence between the navigation techniques can be attributed to the techniques, rather than to a difference in the maps.

The solution this paper considers consists of two components. The primary component for this research entails the navigation between the subgoals, but for that the underlying component of placing the subgoals is required. While this paper does not concern itself with the optimal placement of these subgoals, a self-learning technique was chosen to support varying behaviour of the navigation techniques. If one of the techniques were to take a slower path, but still protect more houses on the way, that should be beneficial. This is why the subgoals also need to be learned along with the navigation technique.

The navigation will be governed by three different algorithms. The first algorithm will simply identify and follow the shortest path while considering the different movement speeds, this will form a baseline for the other algorithms. This shortest path is determined by Dijkstra's shortest path algorithm (Dijkstra, 1959). The other algorithms work on a variation of CoSyNE.

The algorithms are evaluated on their ability to protect as many houses as possible, as this is likely the easiest component to learn on.

2.2 CoSyNE

Cooperative Synapse NeuroEvolution (Gomez et al., 2006) is a technique which can be applied to a neural network of any architecture to effectively adapt its weights under the supervision of a fitness function. It improves upon standard evolutionary algorithms by not trying to select, breed and mutate networks as a whole, but instead apply this procedure to the individual connections. While other approaches of evolving sub-sections of a network (e.g. ESP (Gomez, 2003)) exist, the evolving of the smallest individual connections has been found to be the most effective (Gomez et al., 2008).

Algorithm 2.1 presents the CoSyNE algorithm in pseudocode as introduced by Gomez et al. (2006). It starts by initializing a population $\mathcal{P}$. This contains subpopulations P_i representing each weight in the neural network. Each subpopulation P_i consists of m values in the interval $[-\alpha, \alpha]$, to be defined for the specific experiment. These $P_{i,j}$ values represent the different possible weights j for each connection i.

After initialization the algorithm repeats over a number of generations. Each generation starts with forming networks X_j, and assessing its performance on the environment through architecture Ψ. After each network has been constructed and evaluated

Algorithm 2.1 CoSyNE (n, m, Ψ)

```
Initialize P = {P_1, ..., P_n}
repeat
  for j = 1 to m do
    X_j ⇐ (P_{1,j}, ..., P_{n,j}) //Form networks
    Evaluate(X_j, Ψ)
  end for
  O ⇐ Recombine(P)
  for i = 1 to n do
    Sort(P_i) //Order weights based on fitness
    for k = 1 to l do
      P_{i,m−k} ⇐ o_{i,k} //Replace least fit weights
    end for
    Permute(P_i)
  end for
until solution is found
```

the breeding process occurs within each subpopulation P_i. Firstly, new offspring $\mathcal{O}$ is created through crossover and mutation from the best performing 25% of the subpopulation. This number of offspring in each generation is again to be defined for the specific experiment. The new offspring replace the least performing weights, which allows the subpopulation to converge to the best values.

Lastly, the weights within each subpopulation are permuted, so that each weight can become part of a potentially different network. This is the coevolution component, and it ensures that all weights within the network converge to their best value, rather than having a non-contributing weight lift along on the good performance in a well performing network, as you might risk in a neuroevolutionary approach that does not use a form of coevolution.

2.3 Applying CoSyNE to the forest fires

The CoSyNE base-algorithm leaves a lot to be defined for the specific experiment. The primary component of this is the neural network architecture. In this case, that would be a Multi-Layer Perceptron (Gardner and Dorling, 1998). Since the subgoal placement is somewhat trivial, and not the focus of this paper, a sufficient implementation is found easily, so no optimization is done.

The network for subgoal placement has one input neuron, which is given the iterator of the subgoal, divided by the total number of subgoals. This ensures values between 0 and 1, and allows the network to somewhat change behavior based on the location of the subgoal. The network has a hidden layer of 3 neurons, and one output neuron. Because all neurons have a sigmoidal activation function, this comes to a value between 0 and 1, where 0 is linked to putting the subgoal in the center of the fire, and 1 is linked to putting the neuron at the edge of the map. The values in between are scaled linearly to the intermittent locations. The subpopulations are initiated in the range $[-3, 3]$, and at each generation 5 new weights are created from the top 25% of weights to replace the 5 worst weights.

The primary interest is the navigation problem. This network has 26 input neurons, a single hidden layer of 8 neurons, and 4 output neurons representing the different steps. A key influencer in this network is a set of four extra input neurons connected directly to the output layer. These four neurons present the directions that would be recommended by the shortest path to the next subgoal. This is an essential component, as it gives the network key information on how to navigate effectively, so that it can slowly learn to improve upon that. These are connected directly, rather than through the hidden layer, because there is no non-linearity to be exerted over this information.

The activation function for the hidden layer is sigmoidal (Han and Moraga, 1995). This is done to prevent the hidden layer from drowning out the extra shortest path neurons. The activation function for the output layer is linear.

The 26 input neurons primarily help the network see nearby houses. 16 of these neurons are responsible for a vision grid identifying houses. The first 8 of these will hold the value 1 if the corresponding cell around the agent is a house, and 0 if it is not a house. The second 8 of these govern 3x3 areas around the immediate connections, where they value at 1 if one of the cells in the area is a house, and 0 if none are. This allows the agent to see houses in the 9x9 cells surrounding it, with increased resolution in the adjacent cells. The vision grid is visually represented in figure 2.2. Another 8 neurons are responsible for localizing the next subgoal and the fire. Four of these are dedicated to the fire, while the other four are dedicated to the next subgoal. The neurons encode for whether the target is north, east, south or west. Since the tar-

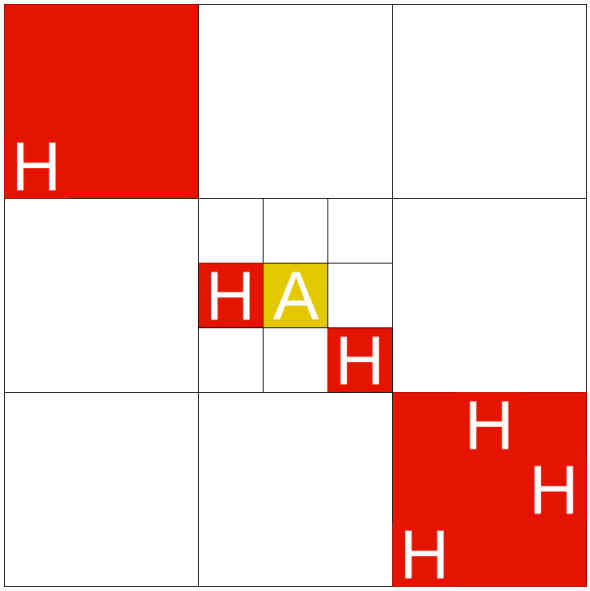

Figure 2.2: A visiual representation of the vision grid, where the yellow cell represents the agent, and the letters "H" represent the houses. All white cells are represented by neurons with the value 0, while the red cells are represetented by neurons with the value 1.

gets are rarely in a perfectly carthagonal direction, usually two neurons will be active at the same time to encode for, by example, having the target be anywhere between perfect north and perfect east. These neurons hold binary 0 or 1 values for whether they are active or not. Another neuron is responsible for identifying if there are any houses between the agent and the next subgoal, having a value of 1 if one or more houses are present, and 0 if none are present. The last input neuron indicates the distance to the next subgoal according to the shortest possible paths. This is computed by taking the average distance for the different valid directions the agent can travel in from the current position, and dividing it by 100. This ensures that the values will be between 0 and 1, so differences do not get diminished by the sigmoid function. The network mainly focuses on houses, as this is the most accessible to translate into key points to protect.

This network consists of a total of 256 connections, which will be represented through 256 subpopulations. Each subpopulation in turn will contain 20 entries, in the range $[-5, 5]$. At each generational step 10 of these will be replaced by children generated from the top 5. For each child, one of the top 5 is randomly selected to be copied, but with a 5% chance of becoming a random new value within the $[-5, 5]$ range.

Some deviations from the CoSyNE algorithm have been introduced however. Firstly, while the CoSyNE algorithm exposes every weight to one network before evolving, this approach exposes every weight to 10 different networks, and assesses its fitness as an average. This decreases stochasticity and should allow for smoother convergence to the final network.

Since the average fitness is tracked over multiple generations, again to decrease stochasticity, calculating the true average can become computationally expensive. Therefore, this is altered to the formula $F_{old} = F_{old} - 0.1(F_{old} - F_{new})$, where F_{new} represents the fitness of the most recent run, and F_{old} represents the previously estimated average. When a weight does not yet have a previously estimated average, F_{old} is considered equal to F_{new}, so that an accurate approximation is reached on the first run. This makes it so that the fitness is tracked over the most recent runs, so that the weight is evaluated to its performance within the current set of networks, but it still holds some virtue if it was able to contribute notably to previous networks.

Another interesting deviation is the addition of SoftMax (Luce, 2012) to the output layer. The output of the neurons are fed through SoftMax, so instead of simply picking the action with the highest activation, an action is chosen randomly, where the neurons with the higher activation have a higher probability of being picked. This allows to create a more exploratory behavior, as well as ensuring that a large difference in activation is beneficial over a small difference. SoftMax is governed by the formula $P_i = \exp(o_i/T)/\sum_{j=0}^{3}\exp(o_j/T)$ (Luce, 2012), where P_x is the probability of action x, o_x is the activation of output neuron x, and T is a parameter which determines the slope of the SoftMax. This can be predefined, but instead this was made to evolve along with the network, as its own subpopulation. This allows the exploratory behavior introduced by SoftMax to be diminished as the network learns to pick the appropriate actions.

A visual summary of the network is presented in figure 2.3.

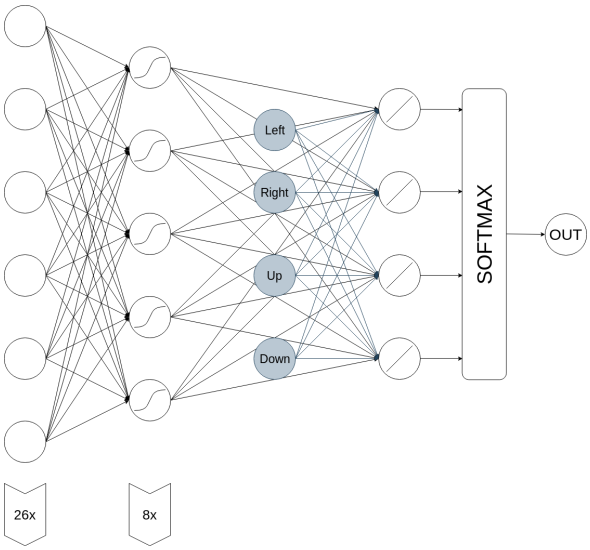

Figure 2.3: A visual summary of the navigation network, consisting of 26 input neurons, 8 hidden neurons and 4 output neurons. The curves in the hidden and output layer represent the activation functions, and the grey neurons represent the shortest path recommendations. The figure also shows the post-processing of the output neurons by SoftMax.

2.4 Dropconnect

As shown by Wan et al. (2013), dropconnect has been found to have beneficial effects on performance in backpropagating neural networks. However, no prominent research exists on how dropconnect affects evolutionary neural networks.

Dropconnect is a technique where each connection has a chance to be "dropped", which means that its weight will be 0, so that it does not contribute anything to the network. This probability can be changed per layer, but in this experiment it is kept to a static 20% over all weights and generations. When a connection is dropped, it renders in the network as a weight of 0, but in the subpopulation the fitness of the weight is simply not updated.

To concretely identify whether this gives any beneficial behavior, the addition of dropconnect is compared to the network as discussed before without dropconnect. Dropconnect supposedly helps to solve the problem where one concept is divided over multiple connections. By occasionally removing one of the connections, the other connections will have to be able to represent the concept on their own.

2.5 Fitness

As the goal of the networks is to maximize fitness, we define our fitness according to the problem we want it to solve. Therefore the fitness is defined as the inverse of the houses burned, so that protecting houses grants a higher fitness. To further decrease stochasticity, this fitness is determined as a ratio of all houses, rather than the absolute number. This can help solve the problem where a map with more houses would always result in higher performances than maps with fewer houses. This works well as a fitness for defining the performance of the subgoal placement. However, for the navigation between the subgoals, the network only gets a reward if the circle is completed, since all the houses burn down if it is not. This leads to a problem, since it can not learn anything until a circle has been completed, which is hard to achieve randomly. Instead, the fitness is slightly altered, where a small component of the number of subgoals reached is also added to the fitness. This has some risky disadvantages as discussed in Sutton and Barto (1998, chapter 3.2). The risk here is that the agent will execute behavior

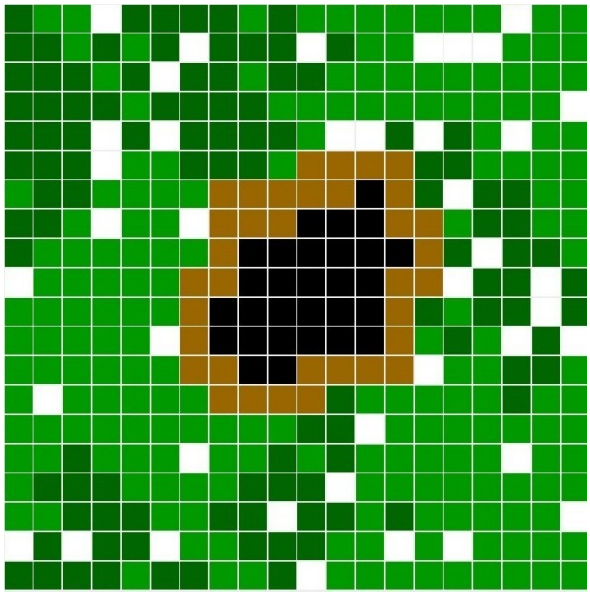

Figure 3.1: A map of the forest fire simulation, where the fire has been successfully encircled and the fire died down.

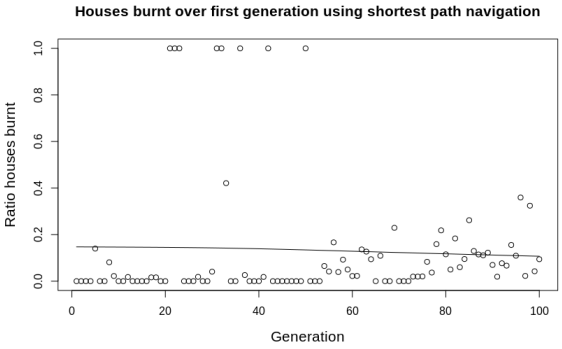

Figure 3.2: The best fitness per generation over the first 100 generations where subgoals are determined by CoSyNE, and navigation is done by following the shortest path.

to reach the reward, rather than solve the problem. This might result in a problem where a house is sacrificed in order to reach the subgoal. However, since completing the circle is essential in being able to protect anything, it is a risk worth taking.

3 Results

Successful solutions of encircling the fire were found easily, as can be seen in figure 3.1. In this image of the simulation, we see that an irregular circle has been drawn around the fire. The irregularities are likely caused by the agent avoiding the forested areas, as they are harder to traverse. This image is taken from the first generation of subgoal allocation with navigation through the shortest path. Therefore, the size of the circle has not yet been defined to be an optimal match.

Figure 3.2 shows that determining a good distance for the subgoals is trivial for CoSyNE. The graph shows that from the start the best fitness per generation is around 18% and this is maintained throughout the generations. There is a slight deflection found at the data points, where between 20 and 50 generations there are a few maps where apparently, performance is very low. As the generations progress this dies out again, while the houses burnt on the good generations go up slightly. This is probably because CoSyNE learned that a small circle, while good sometimes, might be devastating other times, so a larger circled is learned. All in all, this confirms the expectation that picking the distance for the subgoals is trivial, and does not need to be discussed into much further detail.

3.1 CoSyNE Navigation

The performance of the CoSyNE navigation adaptation can be clearly identified in figure 3.3. When looking at the best fitness per generation CoSyNE performs notably worse than the shortest path over the first 600 generations, but better over the 1400 generations after that. Since CoSyNE still needs to learn for this first 600 generations, we consider generations 600-2000, where the mean percentage of houses burned for shortest path navigation is 13.3%, while for CoSyNE it is 10.2%. The best run per generation for CoSyNE is 3.1 percentage point better than the shortest path. This difference is statistically significant according to a Wilcoxon signed rank test (Woolson, 2007) for non-parametric, paired data, with $p < 0.01$.

On the other hand, when looking at the mean fitness for each generation, shortest path navigation performs better than CoSyNE. In these measurements shortest path navigation keeps an average

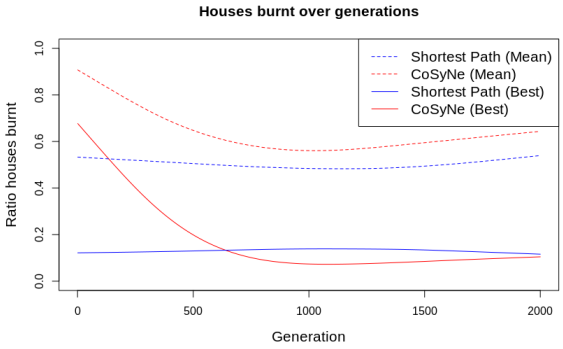

Figure 3.3: The best fitness per generation and the mean fitness per generation over 2000 generations, where subgoals are determined by CoSyNE, and navigation is done by the shortest path or CoSyNE.

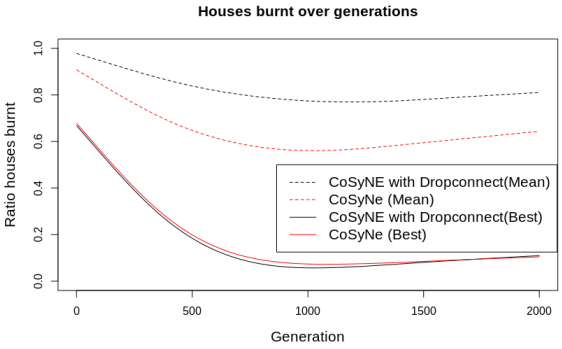

Figure 3.4: The best fitness per generation and the mean fitness per generation over 2000 generations, where subgoals are determined by CoSyNE, and navigation is done by CoSyNe with or without dropconnect.

performance of having 49.4% of houses burnt, while CoSyNE burns 59.5% over generations 600-2000. Again, this difference is clearly highly significant and confirmed by a Wilcoxon signed rank test for non-parametric, paired data, with $p < 0.01$.

3.2 Dropconnect

Lastly, the experiment investigated how dropconnect would affect the performance of CoSyNE. The results of this can be found in figure 3.4. This shows that the performance of the best run per generation is slightly improved overall with dropconnect. Over all generations dropconnect improves performance from 17.5% to 16.5%. After most of the learning, which is again considered from generation 600 onwards, the performance is increased from 10.2% to 9.1%. Both of these difference are considered statistically signiciant by a Wilcoxon signed rank test for non-parametric, paired data with $p < 0.01$.

When investigating the mean performance per generation dropconnect gives a disadvantage. Over all generations CoSyNE without dropconnect loses 63.4% to the fire, but the addition of dropconnect increases this to 81.8%. Clearly this difference is significant, as again confirmed by a Wilcoxon signed rank test for non-parametric, paired data with $p < 0.01$. While the mean performance per generation does improve somewhat over learning, performance clearly remains minimal.

4 Discussion

As expected, allocating subgoals with CoSyNE and navigating between them with the shortest path is a simple and effective way to encase the fire. It takes practically no learning to reach a good performance.

However, there does exist a very large disparity between the best run and the average run in a generation. If this technique would be applied as-is to a real world problem the performance would be considered insufficient, as the average run per generation is likely the most accurate heuristic for the performance of any single fire.

This is likely because the allocation of the subgoals relies on a very limited amount of information. If the allocation of the subgoals would also be based on things like forest density and the spawn location of the agent, performance might be better overall.

Fortunately, when considering the best run per generation, performance is fairly good. While this is not a good heuristic for one-shot problems, like a real forest fire, it can be useful when applied to a realistic simulation. If a realistic simulation representative of an actual forest fire is developed this technique can be effectively applied to the simulation to identify a good solution, which can then be executed on the real fire.

A similar thought process can be applied to the use of the CoSyNE adaptation for navigation.

While the average performance per generation is fairly bad, and even worse than the baseline shortest path navigation, the best performance per generation is notably better.

Like the disparity between best and average performance for the baseline algorithm, this can be acceptable when considering a simulation where there is the option for trial and error. In this light, using the CoSyNE adaptation for navigation is a beneficial addition over shortest path navigation.

With a critical eye there are some insights to be seen as to why there might be better alternatives over CoSyNE. As CoSyNE introduces a certain degree of stochasticity to the movement made by the agent it is likely that one of these random changes is better than the baseline, while many other changes would be worse than the baseline. This is also reflected in the results, where average performance per generation decreases, while best performance per generation increases.

There is some space to explore discrediting this CoSyNE adaptation for navigation by comparing it to a simple shortest path navigation with some noise component added to it. There is a good possibility that this might perform similarly, or even better, than this specific CoSyNE adaptation.

The same problem exists for the addition of dropconnect to the CoSyNE adaptation. While there is again an increased performance in the best run per generation, the addition of dropconnect is detrimental to the average run per generation.

This can partly be attributed to the fact that the neurons which feed shortest path recommendations to the output layer are essential to the performance of the network. When these connections are dropped, the ability to reach a good performance is hindered immensely. Therefore, it is hard to reach a general conclusion about whether dropconnect would have a beneficial effect to regular multi-layer perceptrons under the evolution of CoSyNE.

Fortunately, the performance of the best run per generation is still notably increased overall, so for problems where the ability to explore different solutions before settling on a final decision is available this can still be considered a good addition.

Like the original CoSyNE adaptation without dropconnect, there is still the risk that the benefits on the best run per generation might be attributable to the increased stochasticity that dropconnect adds to the original CoSyNE.

Overall we can confidently conclude that the CoSyNE navigation adaptation, particularly with the addition of dropconnect, is very effective at finding a good solution to save houses over a set of simulations. It notably improves on the performance of simple shortest path navigation. However, it should not be applied as a solution for one-shot problems, since the average performance per generation, for all of these techniques, is insufficient.

References

G Calabri. Recent evolution and prospects for the mediterranean region. In *Forest Fire Prevention and Control*, pages 113–126. Springer, 1982.

Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

Matt W Gardner and SR Dorling. Artificial neural networks (the multilayer perceptron)a review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14-15):2627–2636, 1998.

Faustino Gomez, Jürgen Schmidhuber, and Risto Miikkulainen. Efficient non-linear control through neuroevolution. In *European Conference on Machine Learning*, pages 654–662. Springer, 2006.

Faustino Gomez, Jürgen Schmidhuber, and Risto Miikkulainen. Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, 9(May):937–965, 2008.

Faustino John Gomez. *Robust non-linear control through neuroevolution*. PhD thesis, 2003.

Jun Han and Claudio Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International Workshop on Artificial Neural Networks*, pages 195–201. Springer, 1995.

R Duncan Luce. *Individual choice behavior: A theoretical analysis*. Courier Corporation, 2012.

Narayan Sastry. Forest fires, air pollution, and mortality in southeast asia. *Demography*, 39(1):1–23, 2002.

Brian J Stocks, MA Fosberg, TJ Lynham, L Mearns, BM Wotton, Q Yang, JZ Jin, K Lawrence, GR Hartley, JA Mason, et al. Climate change and forest fire potential in Russian and Canadian boreal forests. *Climatic change*, 38(1):1–13, 1998.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An Introduction*, volume 2. MIT press Cambridge, 1998.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066, 2013.

Marco A Wiering and Marco Dorigo. Learning to control forest fires. In *Umweltinformatik'98: Vernetzte Strukturen in Informatik, Umwelt und Wirtschaft, Proceedings des 12. Internationalen Symposiums 'Informatik den Umweltschutz'*, pages 378–388, 1998.

Marco A Wiering, Fillipo Mignogna, and Bernard Maassen. Evolving neural networks for forest fire control. In *Proceedings of the 14th Belgian-Dutch Conference on Machine Learning*, pages 113–120, 2005.

RF Woolson. Wilcoxon signed-rank test. *Wiley encyclopedia of clinical trials*, pages 1–3, 2007.