university of groningen / faculty of science and engineering / artificial intelligence

FOREST FIRE SPREAD PREDICTION USING PARTICLE SWARM OPTIMIZATION

Ruurd Bijlsma
r.bijlsma.3@student.rug.nl

March 14, 2022

Supervisor:
prof. dr. H.B. Verheij (Artificial Intelligence, University of Groningen)
Dr. M.A. Wiering (Artificial Intelligence, University of Groningen) †
Second supervisor:
M. Sabatelli, MSc (Artificial Intelligence, University of Groningen)

Acknowledgements

I have received great support and assistance while creating this thesis. I would like to thank my supervisors, who helped with the writing process and the technical implementation details. My first supervisor, dr. Marco Wiering, sadly passed away. His expertise was greatly valuable in the first stages of this thesis, when I was formulating research questions and looking into methodology. He was passionate about the subject of using artificial intelligence to combat forest fires, and he is the reason I started looking into this subject in the first place.

I would also like to thank my other supervisor, prof. dr. Bart Verheij. He greatly helped me in the writing process by providing invaluable feedback, and by being a positive and reassuring presence whenever we met.

Lastly, I want to thank my second supervisor, Mattia Sabatelli, and the tutors at the University of Groningen for being responsive and open to my technical questions, even though I did not ask for help often. Knowing support is available already helps.

Abstract

Wildfires can be devastating to forest ecosystems and human populations. If we could predict the spread of these fires, they could be controlled more easily. One way to achieve this is to build a simulation of wildfires that learns from real-world data. This thesis describes the theory behind such simulations, a novel simulation that aims to be accurate and fast, and a qualitative analysis of the novel simulation. We hypothesise that the novel simulation could be computationally faster than existing solutions to this problem, while retaining good accuracy. This novel simulation was successfully implemented using real-world weather, land cover, elevation, and burnt area data, with optimization being done using particle swarm optimization. The simulation achieves an area under the curve (AUC) value of 0.67 on test data when comparing simulated burnt area to real-world burnt area data, which is on par with state of the art solutions in some cases or slightly worse in others. This relatively low accuracy value could be due to a local minimum, lack of data, low quality data, or insufficient configurability of the simulation, causing poor generalisation performance. Besides reinforcement learning, the simulation could be suitable for other applications, such as helping firefighters predict the spread of wildfires.

Contents

1 Introduction **4**
1.1 Purpose of the thesis 5

2 Background **5**
2.1 Fire behaviour 5
2.1.1 Forest fire behaviour 5
2.1.2 Topography 6
2.1.3 Weather effects on fire behaviour 6
2.2 Simulations 7
2.2.1 Existing simulations 7
2.2.2 Evolutionary Algorithms 10
2.2.3 Simulation data 11
2.2.4 Evaluating a fire simulation 11
2.2.5 Temporal scale and resolution 11
2.2.6 Spatial scale and resolution 12
2.2.7 Technical considerations 12
2.3 Requirements 13

3 Methods **14**
3.1 Data 14
3.1.1 Data boundaries 14
3.1.2 Corine Land Cover - Land cover types 15
3.1.3 Globfire 16
3.1.4 ERA5-Land - Weather data 16
3.1.5 ASTGTM - Elevation data 18
3.1.6 Data grid implementation 18

3.2 Simulation . . . 19
3.2.1 Particle swarm optimization . . . 20
3.2.2 Simulation . . . 22
3.3 Optimization . . . 22
3.4 Experimental setup . . . 23
3.5 Technical Implementation . . . 23
3.6 Reproducibility . . . 23

4 Results **24**
4.1 Spread of optimised parameters . . . 24
4.2 Computational performance . . . 24
4.3 Encountered situations . . . 24

5 Discussion **24**
5.1 Computational performance . . . 25
5.2 Accuracy . . . 25
5.2.1 Reasons for our results . . . 26
5.3 Optimised simulation parameters . . . 28
5.4 The optimizer of choice . . . 28

6 Conclusion **29**
6.1 Further development options . . . 29
6.2 Further research . . . 30

A Appendix **32**
A.1 Initial parameters . . . 32
A.2 Land cover types . . . 32

1 Introduction

Forest fires, also known as wildfires, are uncontrolled fires in wild areas. They can cause massive damage to the forest ecosystem and to human populations living near or in the forests. In some cases, fires are part of the forest ecosystem. However, in most cases the damage caused by wildfires is much greater than the benefit they bring. Forest fires have been increasingly causing damage in recent years [14].

Artificial intelligence can be used to help minimise the damage caused by forest fires. By building an accurate and fast simulation of forest fires based on real-world data, further methods can be developed to help prevent the spread of fires. One example is using reinforcement learning [12] to learn optimal digging lines for bulldozers to stop the spread of a fire. Such a method would require a fast and accurate simulation of a forest fire. Another example would be to use a simulation to simply predict where a fire will spread once you know where it has started. This information can be used to evacuate certain areas, to more optimally direct where firefighters should go, or to help decision support systems for the suppression of forest fires [24].

While prevention is an important part when minimising the damage caused by wildfires, this project will focus on predicting wildfire spread after the origin has been

detected. There are existing simulations out there. One of such simulations is HIGRAD/FIRETEC [18]. This is a state of the art physics based fire simulation. This simulation combines multiple models that represent adjustments to terrain, types of fuel, combustion, heat transfer, and turbulence. However, this simulation comes with two downsides.

First, it is currently only being used for research purposes, because of the large amount of computational resources it requires to run. Second, because of its complexity, it is not suited for simulating very large areas.

Other simulations sacrifice a lot of detail for performance increases, such as the elliptical model by Glasa et al. [11], or the cellular automaton model by Zheng et al. [29]. The aim of the simulation described in this thesis is to strike a balance between computational performance and detail. We hope to accomplish this by taking advantage of parallel computing on a GPU for the simulation.

1.1 Purpose of the thesis

This thesis examines a new method for simulating forest fires. This new method will be expanding on existing research by striking a balance between the computational speed and accuracy of the simulation. This will allow the simulation to be used for computational learning methods such as reinforcement learning. We hypothesise that our simulation will be as accurate as existing solutions with better computational speed, due to the use of GPU accelerated computing. If this is achieved, besides its hypothetical use for reinforcement learning, the method could be developed into a user-friendly program to help predict forest fire spread in real life.

2 Background

This simulation will need to reflect the real world fire spread dynamics as accurately as possible. In this section background theory for fires and simulations is explored.

2.1 Fire behaviour

On a basic level, a fire needs three elements: heat, fuel, and oxygen. Each of these elements has their own effect on the spread of a fire. Once a fire starts, it will continue to burn for as long as these three elements are present. Firefighting methods will try to remove one of these elements from the equation to slow or stop a fire. By using water, foam, dirt, or fire retardant the nearby fuels can be cooled below the combustion temperature. These techniques can also help to cut off the oxygen supply to the fire. Another common technique is to clear swaths of trees in the path of a forest fire to remove its access to new fuels. This section explains some external variables that have an effect on forest fire spread, which are used in the simulation built for this thesis.

2.1.1 Forest fire behaviour

There are three basic types of forest fires listed below. The effects of these types of fires will not be explicitly programmed into the simulation mentioned in this paper, but

we hypothesise the simulation can learn about the effects itself by making use of the land cover type, and other types of real-world data.

Crown fires The top of the tree, usually containing the foliage, is called the crown. A crown fire burns the entire tree, including the crown of the tree. This type of fire spreads fastest and more intensely than the other types listed. The dynamics of a crown fire depend on the density of the suspended material, canopy height, canopy continuity, vegetation moisture content, and weather conditions.

Surface fires A surface fire burns litter, duff, and low-lying vegetation. This type of fire burns relatively slowly. Steep slopes and winds can accelerate the rate of spread.

Ground fires Dead vegetation, humus, peat, or other buried organic matter can become dry enough to burn. Such fires are very difficult to put out, but move very slowly. These fires can burn for months.

2.1.2 Topography

Another aspect important for predicting forest fire spread is the topography of the area. Mainly the slope of wherever the fire occurs [4]. Fire moves faster up a hill than down a hill, because the flames can reach more unburnt fuel. The heat from a fire also preheats the vegetation above, drying it out and making it easier to ignite.

2.1.3 Weather effects on fire behaviour

The weather effects this simulation uses are listed below. These effects are built into the simulation using real-world weather data. Specifics for the weather data used in the simulation are discussed in section 2.2.3.

Temperature Forest fuels (vegetation) receive heat by radiation from the sun. Because of this, less heat is required by other sources for ignition to occur. Temperature is one of the most important weather elements affecting fire behaviour. With a higher temperature less heat is required for ignition, which means the spread rate of the fire is higher. Fuels exposed to warmth are also dryer than other fuels, resulting in even more fire spread.

Wind Wind also has a strong effect on fire behaviour due to the fanning effect on the fire. Wind supplies oxygen to fires for the combustion process, it also helps in reducing fuel moisture by increasing evaporation, and it physically moves the flames, sparks, and firebrands towards new fuels. Firebrands are combusting pieces of vegetation that release burning elements. Wind can cause a fire to jump barriers that would normally stop its spread by carrying firebrands and sparks.

Humidity Relative humidity is the ratio of water vapour in the air at the moment of measurement compared to how much water vapour would fully saturate the air at that temperature. With a lower relative humidity, a fire will ignite more easily, and burn more vigorously. With a lower relative humidity, the moisture in fuels will easily evaporate and escape into the air. While with a higher relative humidity, it's more difficult for the moisture of the fuel to escape into the air, therefore slowing down the ignition of the fuel. If the relative humidity is at 100%, the fuel will not dry out due to evaporation.

Precipitation Precipitation affects relative humidity and fuel moisture directly. Usually the temperature also drops during precipitation. There are different ways to get an accurate model for fire spread dynamics. In the following sections two of these methods are described.

2.2 Simulations

In this section, the background theory about simulations relevant to this thesis is explained.

2.2.1 Existing simulations

Many existing solutions exist for simulation forest fires. This section discusses the two main types these simulations fall into, and some more specific solutions.

Process-driven modelling The simulation method in this paper is partly process-driven modelling. In process-driven modelling, the simulation is created by implementing dynamics of fire spread based on current theory. For example, if the temperature at a certain position is above a threshold, and there is fuel and oxygen, then fire spreads to that position. Similar to that rule, there are many others, some general rules, and some very specific to certain conditions, such as crown fires, bush fires, and more. Assuming these rules are accurate, and they are implemented soundly and completely, the simulation should accurately reflect real-world fire spread dynamics.

Forest fire behaviour is a complex result of topography, weather, and fuels. The fuel found in a forest consists of different sizes of dead and alive vegetal matter. For example "duff" is a layer of decomposing organic material lying on the ground. "Litter", which are old leaves lying on the ground. And of course, there are shrubs, grass, and trees, especially the crowns of trees, which also govern the spread of fire in a forest [4].

A disadvantage for this method is that it is difficult to account for every situation there is. There might be a situation where multiple effects combine to create a different outcome. I.e., the combination of a sloped ground with wind in a certain direction could negate the effects that the wind normally has there.

Data-driven modelling As mentioned before, the simulation method in this paper is partly process-driven modelling, the other part is data-driven modelling. By creating a model with parameters tuned based on real-world data, the need to have a great

understanding of the underlying mechanics of the system that is being modelled is reduced. This lowers the risk of mistakes in the model, or edge cases being missed.

In the past, data was scarce and frequently had to be obtained by performing experiments yourself. Today, data is abundantly available, making data-driven modelling more feasible. For example, Google Earth Engine, which allows for programmatic access to a large set of geographical information data. This data has been used for mapping wildfire progression [6], this study uses Bayes theorem to synthesise a time series that shows change and stability in the studied land area. Another study has used Google Earth Engine to assess fire hazards by measuring land use and land cover changes [22].

Something similar to this can be used to create our simulation. By using satellite imagery in combination with machine learning algorithms, we can create a data-driven model for wildfire spread dynamics [10].

A disadvantage to this method could be insufficient training data. When something happens that the model has not occurred in the training data, the model might predict incorrectly. This might be an issue when fire spread control measures are to be implemented, for example digging a path using a bulldozer.

HIGRAD / FIRETEC FIRETEC is a physics-based, 3D wildfire simulation, taking into account the interaction between fire, fuels, atmosphere, and topography on a 1000 metre scale. FIRETEC is used in combination with HIGRAD, a fluid-dynamics model for airflow in combination with fires [18]. FIRETEC takes large computational resources to run, therefore it is not well suited for reinforcement learning.

The simulation is 3D in the sense that smoke and heat can travel in three dimensions, and there can be multiple layers of fuel. For example, the bottom layer could be tall grass, and a layer above that could be tree crowns, as shown in Figure 1. Both fuel layers would impact the spread of the fire, in a strictly 2D implementation multiple layers would not be supported.

While this exact implementation is too computationally intensive for our purposes, the paper does describe formulas for moisture conservation, fuel volume conservation, temperature, oxygen, gas density, and the evolution of gas temperature. These can be helpful when implementing a process-driven simulation, or a mixed process/data-driven simulation like the one described in this paper. We can also quantitatively compare the results of our simulation with this one.

Cellular automaton Wiering [28] uses a stochastic cellular automaton to create a wildfire simulation, named Bushfire in his paper on evolving neural networks. Single cells may contain different kinds of trees, grass, water, and digged paths, as well as information about whether that cell is on fire or not. The fire spreads according to wind strength, direction, humidity, and cell type. Cells ignite when a certain fire activity threshold is met, which is defined per cell type. After a cell has burned for a certain time the fuel runs out and the cell stops burning.

This simulation also supports bulldozers digging paths to try to control the spread of the fire. The bulldozers have a parameter stating how fast they can move.

A cellular automaton simulation can be computed extremely quickly, and is therefore practical in reinforcement learning situations. Since the evolution of each cell is

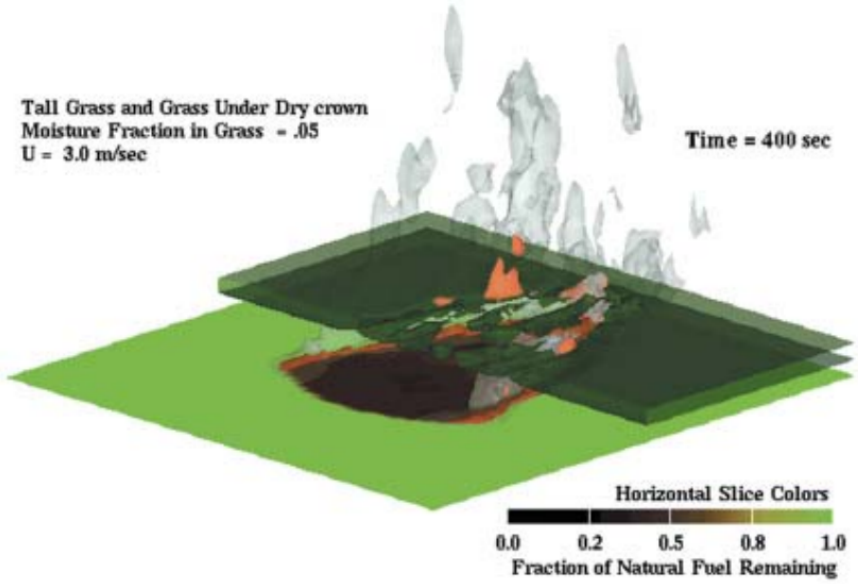


Figure 1: An oblique view of a crown fire, showing three dimensional smoke and ground layers.

only dependent on the state of the cells in the previous time step, the computation for each cell can be done in parallel. This performance is something our simulation also aims to achieve, therefore a cellular automaton similar to the one described above is a good starting point for our simulation.

Genetic algorithm A genetic algorithm can be used to automatically tune a highly configurable model for forest fire spread. By creating a fitness function based on real-world data, the model can be tuned to become better and better. Denham et al. [8] use a genetic algorithm to simulate forest fire spread and get positive results. Such an algorithm can be used to optimise the parameters used in our simulation by comparing it to real-world data.

Neural Networks Zheng, et al. [29] use an extreme learning machine to optimise a cellular automaton based simulation. An extreme learning machine is a type of feed-forward neural network. In this cellular automaton, each cell has a set of driving factors for what the next state of the cell will be, these are fed into the extreme learning machine to get the next state.

Spatial reinforcement learning Ganapathi, et al. [10] use spatial reinforcement learning to build forest wildfire dynamics models from satellite images. Wild fire dynamics can be characterised as a spatially spreading process (SSP), which requires many parameters to be set precisely to properly model the dynamics. They introduce a novel approach for learning SSP domains using reinforcement learning where the fire is the agent at any cell in the landscape. The set of actions the fire can take includes

spreading in any of the cardinal directions, or not spreading. Rewards are provided when the fire spreads correctly according to satellite data.

This paper compares five RL algorithms for this problem, value iteration, policy iteration, Q-learning, Monte Carlo Tree Search (MCTS), and A3C. It is found that A3C has the best performance for predicting spread dynamics at intermediate time steps, whereas MCTS performs better while predicting future spread.

2.2.2 Evolutionary Algorithms

Section 2.2.1 shows an example of a genetic algorithm being used as a possible optimisation method for simulations. A genetic algorithm is a type of evolutionary algorithm, evolutionary algorithms are a form of optimization algorithm using biological evolution mechanisms, such as reproduction, mutation, recombination, and selection. The simulation used in this paper will also use an evolutionary algorithm for optimization, and this section explores some of these algorithms to show which would be a good fit for our problem.

Genetic algorithm Genetic algorithms are a common evolutionary algorithm [20]. A population of individuals, each representing a possible solution, is created. Each candidate has some properties which can be altered or mutated during the evolution process. The population is usually initialised randomly, the fitness for each individual is calculated, and the most fit individuals are selected to base new individuals off of. A few random low fitness individuals are also selected to make it through to increase diversity of the population. The algorithm terminates after a set number of generations, or after a desired fitness is reached. This algorithm can be used to optimise the fire simulation by encoding the parameters of the simulation as solutions, such as a string of 0s and 1s that can be mutated.

Gaussian adaptation In Gaussian adaptation [17], a multivariate Gaussian distribution is used to produce a number of n-dimensional vectors. These samples are tested to pass or fail, the probability of a pass is determined by the fitness function, which produces a value between 0 and 1. The set of samples that pass are used to produce a new Gaussian distribution, completing one generation. This algorithm can be used to optimise the fire simulation by simply using each parameter as a value of the n-dimensional vector, and creating a fitness function that produces value from 0 to 1.

Particle swarm optimization In particle swarm optimization (PSO) [16], a swarm of particles are placed evenly on a multidimensional search space where some optimal solution needs to be found. The particles each calculate the value of their position using the fitness function, and report the value to all other particles. Every particle is guided by the knowledge of their own best-known position in the search space, as well as the entire swarms position. As better solutions are being discovered, the swarm as a whole can move toward them and hypothetically reach a global maximum. This algorithm can be used to optimise the fire simulation by using each parameter as a dimension in the search space, and a fitness function could be used to determine the solution quality.

2.2.3 Simulation data

The data that we will implement for our simulation is as follows:

Land cover type This would indicate what the cell type is. It could be infrastructure, a body of water, a specific type of forest, or another type of vegetation.

Elevation map The elevation of each cell. Knowing the elevation is important for modelling fire spread dynamics, fire is more likely to spread uphill than downhill [4].

Weather data Weather is an important factor for modelling the spread of wildfires. Wind, temperature, and rain are some of the most important meteorological events that affect the spread. Getting accurate data is paramount for creating an accurate simulation.

While training and testing the simulation will be able to use past weather information, however, if the simulation is to be used for predicting the spread of a current forest fire, it will also require future weather data.

For future weather data, we either simulate the weather ourselves or use existing forecasts for the given region and period. Simulating the weather ourselves is possible using existing simulators, however weather forecasting data is abundantly available, accurate, and has nearly no computational costs, therefore this is a better option.

2.2.4 Evaluating a fire simulation

Since the accuracy of the simulation is crucial in creating a good method for forest fire control, we need to be able to evaluate the simulation based on empirical data. [29] propose an objective evaluation for forest fire models by comparing predicted spread to satellite imagery of real-world fires.

Their method uses the area under the receiver operating characteristic curve (AUC) and seven other metrics to determine the performance of a simulation. The AUC was calculated plotting the true positive rate against the false positive rate, and measuring the area under that curve.

2.2.5 Temporal scale and resolution

For the reinforcement learning method we will need a simulation to model the wildfire spread dynamics. Before 1986, a wildfire was contained on average in less than eight days. In 2003, a wildfire burned for 37 days on average [27].

Due to the need for relatively long time frame support, changing weather should be supported in the simulation. Because of the complexity of weather models, it is best to use an existing weather simulation, or to use existing weather predictions for the simulated time period. In section 2.2.3 we explore some possibilities for integrating changing weather over time into the simulation. As for resolution, to accurately simulate firefighting efforts, one simulated tick should be at most 30 minutes to allow the simulation to react to state changes properly.

2.2.6 Spatial scale and resolution

The median wildfire in 2001-2019 burned an area of $210,000m^2(\pm 1,400,000m^2)$ [7], this gives an idea as to the scale that should be supported in our simulation. The resolution should preferably be as high as possible, however this is not feasible when considering limited computing power. Other wildfire simulators seem to use a resolution of about $1m^2$ per cell, as discussed in 2.2.1. Using this resolution, grid dimensions of 840 by 840 should be possible to support the average wildfire mentioned above.

The simulation can be two- or three-dimensional. A two-dimensional simulation would be less complex to implement and it would be much faster to simulate one tick. A three-dimensional simulation could introduce support for some extra simulated features, such as heat rising up or different wind speeds depending on height. More on these differences are discussed in 2.2.1.

2.2.7 Technical considerations

This section will give information about some of the technical choices made for the implementation of our simulation.

Performance requirements As mentioned before, one of the goals for this simulation is to be fast enough for reinforcement learning applications to use. A typical reinforcement learning setup lets an agent perform one of a few actions in an environment, then lets the environment react to the state change, and after a time step the new state of the environment is used to calculate how good the action by the agent was. In a forest fire scenario using a temporal scale of 30 minutes, the average wildfire of 37 days takes 1776 simulated ticks for just 1 epoch on a single type of environment. For learning to occur many epochs have to be performed on many fires, and for it to generalise well, it must train on many types of environment. Taking this into account it is clear that some types of simulation will not be feasible to use in a reinforcement learning scenario.

To give an idea of the magnitude of the simulated ticks required, consider this hypothetical fire-fighting reinforcement learning scenario. Assuming each epoch takes on average 1776 ticks, 50 environment types, and 1000 epochs per environment, there would be $1776 * 50 * 1000 = 88800000$ simulated ticks required to train the agent. Computationally heavy simulations are impractical to use in such scenarios.

Parallel computing The simulation in this paper is implemented as a cellular automaton. This means that the state of each cell is not dependent on the state of other cells during the same time step, only the state of the neighbouring cells from the previous time step, as mentioned in section 3.2.2. This allows for the state of each cell to be computed concurrently.

A graphical processing unit (GPU) was originally built for video processing on computers, calculating the next colour for each pixel in a display. Because of this, GPUs can calculate many tiny computations concurrently, which fits well with our simulation needs.

CUDA The GPU portion of the simulation in this paper is implemented using Nvidia's Compute Unified Device Architecture (CUDA) [5] platform. This architecture allows software to use certain types of GPU for general purpose programming. CUDA gives access to the virtual instruction set and parallel computational elements of the GPU.

CUDA works with existing programming languages such as C [23], C++ [25], and Fortran [3]. For our implementation, C++ is used for its modern feature set and the ability to interface with Python [26] through the Boost C++ library.

In CUDA, one can create compute kernels to enable general purpose programming. These kernels are the part of the code that runs on the GPU hardware.

2.3 Requirements

The goal for the simulation built for this thesis can be described by the following requirements based on the theory presented in this section.

1. The base simulation should be based on real-life fire spread dynamics, which are mentioned in section 2.1.
2. The simulation should take into account the real-world data mentioned in section 2.2.3.
    - Elevation
    - Weather (wind/humidity/temperature)
    - Land cover type
3. The simulation should be initialised using real-world wildfire data, as mentioned in section 2.2.4.
4. The simulation should run on a grid where the size is determined by the affected area and the spatial resolution, mentioned in section 2.2.6.
5. A fitness function should be included which can determine how close the simulated fire is to the real-world wildfire data, as mentioned in section 2.2.1.
6. The simulation should use cellular automatons as a technical framework to support parallelization on the GPU, mentioned in section 2.2.1.
7. The simulation should accept a number of tunable parameters, as mentioned in section 2.2.1, which can influence the spread dynamics in different ways:
    - Rate of burning
    - Height effect, up and down
    - Ignition threshold
    - Spread speed
    - Fire death rate
    - Fire death threshold
    - Spread rate per land cover type

- Wind effect

8. A type of evolutionary algorithm should be used to optimise the parameters given above using the fitness function. Options are mentioned in section 2.2.1.

9. The time frame that the simulation should be able to predict has to be around 20-50 days, which is the duration of real-life fires, mentioned in section 2.2.5.

10. One simulated tick should cover at most 30 minutes to allow the simulation to accurately reflect data changes, such as the weather, as mentioned in section 2.2.5.

11. Support simulation scale of at least 1000 by 1000 metres to allow simulation of real-life sized wildfires and above, as mentioned in section 2.2.6

3 Methods

The simulation created for this thesis uses a mix of process- and data-driven simulation design. First, a cellular automaton simulation is created using some of the formulas from the referenced papers in the background section. A cellular automaton is a grid of cells, each having a certain state. The next state of each cell is determined by a set of rules and the state of its direct neighbours. By repeatedly applying these rules, patterns can occur, or in our case, a configurable forest fire simulation is made.

Many configurable parameters are built into this simulation, such as spread speed, how strong the simulation should respond to wind, or how it should respond to different types of land cover. Real-world data is used to impact the simulation, such as height data, land cover type, and weather data. These configurable parameters are configured using particle swarm optimization. The simulation is given an initial burning area and will simulate for a given number of hours or days. The resulting burnt area from the simulation is compared to the real-world burnt area after the given duration, using this, a performance metric is calculated.

3.1 Data

This section describes the real-world data used by the simulation as well as how the data is parsed. There is a distinction to be made about the fire data and the other data, the fire data is used to set up the initial state of the simulation grid, and to check the fitness of the eventual simulated grid. The other data (weather, elevation, land cover) are used during the simulation and have an effect on the spread dynamics. The data sources used are summarily described in Table 1.

3.1.1 Data boundaries

Data comes in many forms, the data used for this simulation all came from different sources, with different resolutions and boundaries. The weather and burnt area data require a time component, and each have different start and end dates for data availability. Also the temporal resolutions are different. Furthermore, the ground type and

Variable	Source	Scale	Data Range	Date
Elevation	ASTGTM	1 x 1 arc second	South Europe	2000-2013
Land cover	Corine Land Cover	25 x 25 ha	Europe	2017-2018
Temperature	ERA5-Land	0.1° x 0.1°	Europe	1950-2022
Wind	ERA5-Land	0.1° x 0.1°	Europe	1950-2022
Moisture content	ERA5-Land	0.1° x 0.1°	Europe	1950-2022
Precipitation	ERA5-Land	0.1° x 0.1°	Europe	1950-2022
Burnt area	GlobFire	1 x 1 km	Global	2001-2020

Table 1: Explanatory variables used in this study. ASTGTM: ASTER Global Digital Elevation Model, ERA5: European Centre for Medium-Range Weather Forecasts Reanalysis

elevation data have different spatial resolutions to the other types of data, and the elevation data and burnt area data is only available from a limited region.

All data is standardised by parsing it into a grid using a set width, height, and duration, which are calculated based on the set spatial- and temporal resolution and the area and duration of whichever fire is being simulated. This requires interpolation in cases where the spatial resolution of the grid is higher than the source data, or data compression when the source data has a higher spatial resolution. Also, a maximum safe boundary must be calculated to know where simulations can occur based on the data availability. For example, the provider of height data only supplies a few million km^2 per request, so only a relatively small part of the world can be used at once as a source for wildfires for training. A similar issue exists with the burnt area data provider, however, a much larger area is allowed per download request, an example of the scale of these download requests can be seen in Figure 2. The maximum safe boundary calculation takes the largest polygon for which all data sources can provide information.

A similar solution is needed for data sources with temporal variables, such as weather and burnt area. These sources' first and last data records are from different dates, so a maximum safe date range is calculated, and only fires from within this range are used to train the PSO.

3.1.2 Corine Land Cover - Land cover types

The Corine Land Cover 2018 dataset [15], provided by the Copernicus Land Monitoring Service, is used to get information about ground types across the world. Only Europe is covered in this dataset, the spatial resolution is 500 metres. 47 separate types of land cover are specified in the dataset, ranging from 'Continuous urban fabric', 'Rice fields', and 'Sclerophyllous vegetation' to 'Coniferous forest', 'Glaciers and perpetual snow', and 'Peat bogs'. A visualisation of this data is shown in Figure 3. This figure shows the land cover types of Cyprus, in this visualisation the large green area in the north east is coniferous forest, the red area in the centre is a continuous urban fabric, i.e. buildings and infrastructure. Such information can be used by the PSO to adjust the spread rate depending on the land cover type.

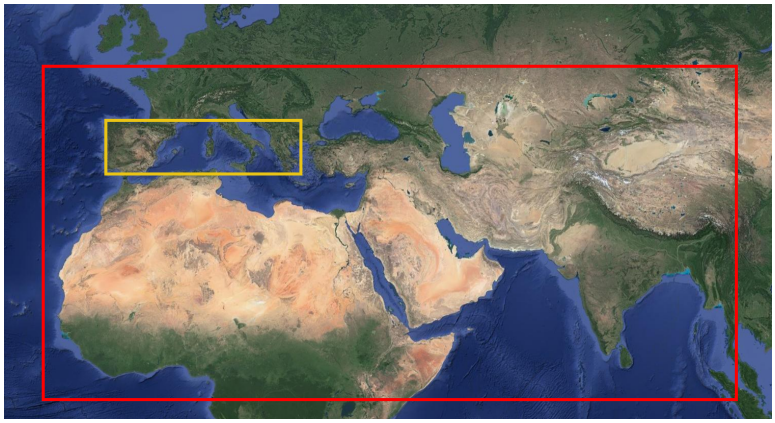


Figure 2: Data boundaries for available elevation data (orange) and weather data (red). The orange area is used for the train and test fires.

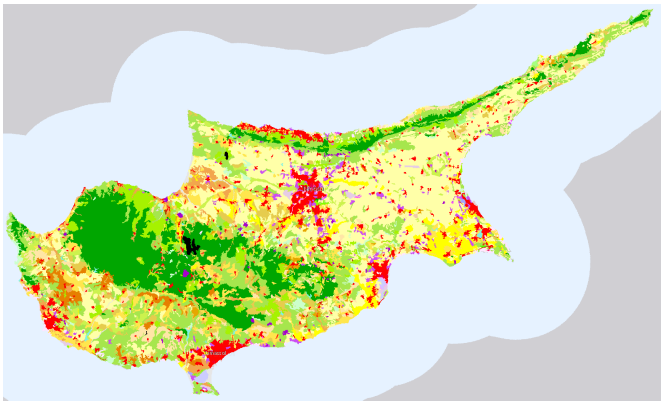


Figure 3: Example visualisation of land cover data for Cyprus.

All types of land cover distinguished by the Corine Land Cover data listed in section A.2.

3.1.3 Globfire

The Globfire database [2] combines wildfire data from multiple sources. Data is stored as a list of fire events, fire events contain multiple steps each representing one day of the given fire event. Each of these steps includes an indicator describing the stage of the fire (active or final), the perimeter, and the date. This dataset covers the entire world, starting at January 1 2001, ranging until December 2020, with more data being added as it becomes available. Figure 4 shows example data from Globfire. In this example four fire events are shown during 14 days, each image representing one day. The final perimeters of the fires are shown in every image that the fire is present in, the filled-in area is the area burned by the fire at that given day.

3.1.4 ERA5-Land - Weather data

The ERA5-Land dataset [21] gives weather data covering the entire world with a 0.1 degree spatial resolution and hourly temporal resolution. This dataset contains wind

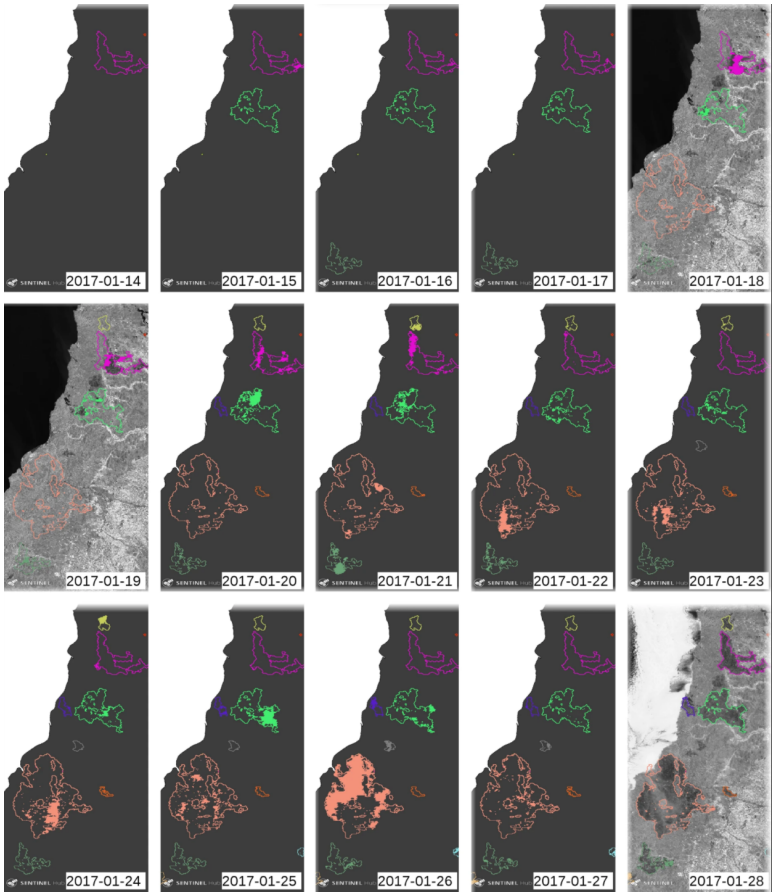


Figure 4: Example of Globfire data. Active fire perimeters shown at different dates [2]. Filled area in colour represents the burned area. Each colour represents a different fire event.

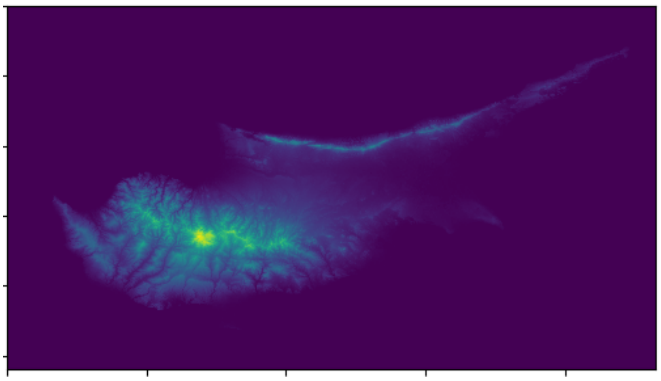


Figure 5: Example elevation data for Cyprus. Purple is lower, yellow is higher.

direction and speed, relative humidity, temperature, precipitation, and many more weather data points. The spatial resolution of 0.1 degrees is not very precise, this isn't an issue considering the weather does not change much across a few kilometres (0.1 degrees is about 8 kilometres).

3.1.5 ASTGTM - Elevation data

The Terra Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER) Global Digital Elevation Model (GDEM) dataset (ASTGTM) [1] provides global coverage for elevation data with a spatial resolution of 0.0003 degrees. Figure 5 shows an example visualisation of this data. This information allows the simulation to increase the rate of spread uphill, and decrease it downhill.

3.1.6 Data grid implementation

A single run of the particle swarm optimization works as follows: first a fire is picked from the burned area dataset which occurs within the region and time frame for which other data is available. The maximum boundary this fire occupies throughout its life is calculated and turned into a square by extending either the height or width to match the other. The simulation will only use the area within those bounds. The data used during the simulation (wind, elevation, etc.) is then cropped from their respective data sources using the square bounds.

The dimensions of the grid are then calculated using the real-world dimensions of the affected area and a set spatial resolution. The spatial resolution used for gathering results for this thesis is 0.001 degrees (world coordinates), which equates to about 100 by 100 metres per cell depending on the latitude of the affected area (longitudinal degrees are 110,000 metres at the equator, 40,000 metres on the Arctic circle). Similarly, the size of the temporal dimension is calculated using a set temporal resolution and the duration of the simulated fire. The chosen temporal resolution used for gathering the results is 1 hour per timestep. Therefore if we take, for example, a fire affecting an area of 0.5 by 0.5 degrees, which lasts 8 days, the simulation grid will be $0.5/0.001 = 500$ wide and high, and have $(8 \text{ days} * 24)/1 = 192$ time steps.

All datasets (fire, weather, elevation, etc.) are then parsed into a grid using the dimensions calculated above. It is very uncommon for the spatial or temporal resolu-

tion of any dataset to match the resolutions of the grid, therefore the data needs to be reshaped using interpolation to fit the grid. The fire grid, other data grids, and values for the tunable parameters determined by the PSO are sent to the GPU to simulate for the given number of time steps (ticks). After these ticks, the fitness value is calculated and sent back to the particle swarm optimization.

3.2 Simulation

This section describes the simulation built for this thesis, the data used in the simulation, and the experimental setup used for testing. A summary of the entire process is as follows:

1. In Python, get a list of all fires within the available data regions, more on these regions in section 3.1.1.
2. Use 5-fold cross-validation to split the dataset into train and test sets, each set containing a list of fires.
3. For each fire in the train set:
   (a) Load the relevant data (land cover, elevation, weather, burnt area).
   (b) For the first fire, initialise parameter values from file, on subsequent fires use values from previous run.
   (c) Send all PSO generated possible solutions (sets of parameters), initial fire data, and simulation data to the GPU.
   (d) The GPU simulates each possible solution using the given data and parameters, and returns the resulting simulated burnt area to Python.
   (e) Calculate fitness by comparing simulated burnt area to actual burnt area.
   (f) Generate new possible solutions based on fitness results and iterate again from step 2b, until the iteration limit is reached.
4. Test the final parameters on the test set, for each fire in the test set:
   (a) Load the relevant data (land cover, elevation, weather, burnt area).
   (b) Send final parameters, fire data, and simulation data to the GPU.
   (c) GPU simulates using the given parameters using the given data and returns the resulting simulated burnt area to Python.
   (d) Calculate fitness.
5. The final performance metric is the average of all fitness values calculated from the test set.

3.2.1 Particle swarm optimization

The evolutionary algorithm chosen to optimise the parameters for this simulation is particle swarm optimization. This particular algorithm was a good fit for our problem because it makes few assumptions about the problem space and it can search large spaces relatively quickly for candidate solutions. Many other optimization algorithms may require a gradient to be known for candidate solutions, this is not possible for our problem since it is not differentiable.

The particle swarm optimization process tries to find the global best fitness value. The optimization algorithm takes a set of configurations for the simulation, known as the candidate solutions, and tries to find the best configuration using a position-velocity update method. Every particle in the swarm represents a solution, all of which are attracted to the best performing particle. The position of each particle in the solution space is updated as follows:

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

Therefore, the position in the next step is equal to the sum of the current position and the computed velocity in the next step. This velocity is calculated as follows:

$$v_{ij}(t+1) = w * v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_j(t) - x_{ij}(t)]$$

Where c_1 is the cognitive coefficient, which models the tendency of a particle to return to its own best found solution. c_2 represents the social coefficient, which determines how much each particle is attracted to the globally found best solution. w controls the inertia of the swarm [9].

As for the other variables, i is the particle, j is the dimension, y_{ij} is the best known position for that particle, $\hat{y}_j$ is the position of the best overall particle, and r_{1j} and r_{2j} are random numbers picked from a uniform distribution $U(0,1)$.

The implementation of the global best particle swarm optimization is achieved using the PySwarms [19] Python package.

Tunable parameters As stated before, the simulation uses a number of tunable parameters. Each parameter changes some part of the behaviour of the simulation, for example, the wind effect parameter is multiplied with the effect of the wind in the simulation. Because of this the PSO should be able to find an optimal value for this parameter, and we do not have to make assumptions about how this should be implemented in the simulation.

The goal of the PSO is to find a set of parameters that makes the simulation reflect as many real-world fires as possible. This section will explain what parameters are used. The parameters are chosen to allow the simulation to reflect many types of fires. Many parameters exist to allow the PSO to influence how the simulation reacts to the real-world data described in section 3.1. The optimizer searches for optimal values for these parameters within the range $0-5$.

- Burn rate: determines how fast fuel is consumed by an active fire.

- Ignition threshold: when the mean neighbour activity exceeds this value, a fire ignites in a given cell.
- Spread speed: multiplier to the neighbourhood activity calculation. Effectively increases spread speed, as the name suggests.
- Fire death threshold: When the fire activity of a given cell is below this value, the fire activity will start to decrease.
- Fire death rate: rate at which fire activity in a cell decreases when the fire death threshold is reached.
- Height effect multiplier: consists of two separate tunable values for height effect up and down. As stated in the background section, fire spreads faster upwards than it does downwards. This effect is multiplied by the height effect multiplier. This parameter can allow the PSO to adjust how the wildfire should react to elevation changes from the real-world data.
- Wind effect multiplier: determines wind effect strength. This allows the PSO to adjust to the wind data.
- Spread rate per land cover type: this represents 47 parameters, each indicating the spread rate for a type of land cover. For example, the spread rate on water could be 0, the spread rate for a coniferous forest 1, and the spread rate of fire on an inland marsh might be 0.3. These parameters allow the PSO to take advantage of the land cover data used in the simulation.

These parameters are learned by the optimizer, at the first training step the parameters are loaded from a file. These initial parameters are given in Table 4 in the Appendix, the initial values are set by intuition, for example: a spread rate of 0 on water, and 1 in a forest. The burn rate is initialised to 0.3, the height effect multiplier both up and down is initialised to 0.1, the wind effect multiplier is 2, the activity threshold is 0.1, the spread speed is 2, the death rate is 0.2, and the fire death threshold is 0.2. These values were set to numbers that intuitively make sense, they are completely changed by the PSO. They are not set to all zero, because that would slow down the learning rate, as it would initially not spread at all. The optimizer would slowly change this, but the learning speed of the PSO would be affected negatively.

Fitness function The fitness function guides the particle swarm optimization, which tries to minimise this function. The fitness is calculated by comparing the simulated final burnt area, and the real-world final burnt area. A receiver operating characteristic (ROC) curve is created showing the true positives (cells where the burnt area matches up with the real-world data) on one axis, and the false positives (cells showing burnt area, which is not burned on the real-world data) on the other axis. $1 - AUC$, where AUC is the area under the ROC curve, is used as the value returned by the fitness function, since the particle swarm optimization aims to minimise the fitness function.

3.2.2 Simulation

The base simulation is the part of the simulation that occurs on the GPU, this is a cellular automaton based on earlier fire simulation cellular automatons. This section describes what happens to a single cell during one simulation tick.

First, the fuel of the cell is updated using the fire activity in that given cell. The fire activity value represents the intensity of the fire in that cell. The fuel of a cell always decreases, the size of the change is proportional to the fire activity. After this the 8 neighbouring cells are considered. The activity for each neighbouring cell is taken and multiplied by a number of factors. Height difference can increase or decrease the activity of a neighbouring cell in relation to the current cell, similarly wind direction and speed are considered. Thus, for example, if the cell to the right has a fire activity value of 0.5, but is higher than the current cell, the activity value will be adjusted to a lower value. If the wind comes from the right, then the adjusted activation is increased a bit.

The adjusted activity matrix is calculated as follows:

$$A = l * \text{activity}_{neighbours} \circ (1 + (\text{height}_{cell} - \text{height}_{neighbours}) * m_{height}) \circ (1 + W * m_{wind})$$

A	Adjusted activity matrix (3x3 grid of neighbours).
$\text{activity}_{neighbours}$	Fire activity matrix of each neighbour.
height_{cell}	Elevation value for the current cell.
$\text{height}_{neighbours}$	Elevation matrix for the neighbouring cells.
l	Land cover spread rate multiplier for cell.
m	Multiplier for height and wind, these values are determined by the PSO.
W	Wind matrix, retrieved from real-world wind data.

Table 2: Variables used in the adjusted activity matrix formula.

The mean adjusted activity values for all neighbours are calculated as shown above and multiplied by the spread rate multiplier for the type of land cover on the cell. If the resulting value is higher than the ignition threshold, the cell will ignite. This is done by setting the activity of the cell proportional to the available fuel and the mean adjusted activity of the neighbours.

3.3 Optimization

Each fire in the train set is given sequentially to the PSO, which then runs for a set number of iterations trying to find the optimal parameters for the given fire. In each iteration, a number (swarm size) of parameter sets are created, which are each used to simulate the fire. For each of these simulations, an AUC score is calculated to indicate how accurately the simulated burnt area reflects the real burnt area of the given fire.

After all iterations are completed for a fire, the previous parameter list is updated by adding the newly optimised parameters multiplied with a learning rate: $P(t+1) = P(t) + \Delta P * \eta$ where P is a vector of the parameters, η is the learning rate, and ΔP is the optimised parameters for the last fire.

3.4 Experimental setup

To test the generalizability of the PSO, k-fold cross-validation is used. For this, the data needs to be split into k parts. Each part contains $1/k$ part of the total amount of fires, $k-1$ of those parts will be used for training, then the remaining part will be used for testing. This is repeated k times, each time using a different part for testing. The mean fitness score is used as the testing performance measure.

For this experiment $k = 5$ is used for the cross-validation. This value was chosen because a higher value leads to longer training times, which is not desirable. Using $k = 5$ takes around 20 hours to complete the entire 5-fold cross-validation. The dataset contains 315 separate fire events, each event containing a number of time steps, each having a polygon for the burnt area at that time step. This dataset is large enough for 5-fold cross-validation to be reliable, as a single fold still has enough data.

Seven experiments are performed using 5-fold cross-validation, each using different hyperparameters, to find the effect these hyperparameters have on the outcome.

3.5 Technical Implementation

The majority of the logic for the simulation is implemented in Python [26] using the NumPy library [13] to speed up array programming.

As mentioned before, the base simulation, that is, the cellular automaton, is implemented on the GPU to speed up computations. When programming compute kernels in CUDA, a number of threads must be chosen. To find the optimal number of threads, a small experiment was performed running the simulation for 50 ticks and measuring the time taken for a given thread count. Each measurement is repeated 10 times to get a more stable average duration for the run. The range of thread counts tested starts at 12 and ends at 512, with a step of 12. The measurements are taken using the high resolution clock included in the C++ standard library.

The simulation also has a CPU implementation in C++, the performance results for this implementation compared to the GPU implementation are also measured. This experiment tests the mean time the CPU and the GPU take to run the simulation for a given grid size. The grid size is varied from 1 by 1 to 140 by 140.

An overview of the different components used in the simulation can be seen in Figure 6. This figure is meant to give an idea of the architecture of the training and testing, and of the different platforms used in the implementation.

3.6 Reproducibility

To ensure the results of this paper are reproducible, all random generation is fixed by using a seed. The code for this project is stored on GitHub[1]. To compile and run the C++ part of the project, the Nvidia CUDA Toolkit is required. The C++ Boost library must also be compiled with support for Python bindings. The Python part of the project requires a set of libraries, which are listed in the GitHub repository. The experimental results used to create the plots shown in the results section are also available for download on the same repository.

[1] https://github.com/RuurdBijlsma/WildfireSimulator

4 Results

This section shows the results from testing our simulation using K-fold cross validation. Table 3 show the mean AUC for different sets of hyperparameters using the test set. The training performance for a random set of 30 fires is 0.772 ($\sigma = 0.117$).

Experiment	Swarm size	Iterations	Learning rate	c1	c2	w	Mean AUC
1	10	10	.2	0.5	0.3	0.9	0.6256
2	30	30	.1	0.5	0.3	0.9	0.6643
3	30	30	.2	0.5	0.5	0.9	0.6683
4	20	20	.2	0.2	0.5	0.9	0.6625
5	20	20	.2	0.5	0.5	2.0	0.6502
6	20	20	.2	0.5	.5	0.5	0.6542
7	50	30	.2	0.5	0.5	0.9	0.6617

Table 3: 5-fold cross validation results for different experiments, tested and trained on 200 fires from southern Europe. The green row represents the experiment with the best performance, orange cells are changes from the optimal configuration.

4.1 Spread of optimised parameters

The variation in parameters after optimization is shown in Figure 7 and Figure 8. Each dot represents a value after optimization is done for a given fold in the k-fold cross validation process.

4.2 Computational performance

The performance results shown in Figure 10 and Figure 9 were gathered on a personal computer (Nvidia RTX 2080ti, AMD Ryzen 2600, 16 GB ram). Figure 9 shows the optimal thread count for our CUDA application, which is 156 threads.

4.3 Encountered situations

Figure 11, Figure 12, Figure 13, and Figure 14 show a number of burnt area comparisons are shown between the simulation and the real-world data, including the ROC curve for that given comparison.

5 Discussion

The technical specifications described in section 2.3 have been met for the newly created simulation. A cellular automaton was created using real-life fire spread dynamics to model fire spread. The simulation takes multiple types of real-world data into account and can be configured using many tunable parameters. A particle swarm optimization is used to minimise the fitness function by adjusting these parameters. The

simulation runs on the GPU and therefore supports the specified large scale fires while staying fast.

5.1 Computational performance

The computational performance shown in Figure 10 and Figure 9 is good. When looking back at the hypothetical proposed in section 2.2.7, it was stated that a reinforcement learning application for controlling fires could require somewhere in the magnitude of 88800000 simulated ticks to train the agent. Using the optimal thread count from Figure 9, a mean duration of 3200 microseconds is required to simulate 50 ticks. With this computational speed, simulating 88800000 ticks would take $88800000/50 * 3200\mu s \approx$ 1 hour and 35 minutes, which is a reasonable time for such an application. The simulation also scales well as the grid becomes larger, because the main bottleneck is the transfer of data from the CPU memory (random access memory) to the GPU memory. This is shown in Figure 10, the CPU time scales quadratically as the map size increases, while the time taken for the GPU implementation is nearly constant.

5.2 Accuracy

The purpose of this thesis was to develop a new method for simulating forest fires that strikes a balance between performance and accuracy. The performance angle certainly worked, however the accuracy is slightly lower than the existing solutions. However, the other solutions use slightly different methodologies, training and testing on only one to five fires, each having similar characteristics. Our solution trains and tests on every fire in southern Europe from 2001 to 2020, making

As mentioned in the background section, Zheng, et al. [29] use the area under the receiver operating characteristic curve (AUC) to evaluate a wildfire simulation. This evaluation method is also the method used in this thesis, so we can compare results. They trained on five different fires, getting an average AUC of 0.73 using an Extreme Learning Machine. However, with their methodology, training and testing was performed on different samples of the same fire event, which could inflate the testing scores. As shown in Table 3, experiment 3 is the best performing experiment using the method described in this paper. The model from this experiment achieves an AUC of 0.67 for the test set and an AUC of 0.77 for the training set. A testing AUC of over 0.7 is generally considered reasonable, therefore our testing results are not up to par.

Ganapathi, et al. [10] do not use the AUC method for determining the quality of their reinforcement learning based simulation, they measure the accuracy percentage instead. They also test on a different fire than they train on, however they train on a single fire and test on one that occurred in a nearby area, whereas our simulation trains on many fires spread across southern Europe. Moreover, they calculate their accuracy values after 16 days, 32 days, 48 days, and 64 days, whereas with our method this calculation occurs at whichever date the data for the given fire ends. Because of these changes in methodology it is difficult to accurately compare results, however AUC ranges from 0 to 1, and percentages range from 0 to 100, so keeping in mind the differences between AUC and accuracy we can still compare results. They tested several

reinforcement learning methods, including Gaussian processes, value iteration, policy iteration, Q-learning, Monte Carlo tree search, and A3C. The best performing method is A3C, with an accuracy of 90.1% after 16 weeks, 81.8% after 32 weeks, 50.8% after 48 weeks, and 13.4% after 64 weeks. There is a steep drop in accuracy after 48 weeks. Assuming no major class imbalance impacting the comparison between AUC and accuracy, the accuracy values after 16 and 32 weeks are better than our solution, but the accuracy values after this are worse.

5.2.1 Reasons for our results

Our results are slightly worse than existing solutions, although when accounting for the differences in methodology this is not so bad. Forest fires might also only be predictable to a certain point, limiting the accuracy we can really achieve with any simulation. However, assuming we are not at the limit of forest fire predictability, this section explores some possible reasons for our lower accuracy results.

Table 3 shows experimental changes in the PSO hyperparameters in an attempt to minimise the 5-fold average AUC score. The best configuration found is highlighted in green, any changes from this configuration are highlighted orange. From the AUC scores we can see that there is not much variance in the scores between the experiments ($\mu = 0.655, \sigma = 0.014$), implying that the problem may not lie with the hyperparameters, but with something else. One possibility is that the search space is too complex for the PSO, since there are over 50 dimensions in the search space, causing the optimizer to quickly converge to a local minimum and get stuck there.

Figure 7 shows the spread of each simulation parameter across the 5 folds. The values are mostly grouped together, implying each value did converge to approximately the same place each time, even after using a different dataset. This could mean they all converged to the same local minimum, however it is more likely they found a global minimum. This would indicate that the accuracy problem does not stem from a local minimum problem, at least not for the simulation parameters. Figure 8 shows the same information, but for each value of the spread rate of different types of land cover. Most of these values are much more spread out, indicating the PSO had more trouble finding a global minimum consistently for these parameters, or even at all.

Another explanation for the inaccuracy could be that the simulation is not configurable enough to generalise to all fires. If changing the parameters does not sufficiently change the outcome of the simulation, the PSO will not be able to optimise to an accurate solution. Similarly, a lack of data provided to the simulation could reduce the generalisation performance. For example, if wind data was not included and a training fire is moving eastward, the PSO could find a way to make the simulated fire also more eastward, but it would not have learned that wind causes the fire to move. If a fire is encountered during testing that spreads westward due to wind, it would not simulate it properly. Of course our simulation does consider wind, and the PSO can optimise this parameter, however it is possible that there are weather elements or other types of data that could improve the accuracy of our solution.

Zheng, et al. [29], use similar explanatory variables, such as temperature, elevation, and land cover. Additionally, they used aspect and slope. Aspect is the direction a structure or geographical formation is facing, so in mountainous regions it could

indicate which part is in the sun during the day, the slope describes the angle of the land. This information is contained in the elevation data, but it is not explicitly part of the data embedded in each cell of the simulation grid.

The mean training AUC for the optimal configuration found is 0.772 ($\pm$0.117). Since the training AUC is significantly larger than the testing AUC, this indicates that the simulation is able to fit the fires well, however the solution found by the optimizer does not generalise to fires outside of the training dataset as well as it possibly could. This low generalizability could be a result of missing data types for the simulation, as explained before.

Another detriment to the scores could be the quality of the data, specifically the resolution of the fire data. The resolution of the fire data is quite low compared to the elevation and land cover data. Because all data is scaled to the same resolution, the resolution of the fire data is artificially increased, which ends up with very square edges which are not naturally modelled with our simulation. Firefighting efforts are also missing from the data, in real life firefighters try to control forest fires, which affects their spread. The data of when and where firefighting efforts are performed is not available on the scale our simulation needs. This is an explanatory variable that is not taken into account in the simulation, and will therefore reduce the performance of the optimizer. Furthermore, some of the fires in the dataset have an empty starting state, which are impossible to simulate correctly, these instances also drag down the AUC.

Figure 11 to Figure 14 show the starting point, simulated burnt area, and the actual burnt area. Here the resolution problem is shown to an extent. The simulation works on a higher resolution than the fire data, because the fire data is lower resolution than the minimum spatial resolution specified for this simulation. Figure 11 and Figure 12 are two examples of fires that are are modelled fairly accurately ($AUC = 0.86$ & $AUC = 0.95$), but even these high AUC scores are reduced because of the higher resolution. The newly added simulated burnt area from the starting point matches well with the newly added actual burnt area.

Figure 13 shows an example of a badly modelled fire. The simulation seems to have spread too quickly in this case, or some other parameter could be wrong. Figure 14 shows a situation that also happens, albeit rarely. The dataset shows there is not a starting fire, however the last stage of the data shows an actual burnt area. If no starting fire is supplied to the simulation, nothing will happen, so these situations always result in an AUC of 0.5, i.e., completely wrong.

Figure 15 shows a part of the data used when simulating a fire, in each of these plots yellow is used for low values, and purple for low values. These grids all represent the same area, the bounds of which are determined by the fire shown in Figure 16. The land cover grid plot shows how easy the fire can spread on that type of land cover, with yellow being more and purple being less. The wind is shown in 2 plots, one showing the eastward wind component, the other showing the northwards component. The spatial resolution of the wind data is quite low, so there is only one data point on the x-axis being stretched out. This is not a problem, since the weather does not change that much over relatively small areas (the resolution of the weather dataset is 0.1 degrees, or about 9 kilometres).

Figure 16 shows that the fire starts in the north-east part of the map, it then spreads

southwards. This makes sense given the land cover grid and elevation data we see in Figure 15. The fire starts at a low point in the north and moves to a more elevated position south. The fire does not move west as much, because there is low-lying land there, and fires do not like to spread downwards. Moreover, the land cover data shows the east and south parts of the map are covered in more burnable land.

The result of our simulation in this environment is also shown in Figure 17. Each one of the four images represents a snap-shot in time during the simulation showing the state of the grid. In these figures the black area represents the burned area, the red area indicates burning area, and the green area indicates the unburned area. The same behaviour explained above is seen in this simulated fire. It moves mostly south and a little east, which again aligns with the available data.

5.3 Optimised simulation parameters

Simulation parameters The simulation parameters from the different folds are closely grouped together, as shown in Figure 7. The resulting values intuitively make sense, the activity threshold is low, meaning fire spreads to other cells easily. The height effect multipliers for uphill and downhill are above and below 1 respectively, because fire travels up a slope more easily.

Land cover parameters The land cover parameters shown in Figure 8 are more spread out. This makes sense, because there are more parameters to optimise, and they can only be meaningfully optimised if the fire occurs near that given land cover. Because of this, the land cover types that were encountered most often are more likely to be less spread out. From the figure we can see that continuous urban fabric is the parameter with the least variance between the folds. This makes sense because most fires occur near some human settlement, where such a land cover type exists, thus making the parameter correctly optimised. To further show this effect, we can look at the most varied parameter values, which are for peat bogs. This land cover type does not occur often, and even if it does it is only a small part of the grid, likely having very little influence on the spread of the fire, making it nearly impossible to optimise.

Sea and ocean, traditional woodland-shrub, natural grasslands, and broad-leaved forest also have little varying in their optimised values. Such land cover types have a large effect on the spread of fire, and they occur often in the datasets, so they are likely to be properly optimised by the PSO.

5.4 The optimizer of choice

Section 2.2.2 explains why the particle swarm optimization was chosen for this project. The PSO algorithm works well on a high-dimensional search space, and it does not require a gradient from the fitness function, which many other optimization algorithms do. As partly explained above, changing out the optimizer is not likely to improve the resulting accuracy. Our most important parameters, the simulation parameters, seem to land in a global minimum. Our main accuracy loss seems to come from the lack of resolution in our data, or the lack of other data types.

Over the course of this project the source of burnt area data was already switched out for a higher resolution one, however the resolution is still only 1 km by 1 km. Had we not looked for a global dataset, but rather looked for a more localised dataset, the spatial resolution might have been greater. However, our aim was to train on as many fires as possible, which led us to the global GlobFire dataset. For example, Zheng, et al. [29] trained and tested on only 5 fires, however, these fires had a much greater spatial resolution than what is available in GlobFire.

6 Conclusion

We asked whether it is possible to keep current state-of-the-art accuracy levels while creating a more computationally fast simulation. While this might be possible, the method used in this paper did not achieve the desired accuracy results. Our simulation does achieve similar, but slightly worse performance to existing solutions, even when the methodology used is more ambitious. A good use for this simulation could be a type of problem that requires a roughly accurate simulation, but needs to support a relatively large scale area with good computational performance, such as reinforcement learning.

There might be multiple reasons this simulation was not as accurate as expected. It might be impossible to predict forest fires with 100% accuracy. However, assuming we are not limited by the inherent unpredictability of forest fires, it could be the simulation not being configurable enough, resulting in the changes the PSO makes on the parameters not having enough effect on the outcome of the simulated fire. Another option could be that the search space is too complex. There are many parameters to be optimised, resulting in a 47 dimensional search space. The AUC scores could also be negatively impacted by the low resolution of the fire data.

6.1 Further development options

The simulation could be expanded with a friendly interface to simulate fires. This might be useful for just exploring what the simulation is capable of, or if implemented correctly, could be used by firefighters to help determine how to best control a wildfire by predicting its spread. The simulation can quite easily be extended to support a probability map for where the fire will spread, similar to how hurricane prediction maps are visualised. This could be accomplished by running many simulations with the same initial parameters and fire start area, but different seeds for the stochastic part of the simulation. Then visually overlap the predicted spread behaviours showing overlapping cells as more opaque.

Furthermore, it could be beneficial to remove the least impactful variables from the optimization process, especially some of the land cover types. With fewer dimensions, the search for an optimal solution should be easier for the particle swarm optimization. One could also take this a step further, and remove all land cover types from the optimization process that do not occur in the area of the fire that is currently being optimised.

6.2 Further research

Testing out different options for the evolutionary algorithm could be interesting, especially the Gaussian adaptation algorithm, since it is an evolutionary algorithm that fits for the same reasons as the PSO does. Furthermore, implementing this simulation in existing reinforcement learning wildfire applications could be a good test of the overall performance of the simulation, i.e., the computational speed benefits, and accuracy.

The generalisation ability of the simulation might also be improved by using more real-world data, such as the weather data available in the ERA5 dataset. This dataset contains many variables, such as the leaf area index, which represents the amount of vegetation in the area. The more relevant data the simulation has, the better it should generalise to new fires, since the optimizer should learn how to correct for every variable.

References

[1] M. Abrams, R. Crippen, and H. Fujisada. "ASTER global digital elevation model (GDEM) and ASTER global water body dataset (ASTWBD)". In: *Remote Sensing* 12.7 (2020), p. 1156.

[2] T. Artés et al. "A global wildfire dataset for the analysis of fire regimes and fire behaviour". In: *Scientific data* 6.1 (2019), pp. 1–11.

[3] J. W. Backus and W. P. Heising. "Fortran". In: *IEEE Transactions on Electronic Computers* 4 (1964), pp. 382–385.

[4] L. Bodrožić, J. Marasović, and D. Stipaničev. "Fire modeling in forest fire management". In: *CEEPUS Spring School, Kielce, Poland* (2005).

[5] S. Cook. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2012. ISBN: 9780124159334.

[6] M. A. Crowley et al. "Multi-sensor, multi-scale, Bayesian data synthesis for mapping within-year wildfire progression". In: *Remote sensing letters* 10.3 (2019), pp. 302–311.

[7] T. Curt, A. Aini, and S. Dupire. "Fire Activity in Mediterranean Forests (The Algerian Case)". In: *Fire* 3.4 (2020), p. 58.

[8] M. Denham et al. "Dynamic data-driven genetic algorithm for forest fire spread prediction". In: *Journal of Computational Science* 3.5 (2012), pp. 398–404.

[9] M. Dorigo, M. A. M. de Oca, and A. Engelbrecht. "Particle swarm optimization". In: *Scholarpedia* 3.11 (2008), p. 1486.

[10] S. Ganapathi Subramanian and M. Crowley. "Using spatial reinforcement learning to build forest wildfire dynamics models from satellite images". In: *Frontiers in ICT* 5 (2018), p. 6.

[11] J. Glasa and L. Halada. "On elliptical model for forest fire spread modeling and simulation". In: *Mathematics and Computers in Simulation* 78.1 (2008), pp. 76–88.

[12] T. Hammond et al. “Forest Fire Control with Learning from Demonstration and Reinforcement Learning”. In: *2020 International Joint Conference on Neural Networks (IJCNN)*. Piscataway, NJ, USA: IEEE, 2020, pp. 1–8.

[13] C. R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[14] M. Hefeeda and M. Bagheri. “Forest fire modeling and early detection using wireless sensor networks.” In: *Ad Hoc Sens. Wirel. Networks* 7.3-4 (2009), pp. 169–224.

[15] M. Keil et al. “Creation of a High-Resolution Product CLC2006_Backdating by a Backward Look from the Digital Land Cover Model DLM-DE2009 to 2006-A Contribution to the German CORINE Land Cover 2012 Project within a Bottom-Up Approach.” In: *35th International Symposium on Remote Sensing of Environment (ISRSE35)*. Vol. 40. Oberpfaffenhofen, Germany: Copernicus Publications, 2015, pp. 1093–1100.

[16] J. Kennedy and R. Eberhart. “Particle swarm optimization”. In: *Proceedings of ICNN’95-international conference on neural networks*. Vol. 4. IEEE. Indianapolis, USA: IEEE, 1995, pp. 1942–1948.

[17] G. Kjellström. “On the efficiency of gaussian adaptation”. In: *Journal of Optimization Theory and Applications* 71.3 (1991), pp. 589–597.

[18] E. Koo et al. “Modelling firebrand transport in wildfires using HIGRAD/FIRETEC”. In: *International journal of wildland fire* 21.4 (2012), pp. 396–417.

[19] L. J. V. Miranda. “PySwarms, a research-toolkit for Particle Swarm Optimization in Python”. In: *Journal of Open Source Software* 3 (21 2018). DOI: 10.21105/joss.00433. URL: https://doi.org/10.21105/joss.00433.

[20] S. Mirjalili. “Genetic algorithm”. In: *Evolutionary algorithms and neural networks*. Brisbane, QLD, Australia: Springer, 2019, pp. 43–55.

[21] J. Muñoz-Sabater et al. “ERA5-Land: A state-of-the-art global reanalysis dataset for land applications”. In: *Earth System Science Data Discussions* (2021), pp. 1–50.

[22] N. Quintero et al. “Assessing landscape fire hazard by multitemporal automatic classification of Landsat Time Series using the Google Earth Engine in West-Central Spain”. In: *Forests* 10.6 (2019), p. 518.

[23] D. Ritchie and B. Kernighan. *The C programming language*. Madison, USA: Bell Laboratories, 1988.

[24] S. Sakellariou et al. “Review of state-of-the-art decision support systems (DSSs) for prevention and suppression of forest fires”. In: *Journal of Forestry Research* 28.6 (2017), pp. 1107–1117.

[25] B. Stroustrup. *The C++ programming language*. 3rd. Boston, USA: Addison-Wesley, 1997. ISBN: 978-0-201-88954-3.

[26] G. Van Rossum and F. L. Drake Jr. *Python reference manual*. Amsterdam, The Netherlands: Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[27] A. L. Westerling et al. “Warming and earlier spring increase western US forest wildfire activity”. In: *science* 313.5789 (2006), pp. 940–943.

[28] M. A. Wiering, F. Mignogna, and B. Maassen. “Evolving neural networks for forest fire control”. In: *Proceedings of the 14th Belgian-Dutch Conference on Machine Learning*. Enschede, The Netherlands: University of Twente, 2005, pp. 113–120.

[29] Z. Zheng et al. “Forest fire spread simulating model using cellular automaton with extreme learning machine”. In: *Ecological Modelling* 348 (2017), pp. 33–43.

A Appendix

A.1 Initial parameters

Table 4 shows the initial spread rates for each type of land cover.

A.2 Land cover types

- Continuous urban fabric
- Discontinuous urban fabric
- Industrial or commercial units
- Road and rail networks and associated land
- Port areas
- Airports
- Mineral extraction sites
- Dump sites
- Construction sites
- Green urban areas
- Sport and leisure facilities
- Non-irrigated arable land
- Permanently irrigated land
- Rice fields
- Vineyards
- Fruit trees and berry plantations

- Olive groves
- Pastures
- Annual crops associated with permanent crops
- Complex cultivation patterns
- Land principally occupied by agriculture, with significant areas of natural vegetation
- Agro-forestry areas
- Broad-leaved forest
- Coniferous forest
- Mixed forest
- Natural grasslands
- Moors and heathland
- Sclerophyllous vegetation
- Transitional woodland-shrub
- Beaches, dunes, sands
- Bare rocks
- Sparsely vegetated areas
- Burnt areas
- Glaciers and perpetual snow
- Inland marshes
- Peat bogs
- Salt marshes
- Salines
- Intertidal flats
- Water courses
- Water bodies
- Coastal lagoons
- Estuaries
- Sea and ocean

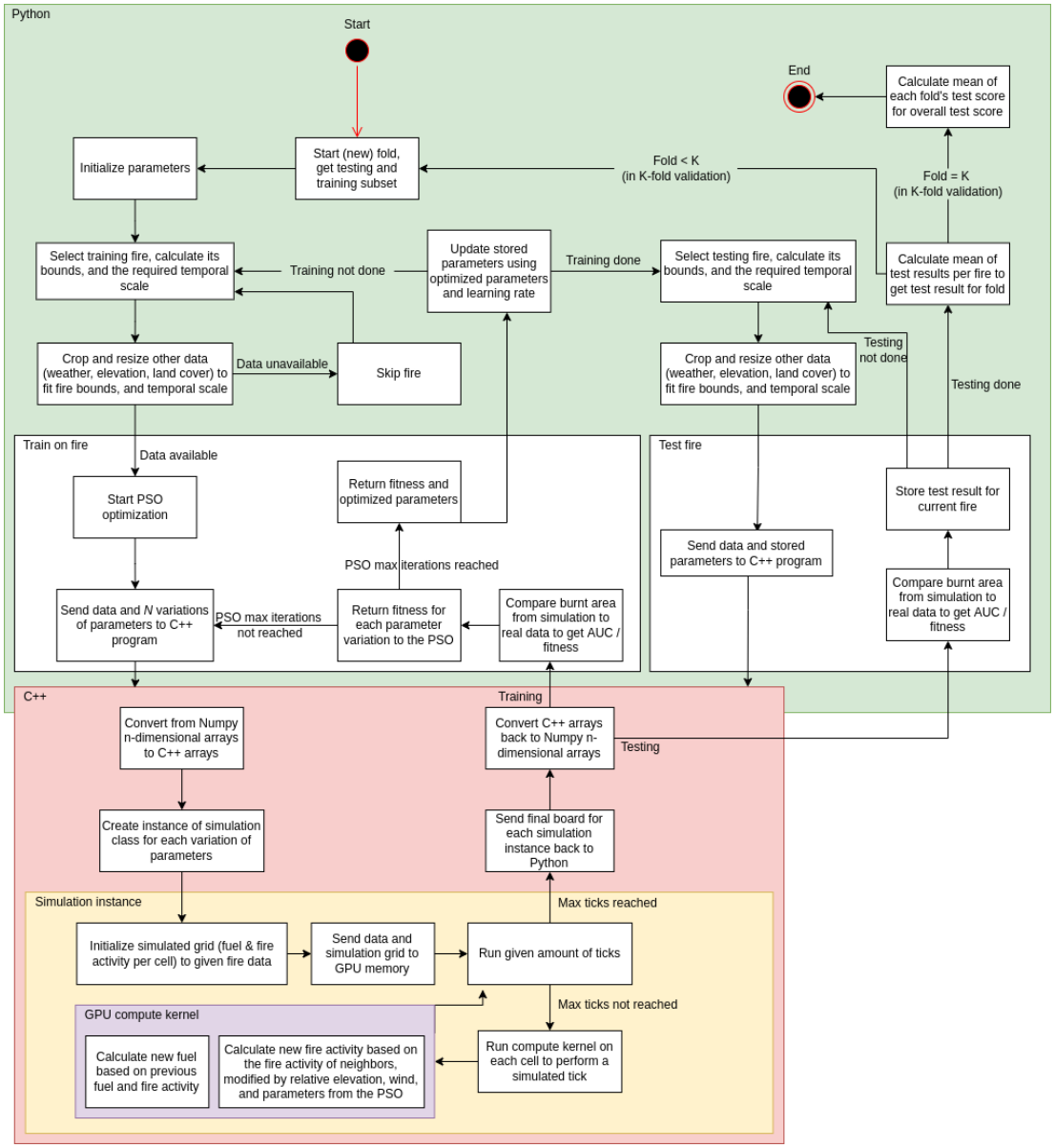


Figure 6: Flowchart for the implementation of the simulation.

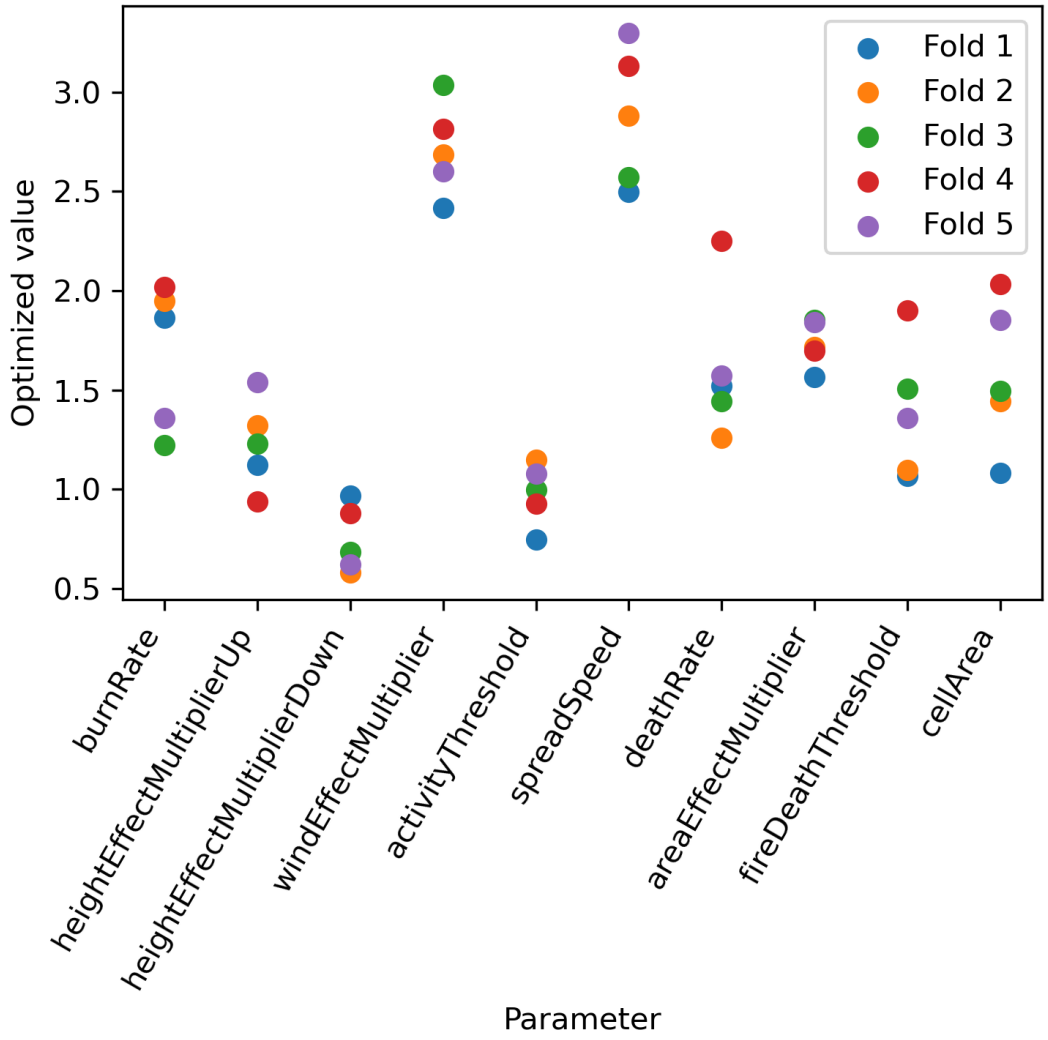


Figure 7: Values of simulation parameters after each fold.

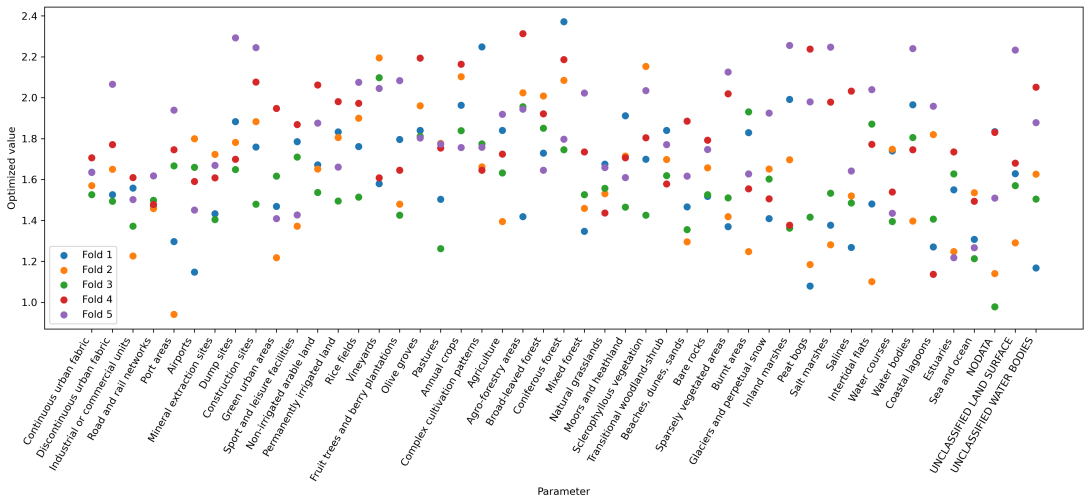


Figure 8: Values of land cover rate parameters after each fold.

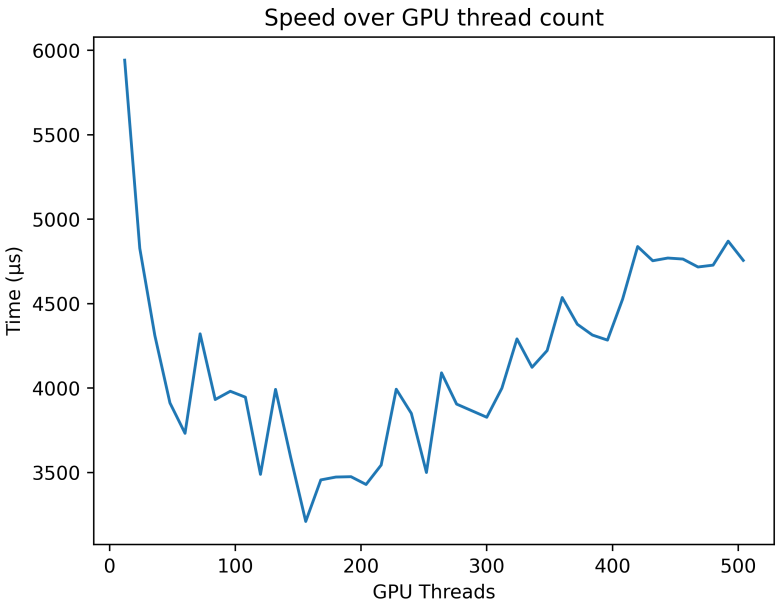


Figure 9: GPU time taken when using different thread counts.

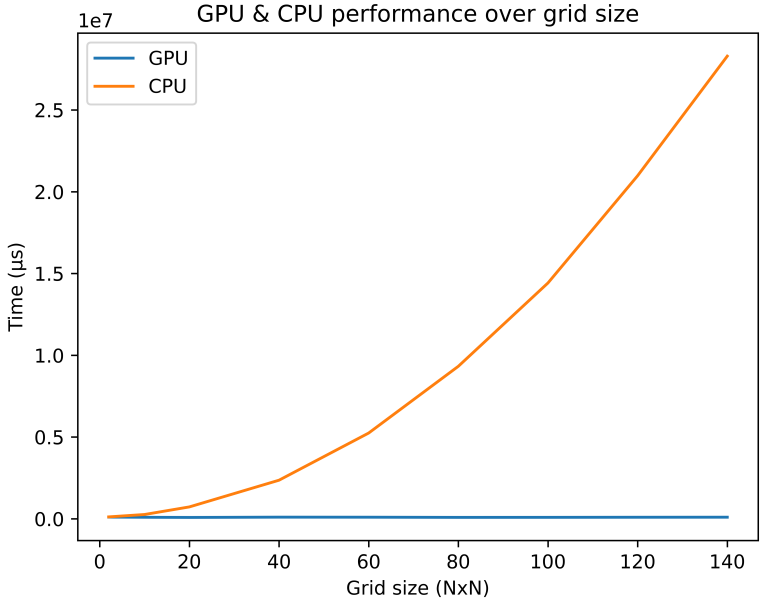


Figure 10: Mean CPU & GPU time taken for simulation running for 100 ticks on grids with increasing resolutions.

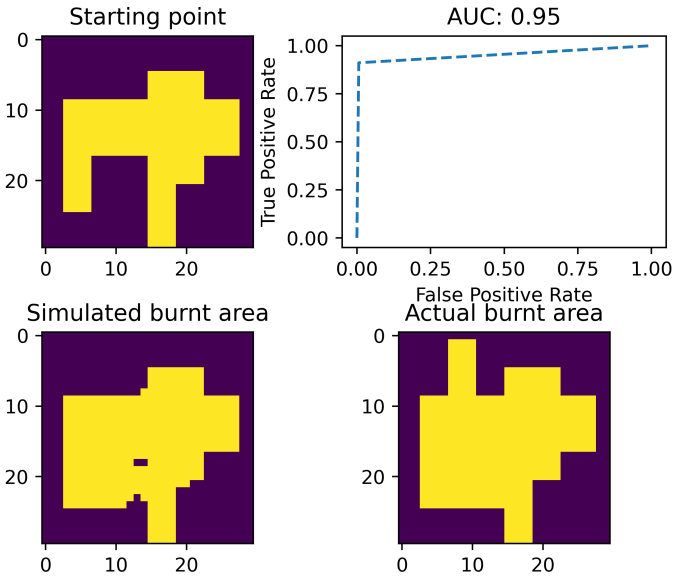


Figure 11: Visualisation of burnt area with good match between simulated and real burnt area.

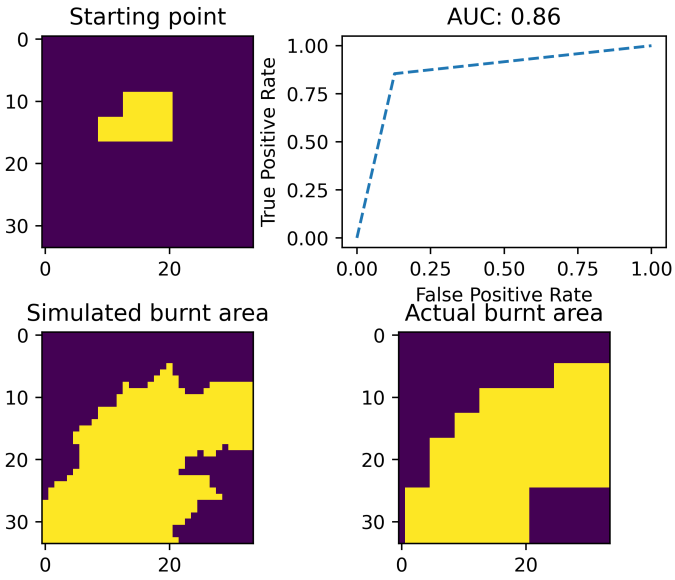


Figure 12: Visualisation of burnt area with good match between simulated and real burnt area.

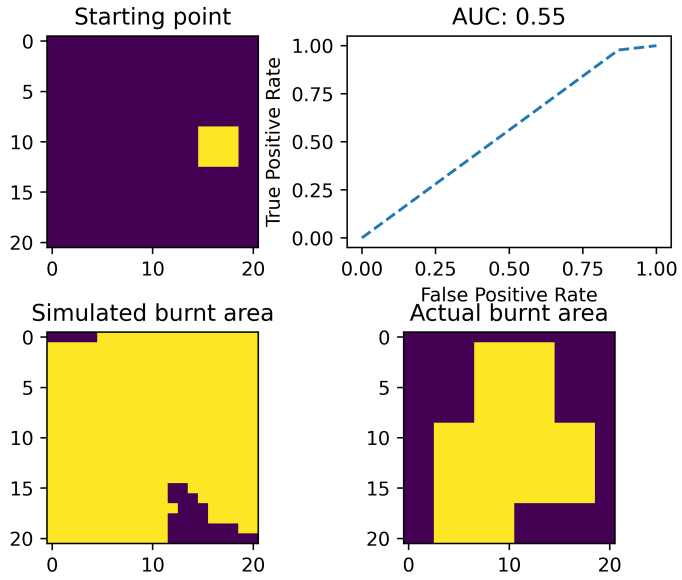


Figure 13: Visualisation of burnt area with poor match between simulated and real burnt area.

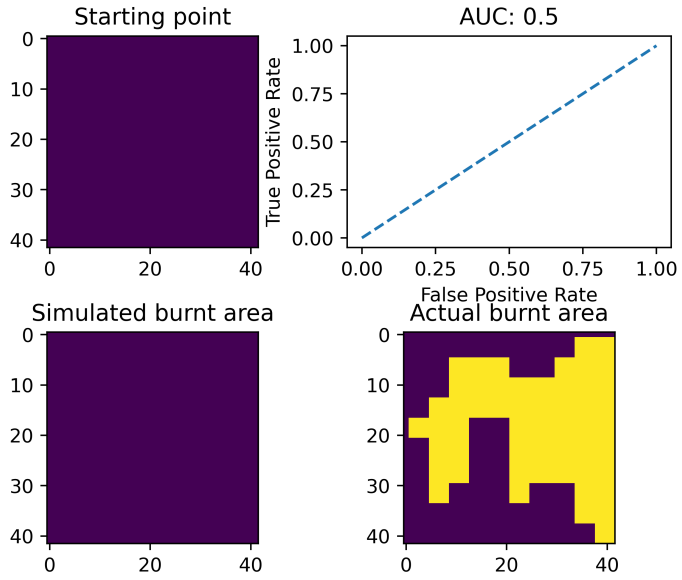


Figure 14: Visualisation of burnt area with no match between simulated and real burnt area.

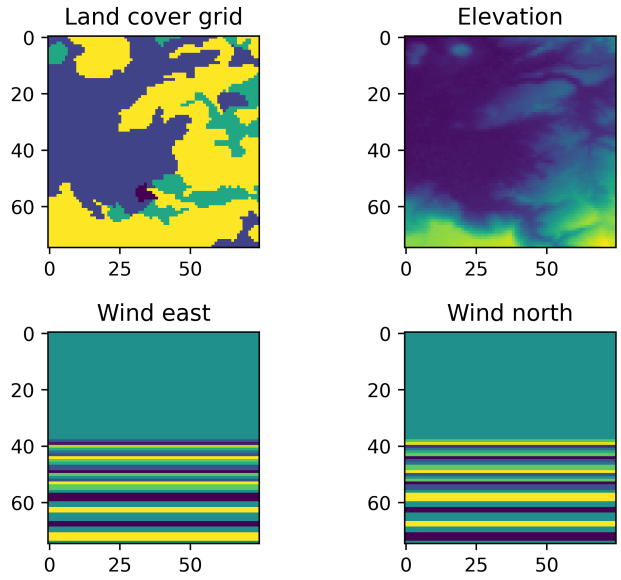


Figure 15: Visualisation part of the real-world data used for a simulated fire. Yellow for high values, purple for low values. The land cover grid shows spread rates from the type of land cover.

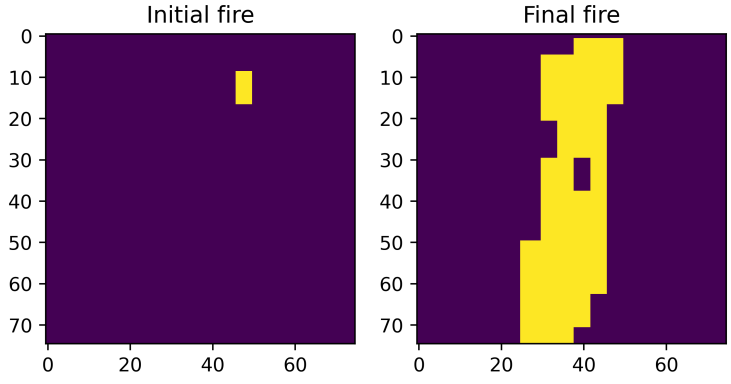
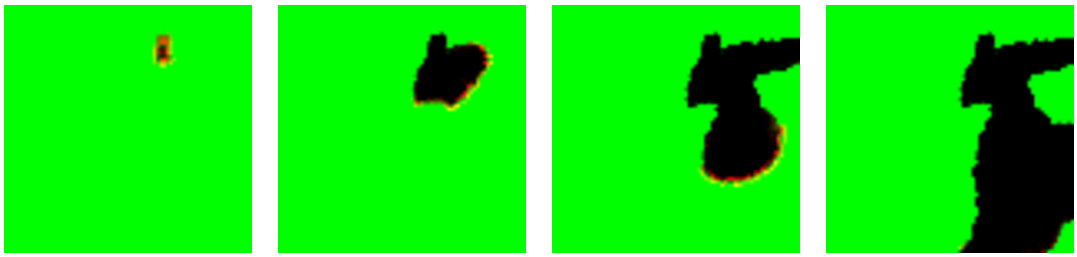


Figure 16: Visualising the burnt area for the fire mentioned in Figure 15.



(a) Start (b) Intermediate step (c) Intermediate step (d) End

Figure 17: Snapshots of the simulated grid during a simulation of the fire from Figure 16.

Label	Spread rate
Continuous urban fabric	0.1
Discontinuous urban fabric	0.1
Industrial or commercial units	0.1
Road and rail networks and associated land	0.2
Port areas	0.2
Airports	0.1
Mineral extraction sites	0.1
Dump sites	0.5
Construction sites	0.2
Green urban areas	0.5
Sport and leisure facilities	0.2
Non-irrigated arable land	0.6
Permanently irrigated land	0.5
Rice fields	0.9
Vineyards	0.9
Fruit trees and berry plantations	1
Olive groves	1.2
Pastures	0.6
Annual crops associated with permanent crops	0.7
Complex cultivation patterns	0.6
Land principally occupied by agriculture, with significant areas of natural vegetation	0.8
Agro-forestry areas	1
Broad-leaved forest	1
Coniferous forest	1
Mixed forest	1
Natural grasslands	0.6
Moors and heathland	0.7
Sclerophyllous vegetation	1
Transitional woodland-shrub	0.8
Beaches, dunes, sands	0.1
Bare rocks	0.1
Sparsely vegetated areas	0.3
Burnt areas	0.5
Glaciers and perpetual snow	0.1
Inland marshes	0.4
Peat bogs	0.5
Salt marshes	0.5
Salines	0
Intertidal flats	0
Water courses	0
Water bodies	0
Coastal lagoons	0
Estuaries	0.1
Sea and ocean	0
NODATA	0.3
UNCLASSIFIED LAND SURFACE	0.3

Table 4: Initial spread rates before any optimization happens to adjust them.