Universitat Autònoma
de Barcelona

Computer Architecture and
Operating Systems Department

Master in
Computational Science and Engineering

Efficient knowledge retrieval to calibrate input variables in forest fire prediction

MSc research project for the “Master in Computational Science and Engineering” submitted by KERSTIN WENDT, advised by ANA CORTÉS FITÉ. Dissertation done at Escola Tècnica Superior d’Enginyeria (Computer Architecture and Operating Systems Department).

July 2008

Trabajo de investigación

Máster en Ciencia e Ingeniería Computacional

Curso 2007-08

Efficient knowledge retrieval to calibrate input variables in forest fire prediction

Autor: Kerstin Wendt
Director: Ana Cortés Fité

Departamento Arquitectura de Computadores y Sistemas Operativos
Escuela Técnica Superior de Ingeniería (ETSE)
Universidad Autónoma de Barcelona

Firmado

Autor Director

Abstract

Forest fires are a serious threat to humans and nature from an ecological, social and economic point of view. Predicting their behaviour by simulation still delivers unreliable results and remains a challenging task. Latest approaches try to calibrate input variables, often tainted with imprecision, using optimisation techniques like Genetic Algorithms. To converge faster towards fitter solutions, the GA is guided with knowledge obtained from historical or synthetical fires.
We developed a robust and efficient knowledge storage and retrieval method. Nearest neighbour search is applied to find the fire configuration from knowledge base most similar to the current configuration. Therefore, a distance measure was elaborated and implemented in several ways. Experiments show the performance of the different implementations regarding occupied storage and retrieval time with overly satisfactory results.

Keywords

Forest fire, Simulation, Knowledge-guided Genetic Algorithm, Nearest neighbour, Distance measure

Resumen

Los incendios forestales son una grave amenaza para seres humanos y para la naturalza desde el punto de vista ecológigo, social y económico. Predecir su comportamiento usando simulaciones todavía da resultados poco fiables y sigue siendo una tarea desafiante. Trabajos más recientes, intentan calibrar variables de entrada, muchas veces imprecisas, aplicando técnicas de optimización como algoritmos genéticos. Para converger más rápido hacia soluciones más adecuadas, el algoritmo genético es guiado con conocimiento obtenido de fuegos históricos o sintéticos.
Hemos desarrollado un método robusto y eficiente para almacenar y recuperar ese conocimiento. Aplicamos la búsqueda del vecino más cercano para encontrar la configuración del fuego más similar a la configuración actual dentro de la base de conocimiento. Para esto, hemos elaborado una función de distancia y la hemos implementado de diferentes maneras. Experimentos muestran el rendimiento de las distintas implementaciones considerando el almacenamiento ocupado y el tiempo de recuperación con resultados muy satisfactorios.

Palabras claves

Incendios forestales, Simulación, Algoritmo genético guiado por conocimiento, Vecino más cercano, Función de distancia

Resum

Els incendis forestals són una amenaça important tant pels homes com per a la natura des d'un punt de vista ecològic, social i econòmic. La predicció del comportament dels incendis forestals utilitzant simulació encara genera resultats poc fiables i, per tant, segueix essent un desafiament important. Aproximacions recents a aquest problema, intenten calibrar les variables d'entrada dels simuladors, les quals sovint presenten un grau important d'incertesa, utilitzant tècniques d'optimització com poden ser els Algoritmes Genètics (AG). Per tal de que la convergència dels AG a una solució bona sigui ràpida, l'AG es guia mitjançant el coneixement obtingut d'històrics d'incendis o focs sintètics.
Per aquest treball s'ha desenvolupat un mètode eficient i robust d'emmagatzemament i recuperació del coneixement. El mètode anomenat Nearest Neighbour Search s'aplica per trobar la configuració guardada en la base de coneixements que més s'assembli a la configuració real de l'incendi. Per a tal efecte, s'ha desenvolupat una mètrica de distància la qual ha estat implementada de varies formes alternatives. L'experimentació realitzada mostra resultats encoratjadors en el rendiment de les diferents implementacions tenint compte l'emmagatzemament ocupat i el temps de recuperació de la informació.

Paraules claus

Incendis forestals, Simulació, Algoritmes genètics guiats pel coneixement, Veïns-propers, Mètrica de distancia

Contents

Chapter 1

Introduction

1.1 Motivation

Predicting the behaviour of forest fires is an art - as much as it is a science. Even experienced fire fighters may encounter problems reading fire behaviour and predicting a fire's potential threat to lives and property. False and unreliable predictions may lead to tragedy!

Where forest fires are not a natural part of the ecosystem and where they cannot be prevented, their prediction has become a crucial topic. Forest fires not only damage important ecological resources like forest, scrubland and grassland but also count for a global source of emissions to the atmosphere. Wildland fires provoke climate changes and the area where a fire has passed is very susceptible to erosion, possibly being the cause for floods. But fires are not only a threat to nature. Due to the ever increasing world population they also have become a danger for human lives destroying buildings and infrastructure and claiming deaths. Having mentioned the negative consequences of wildland burnings to ecology, society and economy, the best strategies are needed to quickly and efficiently extinguish an ongoing fire to minimise its effects. Up-to-the-minute satellite mapping and weather information together with remote sensing technologies, computer modelling and internet communications have changed the course of fire behaviour analysis, lifting the task of fire suppression to a new level [1]. Especially computer modelling and simulation tools are used to predict the fire front for a given time in the near future in order to better coordinate disaster groups, decide on which front to fight and with which instruments.
But also fire prevention politics benefit from fire propagation simulation tools in order to elaborate fire risk maps or to experiment the correct location of fire breaks. In both cases, effective tools are needed to forecast the propagation of forest fires the most reliably possible.

This work is part of the research undertaken to enhance the prediction results of current fire simulators. The next section introduces some forest fire characteristics and presents the two starting points to fight against this hazard - prevention and suppression.

1.2 Forest Fire Characteristics

Forest fires, also known as wildfires or wildland fires, are non-structure fires that occur in the wildland. They either start naturally caused by lightning or sometimes by spontaneous combustion of dry fuel, or in their majority are provoked by human carelessness due to smoking or recreational activities or, unfortunately, on purpose. Human-caused fires constitute the much greater percentage of forest fires, but natural fires account for the larger total area burned because natural fires can burn for hours before being detected by fire fighters whereas human-caused fires tend to be detected early [2]. There exist smouldering fires without flames but with a lot of smoke, flaming and glowing combustions.
In order to burn, a wildland fire needs sufficient heat, oxygen, and fuel; three components which together form the so-called fire triangle depicted in figure 1.1. The fire's rate of combustion is usually restricted by one of the three elements. Although it is generally accepted that wildfires form a natural part of the ecosystem, some environments suffer from too much fire [3].

Figure 1.1: *Fire triangle* indicating the three elements that are required for a forest fire to burn. Taken from [4].

To fight against this real threat there are two starting points: prevention and suppression.

Prevention

Among the forest fire research community fire risk analysis is an emerging field of research [5]. The elaboration of fire risk maps, taking into account meteorological phenomena like extreme heat, cyclical

climate changes and droughts contributing to an increased risk of forest fires as well as the specific vegetation of the area, may help to position fire lookouts, still used in many countries around the world for early detection. But also terrain planning including natural and artificial fire breaks and the reduction of excessive fuel by controlled burns or physical fuel removal may help to lower the risk of forest fires.

Suppression

Prescribed burns are only partially effective, because many catastrophic fires are wind driven where the amount of fuel is not the most important factor in determining fire spread. In order to fight forest fires, fast detection is a key factor. This can be done by fire lookouts or at an automated level using local sensor networks, infrared scanning towers and satellite and aero monitoring [3]. The majority of forest fires are suppressed before growing out of control applying water-spraying fire apparatus, flame retardant chemicals, and the construction of fire breaks.

In both cases, efficient tools are needed to calculate forest fire propagation: simulating forest fires to establish fire risk maps or predicting the fire line in case of an ongoing burning. But before stepping into details of forest fire simulation and prediction, in the next chapter we will situate the topic in the area of Computational Science where mathematics mix with computer science and natural sciences.

1.3 Computational Science

Computational Science - often also referred to as *Computational Science and Engineering* (CS&E) or *Scientific Computing* - is, simply spoken, using computers to do science. If, in former times, researchers mainly used experiments and observation or theory, the traditional forms of science and engineering, for scientific investigation and engineering design, computational science now is the widely accepted crucial third pillar to tackle scientific problems.

We wish to gain understanding of real-world processes such as weather and climate prediction, air flow around a plane or the design and control of vehicles, and are searching for an effective way to better understand the world around us. If we want to know how nature behaves we have to apply domain expertise from physics, biology, chemistry and others, and analyse the phenomena with mathematical algorithms and models. We can then use the computer to implement these models. Thus, we will be able to run simulations of real-world occurrences that can be used instead of time-consuming and cost-intensive experiments [6].

Typically, the implemented models require a massive amount of calculations because of the sheer size of many complex nonlinear challenges. This is why the emergence of computational science became necessary. And it became possible due to the development of parallel computation and high performance computing. Many current difficult problems require new computational tools and concepts in order to discover, understand, and design solutions to these problems and their related mathematical structures. Thus, the use of supercomputing, parallel processing, and sophisticated algorithms, is inevitable and will also serve to satisfy the quest for ever higher levels of detail and realism in models and simulations.

Summarising, Computational Science and Engineering is an interdisciplinary field at the intersection of three domains: applied mathematics, computer science, and the (social and natural) sciences as shown in figure 1.2. It focuses on the integration of knowledge from the three domains and on the development of problem-solving methodologies and robust tools. Scientific computing combines domain expertise, mathematical modelling, numerical analysis, algorithm development, software implementation, program execution, analysis, validation and the visualisation of results.

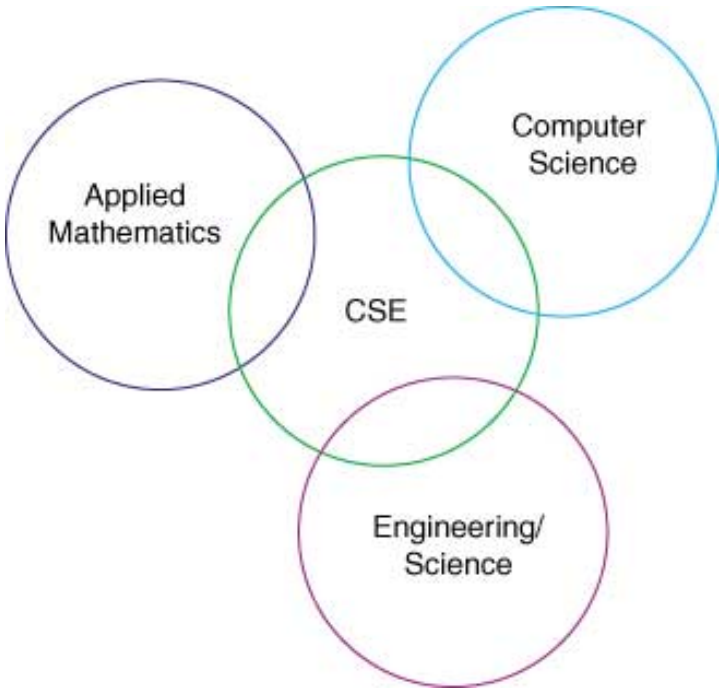

Figure 1.2: Position of Computational Science and Engineering as a multidisciplinary field connecting applied mathematics, engineering and (natural and social) sciences and computer science. Taken from [7].

Computational Science still resides at an early stage of development and is a rapidly growing multidisciplinary area. It will play an important, if not dominating, role for the future of the scientific discovery process and engineering design. Until now, much of Computational Science and Engineering has involved analysis, but the future will move towards optimisation and design, especially in the presence of uncertainty [7]. And it is to optimisation to which the current work contributes.
The physical behaviour of forest fires is already captured in well-established and widely agreed mathematical models (see section 2.2). There also exists a variety of computational tools (see section 2.3) to apply these models and to run simulations to gain a deeper understanding of fire spread and to make predictions of the progress of the fire front in an ongoing burning. But we will see that current predictions are not entirely reliable because of the uncertainty of input parameters. By removing part of this uncertainty and enhancing input parameters the overall prediction result will be enhanced, too. Before explaining details on fire simulators, the following chapter presents simulation from a general point of view.

1.4 Simulation

The advances in computer technology cleared the way for computer simulation which tries to model a real-life situation on a computer in order to study how the system works. By changing input variables, predictions about the behaviour of the system may become possible.
Computer simulation has become a fundamental part of modelling natural systems in natural sciences and engineering, and human systems in social science and economics. If the formal modelling of systems has been carried out via a mathematical model, computer simulation now is often used to supplement or substitute modelling systems for which simple analytic solutions cannot be given. Computer simulation thus gave rise to a new way of doing science. And in some cases, outcomes from simulations will lead to further questions about the phenomenon being studied.

The great advantage of computer simulation is that it can be used when physical experiments are too costly or time consuming. There also exist natural phenomena that would be too dangerous or even impossible to be studied by direct experimentation.
In the case of wildland burnings, it results infeasible to reproduce a fire in order to be able to monitor the impact of changed initial conditions. It is also impractical to initiate a huge forest fire just for observation purposes; costs and benefits would be out of all proportion to each other.
Although small burnings are accomplished to a minor extent for investigation, they only allow capturing a minimum range of possible input variables to measure their effect on propagation. By contrast, simulation permits the application of

a wide range of input variables and observing the result without damaging the environment.
Regarding the present work on enhanced fire prediction, we use simulation to know about how the fire will be like before it has finished. In doing so, fire fighters receive decision support and may be able to implement counteractions on time.

1.5 Contributions and Outline

In Computational Science and Engineering until now much work has involved analysis, but the future is moving towards optimisation and design. The present work contributes to the optimisation of an application situated in CS&E.
Especially in the context of forest fire prediction, correct and reliable prediction results are essential for decision support in case of an ongoing fire but also for planning and prevention. And not only trustworthy simulation results are required but we need to be able to dispose of outcomes in a given margin of time to be of use for fire fighters.
This thesis is based on the data-driven fire prediction method using a knowledge-guided Genetic Algorithm to calibrate input variables with the intention to speed up and improve prediction results realised by [8]. She developed a working yet simple approach to store and to retrieve the knowledge used to guide the Genetic Algorithm. But special attention has to be paid to data storage as well as data retrieval that this process does not turn out to be a bottleneck of the application or that it penalises the improvements accomplished by [8] more than needed.
Within the scope of the present work we will suggest a more robust and above all scalable approach to store and retrieve the knowledge, which is independent of the used fire simulator as well as of the underlying used database where the knowledge will be stored. Applying a database management system, it becomes possible to store and maintain an augmented amount of accurate knowledge, nearly without increasing retrieval times. Thus, we are able to provide more precision and to fasten and improve the predictions of forest fires.

The remaining work is organised as follows: The subsequent chapter 2 gives an introduction to forest fire simulation. The widely used fire behaviour prediction model by Rothermel is presented, followed by an overview of some fire behaviour simulators, concentrating on fireLib which is the basic framework for this thesis. Chapter 2 finishes with an examination of input variables required by fire simulators.
Chapter 3 describes the process from fire simulation to fire prediction. We present the classical prediction method, i.e. simple simulation, and show its drawbacks. These can be partly removed applying a data-driven prediction approach.
Precisely this last approach is explored in detail in chapter 4. Besides the usage of

a Genetic Algorithm to calibrate fire simulator input variables, it comprises advices how to guide the Genetic Algorithm using domain knowledge to converge faster towards fitter solutions.
Chapter 5, the main chapter of the present work, explains the retrieval of domain knowledge in an elaborated manner. First, the former naïve approach is mentioned. Afterwards, the theoretical principles and implementational design issues of the improved method are described.
In chapter 6 the results of conducted experiments are presented to back up the assumptions and expectations of the prior chapters, before the work is terminated with conclusions and some proposals for future enhancements in chapter 7.

Chapter 2

Forest Fire Simulation

2.1 Overview

With the progress and development of powerful parallel computing machines in recent years, modelling of complex problems has become possible. Modelling of real-life occurrences, e.g. forest fires, can be performed using simulation 2. Simulation requires a model of the phenomenon to simulate which then often is implemented in a simulation tool.
Forest fire simulation can be used to support fire management decisions, as a training tool improving fire management skills, and it can also help displaying and explaining fire behaviour to the general public. But simulation will never be able to replace the knowledge and experience of qualified wildland managers because of its inherent limitations. Current models still have difficulties simulating several important conditions related to fire behaviour [9].
At present, there exists a series of established and widely used fire simulators. Current research tries to improve simulation results linking fire simulators to geographic and meteorological information systems in order to obtain more precise input variables and to dynamise the process of fire simulation.
This chapter introduces the widely used fire behaviour prediction model by Rothermel 2.2 and then gives an introduction to fire behaviour simulators 2.3. In the corresponding section we will concentrate on *fireLib*, the simulator used in the present work. The chapter concludes with a detailed examination of environmental variables 2.4 used as input by fire simulators to calculate the fire propagation. Studying these variables in detail allows us to search explicitly for ways to remove part of their imprecision which often leads to unreliable simulation results.

2.2 Fire Behaviour Prediction Model

In many scientific areas, particularly in the natural sciences and engineering disciplines, it has become common practice to represent physical systems by the use of, mainly mathematical, models. The mathematical model usually is a set of variables and a very large collection of equations and inequalities that establish relationships between the variables. The model receives an input, representing the specific conditions, and provides the output which represents the evolution of the modelled system [10].
The complexity of the model, in general, involves a trade-off between simplicity and accuracy of the model. Added complexity normally improves the fit of a model but it can result in a model difficult to understand and to work with. An excessive model complexity can even pose computational problems, including numerical instability. Normally, models are integrated in a form of simulation tool that can be run on a computer.

The quasi standard model to simulate fire behaviour was proposed by Rothermel in 1972 and consists of about 80 equations [11]. It is the most frequently used fire behaviour model in forest fire research and implemented by the majority of fire simulators (see Simulators). Rothermel's model considers an input parameter with 17 features which describe terrain, type of fuel, fuel moisture and wind. Furthermore, it assumes that all input variables are homogeneous for a sufficiently small area and time period. The output of the calculation is the rate of fire spread, the direction of maximum spread and the effective wind speed. The model predicts the flaming front of the fire and not the burnout that occurs after the front has passed.
Moving from the original problem to the mathematical model almost always requires making some simplifying assumptions. The model has to strike the balance between its degree of abstraction and a sufficient complexity for reasonable results.

The model proposed by Rothermel assumes that [12]

1. the fuel strata carrying the fire is continuous and uniform
2. the fire is spreading in surface fuels
3. the wind speed, slope, and fuel moisture values remain reasonably constant.

Moreover, the model possesses some limitations [12]:

1. It does not consider fire spread due to spotting.
2. Crown fires are not modelled, but the potential is predicted.
3. Fire whirls and other similar extreme, fire-induced atmospheric disturbances are not modelled.

This, as well as user skills and knowledge and the accuracy of input values, is important for the final result and will determine and limit the prediction precision. It is not possible to predict exactly what a fire will do, but it is possible and useful to obtain a reasonable estimate [12]. The more exactly the input variables, the more exactly the output, hence the prediction result, will be.
The next section presents some of the main forest fire simulator tools.

2.3 Fire Behaviour Simulators

Fire behaviour simulators are tools which implement a fire behaviour prediction model and which can be used to simulate forest fires. Often a graphical user interface is provided for a better handling of input variables and a comprehensible display of propagation results. Fire simulators may vary in the required format of input variables, output variables, implemented forest fire models, respective capabilities, specific restrictions, and addressed end user groups.
In order to be able to improve the result of simulators it is necessary to fully understand its functionality - how it treats and manipulates data. However, not the simulator itself is to be changed, but its input variables during a running simulation. Most fire simulators still work with a set of static input variables, i.e. these do not change during an ongoing simulation.
In addition to the input variables describing the environmental conditions, fire simulators require the initial fire line from which to start the simulation. Often this is entered as a list of burnt cells being part of the area where the fire takes place. Fire simulators based upon the cellular automaton approach divide the area in question into rectangles of user-defined size. Starting the simulator lets the fire spread from burnt cells to unburnt neighbouring cells.
Next, some well-established tools to simulate fire behaviour are presented with a focus on *fireLib*, the simulator used in this thesis.

FARSITE

FARSITE (Fire Area Simulator) is frequently used by biologists or ecologists, taking advantage of its ability to run on the Windows operating system. This simulator requires GIS (Geographic Information System) data and utilises standard fuel models. It predicts the fire spread in two dimensions and outputs its results as real-time on-screen graphics, portable to other desktop applications or GIS. FARSITE is a deterministic model [13] and calculates the behaviour of fire under heterogeneous conditions of terrain, fuels, and weather. Because of its complexity, fire behaviour training or sufficient experience are a requisite for its proper use.

BehavePlus

This fire simulator was originally written in FORTRAN and can be considered as the successor of the BEHAVE simulator (1986). The most recent version includes user-friendly graphical interfaces and estimates wildland fire behaviour under various fuel, weather, and topographic situations. Input values are entered directly or by utilising input "guide screens" which display valid ranges of values. BehavePlus supports other fire phenomena as crown fire spread, large fuel burnout, smoke production, and soil heating and fire rate of spread is simulated in one dimension. Others than standard fuel models can be used and the simulation outputs graphics, tables, and charts after finishing the simulation.

fireLib

fireLib is a C library derived directly from the BEHAVE fire behaviour algorithms for predicting spread rate, intensity, flame length, and scorch height of free-burning surface fires in two dimensions. *fireLib* serves well for programmers who need a simple but highly optimised API for developing and investigating on fire behaviour simulators [13]. This simulator contains thirteen functions but only as few as four functions are required to create a simple yet efficient and functional fire simulator.
As input, the map with the initial fire front at a given time t_x is required along with input variables containing necessary information on environmental circumstances to run the simulation. These input variables comprise fuel model, fuel moisture of living fuel and dead fuel (after 1, 10 and 100 hours), wind speed and direction, slope and aspect.
The output of the simulation is a map of the simulated fire for a later time t_{x+1}. Each cell representing the total terrain holds a minute value indicating the fire arrival time with respect to the simulation start time. This value is 0 if the cell never gets burnt.
The calculation of fire spread may be represented as a pipeline through which sets of input parameters are introduced at four stages to calculate succeeding fire behaviour variables [14]. Each stage is implemented with one of the four basic functions mentioned earlier. During simulation, the methods iterate over each cell included in the map propagating the fire from one cell to another, if conditions indicate so, until reaching a terrain border. The stages accomplished for each cell can be found in table 2.1.

Stage	Stage Inputs	Stage Outputs
1: Fuel	Fuel bed and fuel particle characteristics	Characteristic fuel area, load, fuel bed bulk density, and fire residence time
2: Moisture	Fuel moisture	No-wind and no-slope spread rate, reaction intensity, heat per unit area, and live fuel extinction moisture
3: Wind	Wind speed & direction, slope & aspect	Maximum spread rate and direction of maximum spread
4: Direction	Spread azimuth	Spread rate, fire line intensity, flame length, and scorch height

Table 2.1: Simulation stages in *fireLib*. Taken from [14].

In the first stage of the pipeline, variables such as characteristic surface area, loading, and residence time are derived. These are important features of the chosen fuel model which is usually considered invariant within the time frame of a fire behaviour simulation.
Fuel moisture is introduced in the second stage of the pipeline because it is required by the Rothermel fire model to calculate the heat sink term.
The effect of slope on fire behaviour is modelled using the same mechanism as for wind. Thus, slope and aspect are introduced into the pipeline at stage three along with wind speed and wind direction in order to obtain maximum spread rate and direction of maximum spread.
In stage four, fire behaviour in any of the eight primary wind directions is determined using an elliptical growth model which leads to the computation of spread rate, fire line intensity, flame length, and scorch height.

The performance of the fire model could be significantly improved by partitioning the fire behaviour computations into these four stages. Only those stages are proceeded which are necessary to arrive at new behaviour estimates.

The following section presents the input variables required by fire simulators in more detail examining their impact on an ongoing fire, classifying them by various subjects to identify possible obstacles and introducing the sensitivity or importance of each variable.

2.4 Input Variables

The correctness of many computing tools simulating real-world occurrences strongly depends on the quality of the provided input. One cannot expect correct results if the entries fed into the system were erroneous. Unfortunately, fire behaviour simulators are no exception because many input variables are highly dynamic and difficult to measure. This uncertainty in input may lead to unreliable predictions.
Most of the problems that are treated by computational science involve a vast amount of data and a large number of variables. As mentioned in 2.2, the fire behaviour model of Rothermel, which is applied in the present work requires 17 input variables if used to its full extent [5] in order to determine the propagation of a wildland burning. The exact number of input variables necessary to predict fire behaviour depends on the applied simulator and may vary from tool to tool (see section 2.3). Most of these input variables are continuously distributed in space and many of them are correlated [5].

In the following subsections we first present the principal input variables of a fire simulator. Second, we classify the main variables according to different criteria in order to gain a better understanding of the data's nature. The last part of this section summarises some of the information available about the sensitivity of the single variables and their influence on the final prediction result. Throughout the present work the terms variable, attribute and feature are used synonymously. A set of variables then forms the complete input parameter, feature vector, or record, if database related.

2.4.1 Presentation of Input Variables

Following, typical input variables for fire behaviour simulators are presented with some of their characteristics to provide a better understanding of the fire prediction problem. Thereby, each simulator may require all of these variables or only part of them to run the simulation. Some simulators may expect further information not mentioned in this section. Input variables are assumed to be static and do not change during a running simulation which is contrary to their dynamic reality.

Fuel type

Nature may show a great variety of plants and herbal species in a specific area. The dominating type of fuel which determines the fire propagation for this area has to be identified and can then be matched with an existing fuel model. A fuel model is the mathematical description of the structure and texture of a fuel type.
Currently, thirteen stylized commonly agreed fuel models representing

a wide variety of fuel conditions are used to make fire behaviour predictions [15]. Every fuel model carries its own standardised characteristics covering the properties ratio of surface area to volume for each size class, fuel bed depth, moisture of extinction for dead fuel and three different fuel loads. Thus, the mathematical model is eased and information becomes manageable for the computer.
The fuel models are made up of variations of the components describing the vegetation and environment: needles or leaf litter, dead and down woody material, grasses and forbs, shrubs, and regeneration [12]. The thirteen fuel models are divided into four major fuel community groups consisting of grass, shrub, timber and slash. Choosing the appropriate fuel model requires plenty of experience and personal judgment. In the ongoing work, the mention of fuel model always refers to the type of fuel present in the specific area.

Fuel load

Fuel load refers to the total amount of combustible material in a defined space and is measured as the amount of available fuel per area unit (normally tonne per acre). The fire will spread faster and with an elevated intensity stumbling across an increased amount of available fuel.
This variable is not required by *fireLib* as direct input but encoded in the fuel model number.

Fuel moisture of living and dead fuel (after 1, 10 and 100 hours)

Fuel moisture is a factor which has an impact on the velocity with which the fuel is burnt. Obviously, when fuels are moist, they burn poorly and they burn well if they are dry. This is especially the case after droughts. The higher the humidity of the fuel, the slower will the fire propagate because high moisture content increases the heat required to ignite a fuel.

Wind speed and direction

Wind directly affects the intensity, direction, and rate of spread of wildland fires [12]. It provides a fresh supply of oxygen to the fire and speeds up moisture exchange between the air and the fuels, drying up wet fuel. In addition, it angles the flames to preheat fuels by radiation. Wind can be responsible for 'spotting' and is even able to cause fires to burn erratically. An elevated wind speed fastens propagation and a wind strong enough can determine the propagation direction. Additionally to these general wind characteristics, wildland fires tend to generate a

microclimate with swirls and gusts.

Temperature

Temperature, being a basic weather element and influencing other weather elements, has a direct impact on forest fires - a higher ambience temperature leads to warmer and dryer fuel which ignites faster. Differences in temperature create differences in air density and atmospheric pressure [9].
This variable is not entered into *fireLib*.

Precipitation

Precipitation, e.g. rainfall, can slow down the fire propagation. Furthermore, it is directly related to fuel moisture: small fuel (grass, shrub) absorbs the wetness faster and precipitation thus has a higher impact as on big fuel (timber, slash). But big fuel is able to maintain humidity better and longer (also see **Fuel moisture**).
fireLib does not consider precipitation because this variable belongs to a more complex atmospheric model.

Elevation, slope and aspect

Aspect, slope and elevation have a direct impact on the severity and extent of a burning. Elevation - height of a geographic location above mean sea level - and aspect - direction to which a mountain slope faces - are responsible for the quality of fuel. Varying conditions can have very substantial influences on the distribution of vegetation.
Slope - the inclination of a region measured in degree or radian - plays an important role in deciding the rate of fire spread as shown in figure 2.4.1. Fire propagates faster uphill because smoke and heat ascend and lead to a pre-heating of the close fuel. Further on, flames are closer to the fuel. That is why this critical component may determine and increase the rate of spread significantly.
For *fireLib* slope and aspect are sufficient as input variables describing the terrain.

In the course of the following sections and chapters, we will concentrate on the most important input variables describing fuel model, moisture, slope and wind which are required by *fireLib*.

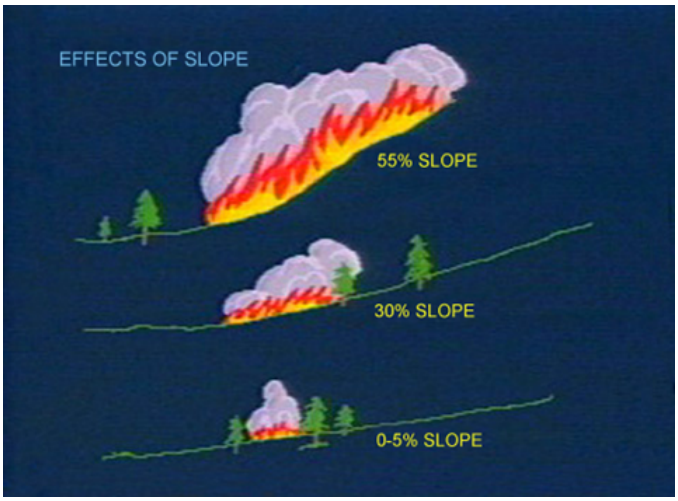

Figure 2.1: The effects of slope on the rate of spread in a forest fire. Taken from [12].

2.4.2 Classification of Input Variables

The precise analysis of input variables helps us understand their specific characteristics and may reveal particular problems that can occur when feeding the simulator with these variables. In order to identify possible difficulties, following we classify the input features by various topics.

Origin

Maybe the most obvious way to classify input variables is organising them by their origin. Fire simulators require features describing the *vegetational* conditions (e.g. type of fuel, moisture of dead fuel) of the area. Further on, the *climatologic* characteristics (e.g. wind speed and direction) of the burning region have to be captured for a specific point in time. The remaining information needed to run proper simulations covers *topographic* details (e.g. slope).

Probability of change

Another manner of categorising the input variables of fire simulators is by their probability of change. We have *static* input variables (e.g. fuel model, slope) which will not change, at most very little, in time and space during an ongoing fire. These variables are known in advance and have little adulterant effects on the prediction result. Therefore, it is less problematic to keep their values constant during a running

simulation. Although the terrain can change considerably over space, this information can be inserted in the simulator before starting the prediction process.
On the other hand, there exist *dynamic* (e.g. moisture) or *highly dynamic* (e.g. wind speed and direction) input variables which may vary significantly during the course of a fire over time and space. These variables are typically more difficult to handle. Their changed values should be considered during a running simulation in order to obtain the most reliable results. But often parameters are not measurable in the instant a fire occurs and most simulators are not capable to process dynamic variables. Simulation, in most cases, still works with a given initial set of variables and does not allow the dynamic introduction of changed variables while running.

Type

The third possibility to group input variables is by their type. We have to cope with *nominal* or symbolic features (e.g. fuel model) and *linear* (quantitative, numeric) variables (e.g. slope, wind direction).
Linear variables can be continuous or discrete and possess a (total) order, i.e. their values can be arranged in the usual sense and the ordering is relevant to the context in which they are used.
Nominal variables are made up of discrete values not necessarily in any linear order. For example, a variable representing colour might have values such as red, green, blue, brown, black and white, which could be represented by the integers 1 through 6, respectively. This just allows us to observe if values are equal, but no comparison of type smaller than or greater than possible. The values are not related in any way other than the fact that they belong to the same set [16].

2.4.3 Sensitivity Analysis

In order to get an idea of the importance of single input variables or to find out if some input variables are correlated, a sensitivity analysis should be carried out. A detailed examination of input variables will furthermore allow to determine "which variables are worth spending time on tuning and which are better to avoid spending such effort on" [17]. Knowing the results of a sensitivity analysis, we want to identify the most significant input variable among all features which, once properly adjusted, would lead to the greatest improvement of precision in the simulated fire. Finding the second, third, and so on, most important input variable permits us to establish an order of importance. This then can be used to adjust further features. The sensitivity of input variables depends on the fire propagation model used. [5]

correctly states that sensitivity analysis for the various fire behaviour models and their input variables are rarely performed. This is why we have to rely on the work fulfilled by [5, 17, 18], all of them investigating the sensitivity of input variables using the Rothermel fire behaviour model. According to this model, the direction of maximum fire spread is basically a function of wind and terrain. While [5] and [18] concentrate on particular fuel types in Switzerland and Spain, respectively, [17] opts for an approach not dependent on the fuel model.
All three works agree that only few variables contribute to the most significant part of the variance from the simulated result compared to the real spread. That is to say, that big imprecision in these variables will lead to big deviations in the result. Errors in variables with low sensitivity have a minor effect on the final result. Nevertheless, [18] recommends caution extrapolating results from local studies to more general types of vegetation or climate.

In conclusion, we can summarise that the variable with highest sensitivity to imprecision is the type of fuel ([17] originally names the variables 'compactness factor' and 'fuel loading' - information which is encoded in the fuel model). Shortly after range wind direction and speed, followed by slope and moistures.
Assuming that the correct fuel model and slope are known in advance and vary little in space, wind conditions and moistures remain as primary sources of uncertainty, the former together with the slope of the terrain primarily responsible for determining the fire spread direction and shape. Thus, it is our objective to consecutively enhance these imprecise variables.

Chapter 3

Forest Fire Prediction

3.1 From Simulation to Prediction

The simulation of forest fires is not only used for fire risk analysis and e.g. to investigate the placing of fire breaks, but in fact, simulation also helps to work out what fires will be like before they are finished or extinguished. Tools simulating forest fires are used to estimate fire behaviour and possibly save time, money, and lives [9].
The drawback of present predictions based on the used models is their missing accuracy which arises from quantifying the dynamic weather conditions. In order to fight this downside, sophisticated information systems (GIS, satellite-based terrain mapping, meteorological information systems) are used to assure a more correct data entry. This proceeding results in data and calculation intensive methods that, while more accurate, often lack real-time capabilities indispensable for successful fire fighting. Even supercomputers fail to provide timely results. In order to still use fire simulators for the prediction of fire spread, approaches are needed that deliver the most reliable results using a justifiable precision of input variables to cope with real-time capacities.

This chapter starts with a short introduction to the classical fire prediction approach 3.2 and then investigates its reliability 3.3. Afterwards, the data-driven fire prediction 3.4 is presented which tries to enhance the quality of input parameters to enable more precise predictions without intensifying calculations up to an unfeasible level.

3.2 Classical Prediction

The classic way of predicting fire behaviour takes the initial state of the fire front (RF = real fire) as input and the variables that describe the ambience in which the fire takes place. These parameters are given for some time t_x and entered into any existing fire simulator (Fire Sim). The simulator then returns the prediction (SF = simulated fire) for the state of fire front at a later time t_{x+1}. Figure 3.1 summarises this process.

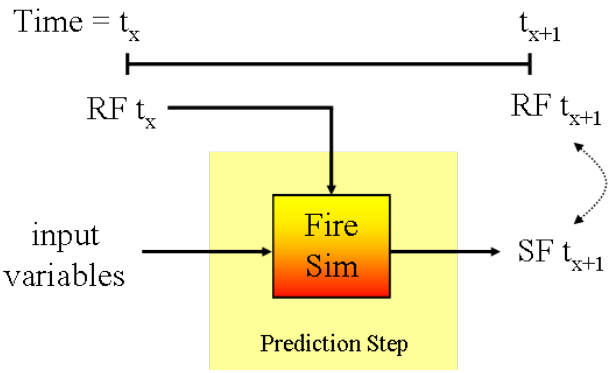

Figure 3.1: Classical prediction.

Depending on the complexity of the chosen simulator, this method consumes rather little computing resources. But comparing the simulation result SF from time t_{x+1} with the advanced real fire RF at the same instant, the forecasted fire front tends to differ from the real fire line. This is principally due to the major problem of the classical prediction: The calculation of the simulation is based upon one single set of variables [8] - the input parameter or input feature vector. Thus, the output of simulation heavily depends on the quality of this specific input parameter.
The next section shows that input variables for fire prediction normally are tainted with imprecision and therefore have to lead to unreliable results for the classical prediction method.

3.3 Prediction Reliability

The results of the classic way of forest fire prediction often are unsatisfactory because the prediction is missing precision and does not reach a sufficient degree of accuracy. This incorrect output makes classic prediction results unreliable. There are multiple reasons causing the lack of acceptable output.
In [17] Abdalhaq mentions computational limitations as possible error sources. Approximation or truncation errors while solving the equations of the mathematical model lead to decreased precision. Also processor limitations and the computers'

missing infinite precision in storing numbers provoke inaccuracy.
Further on, there can be observed sources of defect in the individual simulators and their underlying models. The used mathematical equations to model the progress of the fire simplify a complex process without disregarding the main characteristics. In [17] it was also noted that, on the one hand, the behaviour of dynamic variables like wind is eased too much to be able to deliver correct results. Dynamic input values are also assumed to be constant for the time period of the calculation. But on the other hand, a more detailed model could probably not justify the elevated computational effort.
Another reason concerns the fuel model applied in the simulator. Most simulators are homogeneous, i.e. they apply the same fuel model to the complete area in simulation process which, in many cases, does not reflect real circumstances. Other fire simulators, like fireLib, are heterogeneous and are capable of employing different fuel models to different regions during one simulation. Only few of the available fire simulators (e.g. FARSITE) are able to take into account barriers and fire brakes (like streets, lakes...) that can stop or slow fire spread and, consequentially, cause variations in the shape of the fire line.
The cell form chosen to partition the affected terrain also influences the final result. The majority of simulators uses quadratic cells (also *fireLib*) which limits fire propagation possibilities to the eight main wind directions representing the eight neighbours of each cell. The application of hexagonal cells was investigated in [19] with promising results.
But even more important is the size of the used cells while calculating the fire propagation. Denham remarks in [8] that bigger cells force the use of estimated average values for the complete region represented by the cell and thus contribute to the error-proneness of the prediction. The use of smaller cells increases complexity and the computational effort to an unjustifiable level.

Probably the most important and most mentioned error source deals with the imprecision and uncertainty of input variables. Uncertainty can be considered as the absence of information that may or not be obtainable [5]. Uncertainties in input variables can have a substantial impact on the prediction result.
As explained in 2.4, some information needed to obtain a reliable prediction result is highly dynamic. Above all, this is true for wind conditions. Not only do they tend to vary from cell to cell, but also change frequently within short periods of time in one and the same cell. Even if the wind was modelled to a satisfactory extent, it would be difficult to measure and therefore introduces uncertainty. Until now there exists very few approaches trying to link a fire simulator and a meteorological information system [20] what would, in part, remove this uncertainty.
Although if input variables were measurable with increased frequency, most simulators are not qualified to consider their changed values during a running simulation because they function with a static set of initial variables.
Some input variables might not only be imprecise, but unavailable. This is, amongst

other reasons, due to the non-existence of fire-resistant sensors which could measure input variables like moisture. Thus, in the moment of prediction, the correct values for these variables are not available and in many cases outdated incorrect values from previous measurements have to be applied. The unavailability of input values also emerges from the fact that not all values can be obtained for the complete terrain, but just for parts of it. This leads to weather data being too local for a large fire, or too remote for predictions at a localised situation [9]. Values for missing regions often are interpolated from existing values [8], inserting further inaccuracy.

Having examined the many possible error sources which make classic fire prediction little reliable, one could assume that there are many starting points to enhance the prediction result. But instead of changing well-established fire behaviour models or inventing another simulator, a much more general and promising approach is introduced in the next section. The so-called data-driven prediction tries to remove uncertainty from input variables which presents the principal source of defect for fire prediction.

3.4 Data-Driven Prediction

The classical prediction would be correct for the given set of input variables but because these do not reflect real conditions, the fire front prediction is not correct in reality [8]. This is why another method was developed which refines the set of input variables. The closer the actual circumstances are to the model assumptions, the better the predictions will be [12].

In addition to the single prediction step of the classical method, the data-driven prediction introduces a previous calibration step in order to syntonise input variables. This pre-processing of simulator input makes use of optimisation techniques with the purpose of calibrating the input parameter [10]. The objective is to find a set of input values that, if they feed the simulator, would describe best previous behaviour. It is argued that the same set of values could also be used to describe best the immediate future. This supplementary stage during prediction can be implemented independently of the used simulator and therefore offers a general solution.
Obviously, data-driven prediction results more time-consuming than the classical prediction. This is why we make use of parallel or distributed computation in order to minimise the overall computing time.
Using the data-driven method we can achieve that the prediction does not depend anymore on one single input parameter, like the classical prediction, but is the result of a series of adjusted input parameters. How exactly these adjusted input

parameters are generated, selected, and applied, is explained in detail in the next chapter.

Chapter 4

Data-Driven Prediction

In real circumstances, fire fighters have a lot of experience and use their knowledge of observed fire behaviour to rectify results from automated fire behaviour prediction systems when forecasting the spread of new forest fires. The same idea of using "experience" is applied in the data-driven fire prediction: using knowledge from the direct past to correct input variables for the immediate future in order to get improved prediction results.
This chapter gives a deeper insight into the data-driven prediction. First, the general basics of the method are explained. Next, with more detail, we resume how knowledge is introduced calibrating input parameters with evolutionary optimisation techniques. The last section describes the additional usage of domain knowledge to reach a faster convergence of the evolutionary algorithm towards fitter solutions.

4.1 Overview

The present work deploys the data-driven fire prediction method with its two stages *calibration* and *prediction* to forecast the fire front. In addition to the one-step prediction of the classical method, an auxiliary adjustment step is introduced previous to the prediction step which tries to calibrate the input variables in order to get better performance. In fact, the same simulation as in the classical prediction method is used but enriched with methods and resources from computational science. Different combinations of input variable values are generated, syntonised and selected, using a Genetic Algorithm (GA) from the field of evolutionary computing as an optimisation technique, in order to find the best fitting input parameter. The overall prediction result thus will be refined step by step using an optimised adjusted input parameter during each prediction step. Figure 4.1 illustrates a schematic figure of the enhanced method.

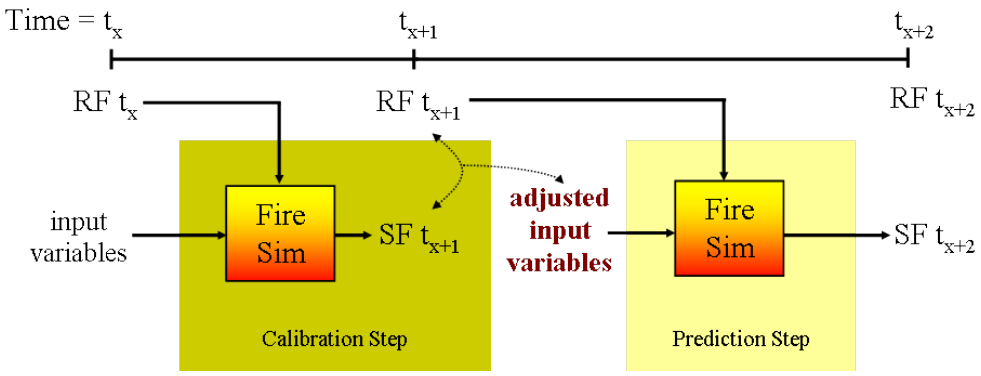

Figure 4.1: Data-driven prediction.

Following, the two stages of the data-driven prediction method are explained in more detail.

Calibration Step

This stage tries to detect improved input variables for the subsequent prediction step. It determines variable values which, entered into the simulator, would have produced a good prediction from time t_x to t_{x+1}. To obtain the enhanced input values, the fire is first simulated with the given input variables and the given fire front (RF) in t_x. The simulator (Fire Sim) returns the simulated fire front (SF) in time t_{x+1} which is then compared to the real fire front in t_{x+1}. In doing so, the quality of the prediction can be evaluated. This information is used to select better values for the dynamic input variables and to use them as input for the simulator in the prediction step.
Using a Genetic Algorithm, various feature vectors are composed and evaluated until getting one (or more) parameters which yield good predictions. Executing a large number of simulations with many individuals eliminates the exclusive dependence of the prediction result from one input parameter.

Prediction Step

Assuming that environmental conditions and thus variable values do not change significantly, the improved input variables are then used in time t_{x+1} to get an improved prediction for time t_{x+2}. After finishing the calibration step, the best fitting parameter with respect to the simulation result (SF) in time t_{x+1} is chosen as input parameter for the simulator together with the real fire front (RF) in time t_{x+1} in order to predict the evolution of the fire for time t_{x+2}.

Although we introduce additional uncertainty to some minor extent using variable values from the past, the overall process of the improved fire prediction method yields more reliability compared to the classical prediction as already shown in [17] and [8].
The data-driven fire prediction is completely simulator-independent and can be used with every available fire simulator providing the speed and direction of fire propagation as output and taking the presented principal variables as input. The simulator itself and therefore the process of fire propagation are treated as a black box, a common approach in computational science.
The next section gives a short introduction to Genetic Algorithms and how such optimisation technique can be successfully applied to generate valuable input parameters.

4.2 Genetic Algorithm to Calibrate Input Variables

"In fire behaviour modelling much more emphasis should be put on the assessment of input data and results." [5] This is not only true for the elaboration of models but the same holds for their application and the evaluation of input variables at prediction time. As described in 3.3, many input variables are not available at the moment of prediction. This is why estimated values from the past or interpolated values from neighbouring regions afflicted with imprecision have to be used. In consequence, this imprecision continues and leads to unreliable prediction results. We now show how to remove part of this introduced imprecision applying a Genetic Algorithm (GA) to find an improved set of input values during calibration stage of the data-driven prediction.

Genetic Algorithms are a heuristic optimisation method and used to find exact or approximate solutions to search problems. Their functionality is inspired by evolutionary biology. For a GA to work, first of all the solution domain has to be represented genetically. The standard encoding is an array of bits but other representations are possible. We use a straightforward real-valued array to symbolise the individuals of the solution space in their "natural" manner. One individual stands for one fire configuration and includes the values for fuel model, slope, moisture of living fuel, moisture of dead fuel after 1, 10 and 100 hours, wind direction and speed. But instead of probing single solutions, a GA operates on populations of individuals and therefore gives a lot of possibilities for parallelisation. Furthermore, a fitness function is needed to evaluate the possible solutions and measure their quality. The fitness function is always problem dependent and defined over the genetic representation. In order to be able to apply the principle of the 'survival of the fittest' we have to detect the fittest solution, i.e. most suitable

fire configuration. In this context, the fittest solution is the one that generates a simulated map the most similar to the real map of fire propagation in time t_{x+1}. Thus, to determine the fitness of each solution, an error function based on a cell-by-cell comparison is applied [8] putting into relation the erroneous burnt cells of the simulated fire with all burnt cells. Finally, the GA tries to find the fittest individual with a minimised error value which then is considered as the final adjusted parameter.
Once we dispose of the genetic representation of fire configuration and have defined the fitness function, an initial population is created randomly. The GA now repetitively applies the methods known from nature: elitism, selection, crossover, mutation, and reinsertion (see figure 4.2). It runs for various iterations, also referred to as generations, until reaching a pre-defined maximum number of generations or a satisfactory fitness level. In each generation the fitness of every individual is calculated whose value influences the next operations.

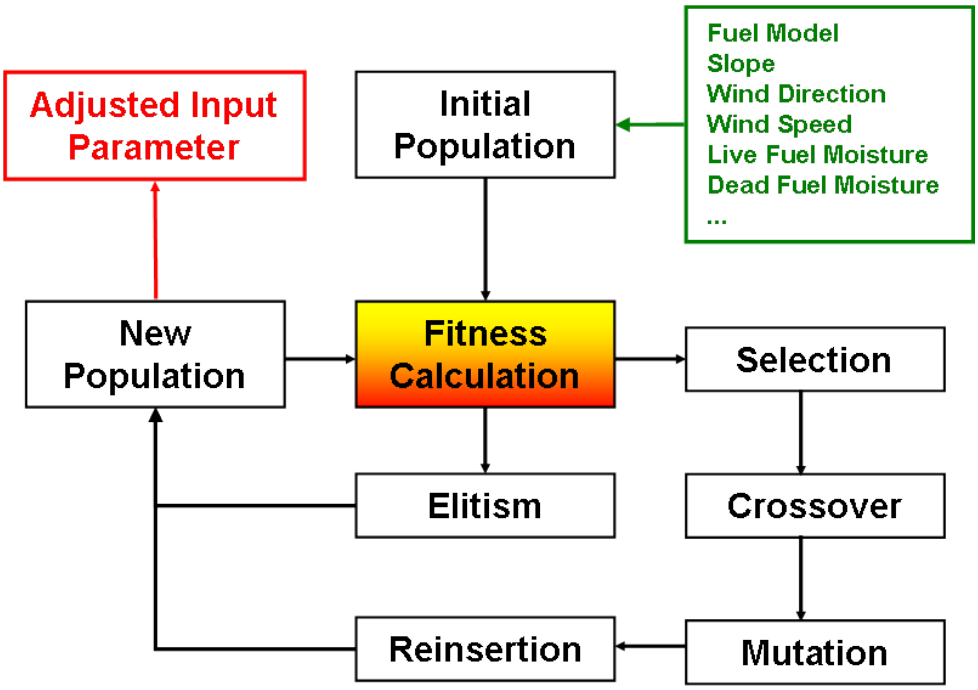

Figure 4.2: Functionality of the Genetic Algorithm to calibrate the input variables.

Genetic Algorithms can be an effective search tool in many application fields but they possess many parameters which are difficult to tune and fine-tune and all of them affect the performance of the algorithm [17]. In order to obtain the detailed configuration settings for population size, mutation probability, crossover method etc., refer to [8]. To continue, the fitness evaluation of each individual is the most time consuming procedure and affects speed and efficiency of the overall algorithm. For every solution a simulation has to be run to compute the fitness of this solution. In order to prevent the GA from becoming a bottleneck for the application, it is

required that it finds a good solution fast. This is especially true for real-time predictions which have to finish quickly and deliver highly reliable results. Thus, the fire simulations needed during the execution of the GA should be reduced to a minimum. To speed up the GA and to reduce the vast search space representing a huge pool of possible solutions, several approaches were investigated by [17]. The most promising tries to guide the GA towards fitter solutions cutting ranges of specific variables using domain knowledge. In doing so, the search space is reduced and the number of iterations needed to find an acceptable solution can be minimised.

4.3 Using Domain Knowledge to Guide the Genetic Algorithm

In the Genetic Algorithm research community there exist two different opinions about to steer or not to steer GA for a variety of purposes. Some argue that the GA should be programmed the most general possible in order to be able to find a solution for a wide range of problems. But for certain problems it proved of value in recent years to guide the GA by introducing expert knowledge [21, 22, 23] and thus obtaining a 'good' solution faster. Particularly directed mutation techniques showed significant potential and garnered promising results [23]. This is why current investigation tries to generalise the introduction of domain knowledge for a variety of problems yielding similar results.

The data-driven fire prediction uses a GA in the calibration step (see section 4.2) which, due to its time consuming fitness evaluation, can become a critical part of the complete prediction application. To avoid high solution variances and long convergence times as well as to reduce computational expenses, [17] proposed three techniques to reduce the search space of the GA.
Firstly, by fixing some variables to their nominal values problem dimensionality can be decreased. This is applied in the case of the static variables fuel model and slope because their values are known in advance and thus can be locked. They do not have to be considered in the search space during the GA.
Secondly, a search in reduced ranges, possible by introducing a certain degree of knowledge, can tune the overall performance of the GA. It results the most effective, limiting the range of the most sensitive variables (wind speed and direction) around a particular value.
Thirdly, [17] suggests sampling the search space, i.e. selecting some discrete values from the range of possible values of the variables used by the GA. By limiting the variable value possibilities a faster convergence is expected but the optimal solution may never be found. Therefore we do not consider this proposition.
All three techniques are based on the sensitivity analysis introduced in 2.4.3. Static variables are fixed to their respective values, dynamic but highly sensitive

variables with a dominating impact on the prediction result will be limited in their range. Applying these steering techniques, we are able to maintain a high number of variables and the most complete data possible needed to obtain good individuals during the GA. At the same time, the search space is reduced decreasing the number of necessary iterations and fitness evaluations. This, finally, leads to improved solution robustness and solution costs.

In order to accomplish the search in reduced variable ranges we need to dispose of expert knowledge. The detailed description of generation, storage and retrieval of domain knowledge is delayed to chapter 5. We for now simply assume that domain knowledge is available and try to evaluate in which operations of the GA specific knowledge can be introduced and used meaningful. In [24] the author proposes to use expert knowledge in nearly all stages of the GA: population initialisation, recombination, mutation, selection, and reproduction. Thus, we could eliminate certain randomness of the algorithm and force the operations to assign good values to get simulations close to reality.
Undoubtedly, population diversity is a key issue in the performance of evolutionary algorithms [21]. It is important to avoid premature convergence and to escape local optima. Population diversity is mainly controlled by means of mutation. A higher mutation rate leads to increased diversity among individuals. But on the other hand, mutation sometimes is considered to be a random walk through the search space, especially if the rate is chosen to high. Though, this is only valid when mutation is applied uniformly over the population [25]. Therefore, mutation seems to be the most obvious operation of the GA where expert knowledge can help to remove randomness. Some recent works [21, 22, 23, 26, 27, 28] recommend knowledge-guided mutation, also referred to as knowledge-driven, data-driven or information-guided mutation and, at the same time, encourage the use of an increased mutation rate (above 0.5). Thus, the overall GA can successfully select attributes from large and high-dimensional datasets.

According to sensitivity analysis in 2.4.3, the dynamic variables with the biggest influence on the fire spread are wind direction and speed. Rothermel's mathematical model (see section 2.2) joins slope and wind to determine the propagation direction. This is the base for the computational method proposed by [8] to adjust the values of the wind conditions to obtain better predictions. Wind, perhaps, is the most variable feature and the most difficult element of weather to predict. Using the computational method at time t_{x+1} the maximum fire propagation direction and speed from the map of real fire are calculated. These are then taken as input parameters, together with fuel model and slope, to query knowledge base realising a "reverse search". During this search the most similar fire configuration with respect to the given configuration is found extracting its causing wind conditions. Thus, we are able to obtain the ideal speed and direction of wind to get a similar propagation compared to the real in the subsequent prediction step.

Once we dispose of the proper wind values, this information then can be used to guide the GA. Wind direction and wind speed are not treated as fixed values like done with slope and fuel model, but the returned values are used as the centre of a new range much smaller than the original range. The current configuration works with +/-5 deg for wind direction and +/- 2 mph for wind speed. We cannot use the total value but have to choose a valid subset of the original range due to approximation errors. The computational method only returns values which are most similar to the real fire. Depending on the degree of detailedness of the information in the knowledge base, it is possible that the configuration of the real fire is found or not. The fewer values stored in the database, the smaller the chance to return wind conditions similar to real fire behaviour. Wind values now oscillate in the shortened range during mutation. Cutting ranges thus helps the GA to converge faster towards better solutions.
The advantages of the computational approach are its portability and independence. Treating the simulator as a black box, this method is applicable for each simulator just by substituting this black box. Also, this method can be generalised for further environmental applications as flood disasters.

But domain knowledge cannot exclusively be used during mutation. It can equally be of value during the operations selection and elitism or during reinsertion to make proper use of mutated individuals. And not only wind conditions can be used to guide the GA, but there are other dynamic input variables, e.g. moistures, that are not precisely measurable at prediction time and whose guidance therefore could be useful. However, in the case of knowledge-guided mutation, steering all variables of an individual will result little efficient. The question here is how much randomness can be stolen from mutation before it stops being a random operation with its intended purposes?
Further enhancements, which remain to be investigated, include interchanging the stages of mutation and reinsertion (see figure 4.2). Knowing that mutation produces valuable individuals, this modification allows them to survive and undergo the next generation. We could only allow mutations which increase the fitness of the corresponding individual, establish a self-adapting mutation rate, or mutate individuals with the highest (lowest) fitness. In order to preserve sufficient randomness during mutation and not to forfeit population diversity we should try to work with decreasing ranges of guided variables from generation to generation. A light guidance allows highly diverse individuals at the beginning in order not to get stuck in local optima too early. The guidance is tightened towards the end of the GA to get a fine-tuning. Instead, a hybrid approach alternating guided mutation with generations without guidance could also result beneficial. Clearly, the main effort should point to reducing the time consuming fitness evaluation, a critical factor for real-world applications. A word of caution is recommended for all proposed GA enhancements: Some improvements may only pay off for Genetic Algorithms with large populations and an elevated generation number, but create additional

overhead otherwise. In the case of calibrating input variables for fire propagation, [8] showed that already after five iterations considerably good solutions were found and could not be improved significantly in subsequent generations.

We have seen how knowledge can be incorporated in the different phases of the GA. Expert knowledge, obtained from observed fire behaviour data to guide the Genetic Algorithm, is retrieved from a knowledge base once in every calibration step before the initial population is generated. It is stored temporally and thus is available during each iteration of the GA. The next chapter presents the former process of obtaining the knowledge from the knowledge base and, in the second part, introduces a novel, faster and more robust approach.

Chapter 5

Knowledge Retrieval

This chapter gives an overview on the retrieval of knowledge incorporated into the GA to guide it towards better solutions and, at the same time, to achieve a faster convergence.
First, the general design and organisation of the former text database is explained, followed by a short introduction to the former naïve retrieval method and an extensive evaluation of the approach.
Then, our enhanced automated retrieval approach is described in detail. The theoretical principles are followed by implementational information proving that a faster method was implemented which occupies less storage and is independent from parameter order. This is the more important as with growing accumulated knowledge its storage and retrieval could turn into a bottleneck for the application.

5.1 Naïve Retrieval

In this section the preliminary work of [8] is summarised which serves as a starting point to establish a more robust and faster knowledge retrieval method. Analysing and understanding the specific drawbacks of the simple approach helps us to implement a more sophisticated technique.

5.1.1 Design and Organisation of Knowledge

This subsection comprises how the information, which is used to extract domain knowledge during the GA, is generated and in which manner it is stored.
To be able to dispose of expert knowledge which results useful to speed up the Genetic Algorithm, a text database containing different fire configurations was generated [8]. These configurations consist of various values for wind, slope and

fuel model which were simulated using fireLib in order to archive the resulting fire propagation results together with their causing configuration variables in a persistent manner. The exact variables of fire evolution included in the sample data are: fuel model, slope, fire direction, fire speed, fire distance, wind direction and wind speed.
Thus, it becomes feasible to do a "reverse search", i.e. given the fire propagation, it is possible to obtain the causing wind values with a certain fuel model and with a particular slope.
The generated sample data is independent from the type of simulation or simulator, but a special data format is required and the data must be available in the correct measuring unit (see table 5.1). The information can be synthetical data which was created with any arbitrary fire simulator generating speed and direction of fire propagation as output and requiring at least fuel model, slope and wind as simulation input. The information could also originate from real forest fires being included as knowledge in the text database, maybe applying a prior correction step to adjust measuring units.

The sample data serving as knowledge base in the naïve retrieval approach was stored in flat text files organised in a tabular style. Thereby, for each fuel model one text file was used, also referred to as 'table', to reduce the needed main memory while accessing the data. This was possible because fuel model is a static input parameter and does not change during a running simulation and therefore only the file for one fuel model has to be loaded into main memory during each calibration step.
The information gathered in the tables is not complete due to the vast amount of possible combinations. In fact, it is impossible to obtain a complete set of data, because most variables are continuous real values which are known to be infinite. Thus, data had to be discretised to some extent selecting a granularity or interval size for each continuous variable. Logically, the finer the chosen data granularity, the more complete results the information. But bear in mind that a finer data granularity also implies a considerably augmented quantity of data which causes elevated retrieval times and might become a bottleneck of the application.
Table 5.1 shows the current discretisation settings applied to the variables which were used to obtain the respective fire configurations. Each of the 13 fuel models was simulated with a particular slope (out of eleven discretised values), a given wind direction (out of eight discretised values representing the eight primary wind directions used in fire simulators which use quadratic cells to divide the area) and a specific wind speed (out of eleven discretised values). This rather coarse discretisation already results in nearly 1,000 data lines for each fuel model.

The minimum and maximum values for each feature determining the range are clear in the case of fuel model and wind direction. In the case of slope, $0.785 rad$ are equivalent to 45°. Above this value with a still more elevated slope the fire propagates the

Variable	Fuel model	Slope	Fire dir.	Fire speed	Fire dist.	Wind dir.	Wind speed
Unit	-	rad	deg °	ft/min	ft	deg °	mph
Range	1-13	0-1.1	-	-	-	0-360	0-30
Granularity	1	0.11	-	-	-	45	3
Cardinality	13	11	-	-	-	8	11

Table 5.1: For each variable of the current synthetic parameters the measuring unit and range (minimum and maximum value) is given along with its granularity, i.e. degree of discretisation and cardinality (number of possible discretised values).

same and the effect of an increased slope does not accelerate fire propagation any further. The values up to $1.1rad$ can be understood as a buffer. Regarding the wind speed, the same phenomenon occurs. Above a certain threshold the wind speed has no further influence. In [17] an upper limit of $20kmh$ ($12.43mph$) was proposed during sensitivity analysis. The values up to $30mph$, again, are considered as a buffer. These observations are typically obtained from field and lab measurements.
But also the discretised set of data does not necessarily need to be complete as it may contain infeasible parameters, i.e. value settings that are impossible to occur in real forest fires, even under extreme conditions, and which therefore might be discarded to free storage space. This is even more the case because we use a large buffer. But for the sake of simplicity we kept all generated data in the text database without discarding any configuration.
Furthermore, the synthetical data from simulated fires might be a possible error source because of the imprecision of the simulation itself and thus the use of synthetical data as knowledge may introduce further uncertainty into the prediction. Obviously, historical data from real forest fires would be of most value (assured the correct measuring unit or applying a previous adjustment step before incorporating it into the knowledge base) to support and complete the synthetical data, but is difficult to obtain.
By contrast, the exclusive use of real data is problematical and not recommended as it requires an immense variety of different forest fires to cover all possible situations, i.e. configurations, and thus to be able to use the knowledge in a general context. Too few example parameters with little variance could even guide the GA into a wrong direction. If the 'most similar' parameter in fact is 'far away' and used to cut ranges, the correct values could disappear and be no longer available during mutation. Inappropriate individuals would be introduced into the population and, depending on the mutation rate, could slow down convergence speed to a greater or lesser extent. If the solely use of data from real forest fires is aspired, some additional quality assurance measure or system has to be employed to guarantee a satisfactory level of data diversity.

Finally, the terms range, granularity and cardinality only apply for simulated or synthetical data. Real data from historical fires may have very different own bounds for each example fire, no fixed granularity and therefore no special cardinality which makes it difficult to estimate the number of configurations, storage space and retrieval times.

5.1.2 Retrieval of Knowledge

Having explained the structure of the former text-based knowledge base, we now concretise how the information to guide the GA was retrieved using a simple approach.
Given fuel model, slope and direction and speed of the maximum fire propagation from the simulation results in step t_{x+1}, a simple search method tries to find the most similar fire behaviour from the text file with minimum difference in direction and speed of the fire propagation between the given parameter and the stored data considering fuel model and slope. To do so, the corresponding table containing the data from the matching fuel model is loaded into main memory and parsed. Then it is scanned line by line until the correct slope value is reached. If the exact slope value is not present, records from the next similar are evaluated. For all entries with the correct slope value fire propagation direction is looked at keeping track of the most similar found so far. If various records with the same propagation direction exist, finally the one with the minimum propagation speed difference is returned as result of the search.

5.1.3 Evaluation

As we can observe, this search method heavily relies on an ordering of records to be effective and to yield a good performance. Retrieval times depend on the slope used in the simulation - the higher the slope value, the longer lasts the search if data is stored in ascending order with respect to slope. The reliance on record order will turn out to be a problem on the incorporation of real data from historical burnings because data has to be introduced preserving the existing order. Avoiding a strict ordering to simplify data introduction, results in the examination of all stored configurations to find the one with the minimum difference which leads to an increased retrieval time. Neither of the two drawbacks is desired and can be easily eliminated using the database knowledge retrieval method which is explained in the next section.
But the maintenance of order or the elevated retrieval times are not the only disadvantages of the text-based approach which can be solved applying a database. There exist a few more obstacles which can be overcome using a database management system.

To begin with, the data stored in the text files is missing a column labelling system which thus makes it error-prone upon data retrieval and data changes. Further on, a parser is needed to process the data. Data changes, e.g. a changed column order, will most likely lead to parser adaptations which are an unnecessary effort. Also, the upgrade to an advanced file type with improved data retrieval abilities, e.g. .xml, causes the introduction of a completely new parser.
With a refinement of data granularity the flat text file would grow significantly which proportionally increases retrieval time and leads to an extended use of main memory, because the complete text file has to be loaded into main memory upon access. To cope with this drawback the data was split up into various tables (one for each fuel model) in order to save main memory and to reduce access time. Upon changing, updating, deleting or inserting data, this may become an error source because the data is not bundled in one place. Another limitation which unnecessarily boosts storage occupation is the lack of an automated strategy to detect unfeasible values which are unlikely to happen in real forest fires, even under extreme conditions.
Finally and very important, no data integrity can be assured if multiple users or connections access the text file; a situation that is very likely to happen in a parallel programming environment and which could result in data incoherence and data inconsistency.
In summary, the entire system turns out to be rather rigid and susceptible to the smallest changes, which is contradictory to the overall idea of the advanced fire propagation method to be of general use with any fire simulator without big adjustments. This is why a new knowledge retrieval approach was developed whose theoretical principles are explained in the next section before moving on to its implementational aspects.

5.2 Database Retrieval

In this section the enhanced knowledge retrieval approach using a database management system is explained. The approach is based on a nearest neighbour search (NNS) implemented using an appropriate distance function.
The first part covers the theoretical basics of the approach and elaborates a distance function fitting the application purposes. Afterwards, the implementational aspects are discussed. This includes a summary of the new method containing its advantages and some problems occurred.

5.2.1 Theoretical Principles

This subsection introduces the theoretical principles of knowledge retrieval from the fire configuration knowledge base improving the naïve search approach presented

in 5.1.2. As stated in 4.2, to guide the genetic algorithm during the calibration stage of the enhanced forest fire propagation prediction, we try to find the wind conditions that would have caused the real spread. These values then are used to reduce the ranges of two dimensions of the search space explored by the GA, thus reducing the overall search space.
In order to do so, we need to retrieve one (or more) fire configuration records from our knowledge base to extract its values of wind direction and wind speed. As already mentioned in 5.1.2, to start the search process the values of fuel model, slope, fire direction and fire speed of the current propagation are needed as input parameter. These fire configuration settings then are used to extract the most similar configuration from the knowledge base with respect to the described input parameter and to output its wind conditions to be used for GA guidance.

But what does the term most similar in the context of fire configurations mean and how can we measure the similarity between configuration records?
First of all, we have to express similarity in a formal mathematical or computational context. Similarity is the quantity that reflects the strength of a relationship between two features [29]. Similarity s_{ij} between two features i and j is often given in a normalised interval ranging from 0 to 1 and is quite difficult to measure. In many cases it results easier to calculate the dissimilarity d_{ij} of two features i and j which measures the discrepancy between these features. Similarity s_{ij} then can be simply determined as

$$s_{ij} = 1 - d_{ij} \tag{5.1}$$

if bound between 0 and 1. Dissimilarity of two objects is modelled applying a distance function which returns the space between the two objects.
A distance function, also called proximity measure, has to fulfil the following three constraints:

1. $d(x, y) \geq 0$ (non-negativity)
2. $d(x, y) = 0$, if and only if $x = y$ (identity of indiscernibles)
3. $d(x, y) = d(y, x)$ (symmetry).

Sometimes, strict positiveness $d(x, y) > 0$ is required instead of the weaker first condition. If the distance function furthermore satisfies the triangle inequality

4. $d(x, z) \leq d(x, y) + d(y, z)$,

it is called a metric.

Together with a set M, a metric d forms a metric space which is an ordered pair (M, d) and where d can be formalised as the function

$$d : M \times M \rightarrow \Re \tag{5.2}$$

defining a notion of distance between the elements of M and satisfying the four before mentioned criteria.

There exists a variety of distance functions. Ranking from the commonly used geometric approaches like the L_p-metrics (also called Minkowski metrics) including Manhattan distance ($p = 1$) and Euclidean distance ($p = 2$), in statistics Mahalanobis, Camberra and Chebychev metrics are the preferred proximity measures. Although there have been proposed many distance functions, by far the most commonly used is the Euclidean distance measure, which is defined as:

$$Eucl\_dist(\vec{x}, \vec{y}) = \sqrt{\sum_{k=1}^{d} (x_k - y_k)^2} \tag{5.3}$$

Having explained similarity and distance, we can now proceed formalising our problem of finding the most similar configuration record in the knowledge base to a given input parameter. We would like to retrieve a configuration x from the knowledge base D such that the distance between this configuration and the input parameter q, also called query point, is minimal compared to the distances between all other configurations and q. This can be expressed mathematically using the following formula:

$$x = \{a \in D | \forall b \in D, a \neq b : dist(a, q) \leq dist(b, q)\} \tag{5.4}$$

In other words, we want to find the nearest neighbour to the query point q in our knowledge base D.

Nearest Neighbour Search (NNS), also known as proximity search, similarity search or closest point search, is an optimisation problem that arises in a number of applications, many of them situated in a high-dimensional environment. The areas where NNS is deployed comprise statistics, data compression, textual and multimedia information retrieval, clustering, and pattern recognition and it is particularly used for classification problems in machine learning. As stated in formula (5.4), NNS requires a distance function $dist(x, y)$ to measure the differences among items in the data set and then to retrieve the closest or the k closest items to the query point with respect to this function.

In our fire configuration application, the closest data point to the query point has to fulfil several conditions which can be summarised as follows:

1. **Fuel model:** feature values of query point and data point have to be equal.
2. **Slope:** value of data point should be the most similar to query point value.
3. **Fire direction:** value of data point should be the most similar to query point value, possibly including a configurable margin.
4. **Fire speed:** value of data point should be the most similar to query point value.

These conditions not simply have to be met but also represent a certain order of importance, i.e. the fulfilment of requirement with importance number $i = 1$ is more significant than the fulfilment of requirement with importance number $i = 2, 3, ..., d$ where d is the number of dimensions of the search space queried with $d \geq i > 0$. The mentioned conditions, including their ordering, were established by [8], follow intuition, and can be easily proven reasonable applying the results of the sensitivity analysis presented in 2.4.3. It shows that the variables most sensitive to the simulator output are fuel model and wind conditions, followed by slope and moistures. Because we want to retrieve wind conditions from the knowledge base, they do not appear in the importance order but are replaced with fire propagation direction and fire propagation speed thus allowing to accomplish the 'reverse search'.
Generalising, we have to perform a *k-NNS* in a d-dimensional space where we set $k = 1$ and $d = 4$, further to including a designated priority for each dimension.

The next step consists of finding or defining a metric which complies with the established conditions. According to [30], the choice of a correct distance function is very important, because an unsuitable function may return false results and/or degrade performance.
Trying to apply the normal Euclidean distance measure (5.3) in order to retrieve the most similar fire configuration from the knowledge base fails. Alas, the pure Euclidean metric is no panacea and suffers some major drawbacks.
A profound analysis of variables (dimensions) reveals three particular problems the Euclidean distance is not able to handle. The fire configuration features spanning the search space present different ranges, different degrees of importance and comprise different variable types.

How we can deal with these problems and find adaptations to amplify the standard Euclidean metric using normalisation, applying weights and introducing a heterogeneous distance measure, respectively, is shown in the following.

Normalisation

The Euclidean distance measure treats each dimension independently and different dimensions with distinct ranges have dissimilar effects on the calculated distance. A pure Euclidean distance is not suitable for our kind of similarity search, since it is isotropic and the problem is not: Not every feature may have similar behaviours, i.e. no uniform distribution of all variables can be supposed, especially not for data from historical fires.
For example, if the values taken by the first feature x_1 over the data images are very concentrated around 0, and x_2 takes uniform values on an much larger interval, then a big difference between $x_2(a)$ and $x_2(b)$ is much more significant than the same big difference between $x_1(a)$ and $x_1(b)$. The Euclidean distance does not take into account this possible asymmetry. The weakness of the basic Euclidean distance function thus is that, if one of the input variables has a relatively large range, then it can overpower the other features.
In our case, the variable fire speed in the generated synthetic data has too much impact on the retrieval result because its generated values during example propagations are very widespread from 0 to 780 whereas the feature slope just ranks from 0 to 1.1. This conflicts with the above established importance order where the feature fire speed should have the lowest impact on the distance calculation. This is why we need to normalise variable value ranges in order to establish accuracy and improve efficiency of the NNS.
Normalisation is the process of mapping values spanning a specific range to another interval, mostly [0...1], in order to result comparable. Normalisation may introduce imprecision by the means of rounding errors.
The suggestion of just normalising the computed distance (e.g. divide each distance by the maximum of all calculated distances) to make it comparable among all other distances does not result sufficient. This process does not affect the variable values and therefore does not eliminate the just described Euclidean distance measure drawback. In our case it results more beneficial to apply variable normalisation before calculating the distance.
The process of normalisation may introduce various distortions or biases into the data. Therefore, the properties and possible weaknesses of the used normalisation method must be understood. The probably most common normalisation technique is called linear scaling transformation (min-max or range normalisation) which does not alter the distribution

of variables and is calculated as follows:

$$x' = \frac{x - min}{max - min}(max' - min') - min' \tag{5.5}$$

with x' being the normalised value of x, min the lower bound and max the upper bound of feature x, and min' the lower bound and max' the upper bound of the new interval. If the value of variable x is mapped in the interval [0...1], formula (5.5) simplifies to

$$x' = \frac{x - min}{max - min} \tag{5.6}$$

This and further normalisation methods (z-score, decimal scaling) are evaluated in detail in [31] and [32]. The enhanced Euclidean metric including range normalisation thus changes to

$$Norm\_Eucl\_dist(\vec{x}, \vec{y}) = \sqrt{\sum_{k=1}^{d} \left(\frac{x_k - y_k}{max_k - min_k} \right)^2} \tag{5.7}$$

with max_k being the biggest value of feature k, min_k being the minimum value of feature k, and d being the number of dimensions.

One arising problem is how to treat extreme values. So-called outliers have a great impact on the contribution of an attribute. A relatively robust alternative in the presence of outliers is to divide the variable values by the standard deviation to reduce the effect of extreme values on the typical cases.

But just applying a normalisation technique still not generates the correct results satisfying the before established search conditions. We have seen earlier that we also have to solve the problem of different degrees of importance and distinct types of variables. That is why we should consider weighting the variables in a next step to introduce information about the importance order and to comply with the search order. More information on weights and how to choose them is given in the next subsection.

Weights

The distance function has to reflect the application-specific search conditions and, above all, it has to maintain the before mentioned established importance order. Obviously, the normalised Euclidean distance (5.7) is not adequate enough and thus cannot be used because there is no possibility to define any type of order. In some cases the choice of weights may even result more critical than the choice between the types of distance measure itself, e.g. Euclidean vs. Manhattan distance.
It is stated correctly in [33], that most distance functions, including the Euclidean distance measure, give equal treatment to all variables (dimensions) which, however, might not be of equal importance. In our application, the distance function has to respect the established importance order, thing that could be achieved by introducing some type of weights or quality criterions. Following intuition, more informative variables should be assigned higher weights than less important variables. Non-informative features could even be discarded.

In [34] is supposed that many variable-weighting variants present only case studies. The same concludes [35], saying that most weight assignations are empirical i.e. the weight values are retrieved by experiment or observation. Furthermore, there exist few theoretical works on optimal weight settings, but much more effort was spent on weight learning methods where NNS is used for classification problems. We are not able to apply these methods because we confront no classification problem, but use NNS just for knowledge retrieval and therefore cannot provide class information which most methods rely on [34].

It was observed by [36] that considering only a small set of weights typically gave better results than using a larger set. Searches run with one non-zero weight, which assumes that a variable is either relevant or irrelevant and which is also known as feature selection, where difficult to outperform. This is due to the "curse of dimensionality" where the number of sample data needed to retrieve a valid parameter grows explosively with the number of dimensions.
Working with synthetical data, we dispose of enough sample data. (The synthetical data is complete with respect to given data granularity.) Therefore, we can use more sophisticated weights than just 0 and 1. Actually, in many applications weights are real values in the interval [0...1] and their values sum up to 1.
We use problem-specific knowledge involving the importance number of every feature to assign specific constant weight settings to all of the

variables. Our proposed static weights are calculated according to the following formula:

$$w_k = \frac{d - (i(k) - 1)}{\sum_{s=1}^{d} s} \tag{5.8}$$

where w_k is the weight of the k^{th} feature of the configuration parameter, d is the number of dimensions in the search space and i is the ordering or importance number of the feature (e.g. $i(fuelmodel) = 1$, $i(slope) = 2$) with $1 \leq i \leq d$. The adapted Euclidean metric including range normalisation and weights thus changes to

$$Weigh\_Norm\_Eucl\_dist(\vec{x}, \vec{y}) = \sqrt{\sum_{k=1}^{d} w_k \left(\frac{x_k - y_k}{max_k - min_k} \right)^2} \tag{5.9}$$

The weighted normalised Euclidean distance measure is still not appropriate to compute distances between fire configurations. The last issue remaining to be solved is the one of different types of variables. In a last step, we analyse the characteristics of each variable type and show how to expand the Euclidean distance into a heterogeneous distance measure to be able to treat nominal variables correctly.

Heterogeneous distance function

Having considered to normalise variable values and to incorporate their different importance using weights, we are still not finished. A last point to look at is the type of variable as already mentioned in 2.4.2. We show why different types of variables need to be handled variably when calculating distances.

We already know that our set of features is not purely quantitative, but includes the nominal variable fuel model, i.e. we have to deal with multivariate data that have different types of measurement scales. All standard L_p-metrics, which includes the Euclidean distance, assume that the input variables are linear. However, using a linear distance measurement on nominal features makes little sense because their values were assigned numbers in an arbitrary manner and these numbers do not represent any kind of linear order. Therefore, applying the arithmetical operations of the Euclidean distance function to nominal unordered values results counterproductive [30].

Think about the following example: The fuel models timber grass (fuel model group grass), dormant brush (fuel model group shrub), and

timber litter and understorey (fuel model group timber) are encoded with their respective values 2, 6 and 10. The Euclidean distance would rate timber twice as distant from grass as shrub, which might not be reasonable from the biological point of view. The variable fuel model consists of a discrete set of unordered attribute values. Therefore, a distance function is needed that handles nominal inputs appropriately.

According to [30] there exist basically three approaches to cope with diverse types of variables. Firstly, one could ignore the nature of the data. This technique is not suitable for us as it would produce wrong results in our application. Secondly, variables could be transformed to use only one scale. If the nominal scale is the simplest in the sample data, quantitative features have to be categorised. Here, deciding the number of categories is difficult and treating categories as nominal the order information vanishes. Further on, transformation from a lower to a higher scale is not possible. The last approach includes the application of a heterogeneous distance function that is capable of handling different scales, instead of using a single homogeneous metric. One realisation of this tactic, investigated and explained in [30, 34, 37], is to expand e.g. the Euclidean distance function to include a correct distance measure for symbolic features.
The resulting distance function is called a heterogeneous distance function because it uses different distance functions on different types of variables. For nominal data, the notion of how far apart two values are reduces to a simple binary relation: they are either the same, or they are different [16]. This gives rise to value-matching-based [30] metrics. If the values are the same, then the distance is 0; otherwise the distance is 1. For example, we could include the overlap metric for symbolic variables into our so far adapted Euclidean distance. This, in fact, was proposed by [34] and [37] but without a weighting factor. They called the resulting distance measure *HEOM (Heterogeneous Euclidean-Overlap Metric)* which is defined as follows:

$$HEOM(\vec{x}, \vec{y}) = \sqrt{\sum_{k=1}^{d} h_k(x_k - y_k)^2} \tag{5.10}$$

where d again is the number of dimensions. The term $h_k(x_k, y_k)$ specifies the distance measure for each variable according to its type the following:

$$h_k(x, y) = \begin{cases} overlap(x, y), & \text{if kth variable is nominal} \\ rn\_diff_k(x, y), & \text{if kth variable is numeric} \end{cases} \tag{5.11}$$

$overlap(x, y)$ then is defined as

$$overlap(x, y) = \begin{cases} 0, & \text{if } x = y \\ 1, & \text{otherwise} \end{cases} \tag{5.12}$$

and $rn\_diff(x, y)$ finally leads to the calculation of the Euclidean distance

$$rn\_diff_k(x, y) = \frac{|x - y|}{range_k} \tag{5.13}$$

already including range normalisation as proposed in the section about normalisation.

$$range_k = max_k - min_k \tag{5.14}$$

As one can easily observe, the above definition for h_k always returns a value which is in the range 0...1, whether the attribute is nominal or linear. According to [16] this is important. A heterogeneous distance function should not just combine the homogeneous metrics into a single metric that applies each homogeneous metric to its corresponding attributes because such a combination could suffer problems with scaling and one metric may still dominate the overall result.
Using HEOM and including the weight factor as formulated in (5.8), the final distance function for our purposes results in

$$WHEOM(\vec{x}, \vec{y}) = \sqrt{\sum_{k=1}^{d} w_k h_k (x_k - y_k)^2} \tag{5.15}$$

We now dispose of a proper distance function that is able to handle different ranges, different types of variables, and includes weights to consider the necessary importance order.

One topic which remains to mention in this context, affects the nominal feature *fuel model*. This variable is not strictly nominal in the sense that its values form an unordered set without further possible comparisons or orderings. Actually, some fuel models are more similar than others. As explained in 2.4.1 and [15], the 13 standard fuel models are arranged in four fuel groups. Thus, fuel models from one group, like grass (encoded as values 1 to 3), are more similar to each other than to models from other groups, e.g. timber (encoded as values from 8 to 10).
How do we deal with this additional information? One approach is to neglect

it as done in the elaborated distance function because this includes only a value matching distance for nominal attributes and does not consider further information. In addition, the established search conditions require the equality of the fuel model. The implemented search simply delivers no result, if the corresponding fuel model is not present in the database. Another, more general approach is to break down the feature fuel model into its respective properties listed in 2.4.1. The six emerging variables then would be included into the database replacing the feature fuel model. The great disadvantage of this solution is the increased dimensionality of data. It will provoke a data explosion in the case of synthetical data because a much larger number of records is needed if the data should be complete in the sense described in 5.1.1.

In the next section we show how the WHEOM distance function is implemented to retrieve the most similar configuration from the knowledge base. In addition, we demonstrate how only a small part of the data has to be searched in order to find the most similar configuration without requiring any sorting. We also point out the advantages of the employment of a database over the naïve approach.

5.2.2 Implementational Details

After having clarified the theoretical basics of the knowledge retrieval, we now can proceed to implement the Nearest Neighbour Search in our database.
However, traditional databases are built around the concept of exact searching: queries to the database return records whose variables match some search criterion [38] and not such rather fuzzy requirements as returning the 'most similar' record.
The simplest solution to the NNS problem therefore is to calculate the distance from the query point to every other point in the knowledge base and then to select the minimum, or better, to keep track of the "minimum so far". This brute-force procedure, also referred to as the naïve NNS approach, works well for small databases but can quickly become intractable as either the size (many records) or the dimensionality (many variables) of the problem become large [39]. In the case of synthetical fire data a rise in dimensionality normally leads to an increased number of records because we generate knowledge data for all possible combinations given a particular data granularity (see 5.1.1).
Remembering that the efficiency of a solution approach is measured by the query execution time and the storage requirements of the underlying data structure, the linear brute-force method might not be the most efficient in runtime but is the most storage-saving because there are no specific search data structures to maintain. Consequently, the linear approach has no additional space complexity beyond the storage of the database.
In one dimension the closest pair problem reduces to sorting. In a sorted set of features, the closest pair corresponds to two features that lie next to each other in

sorted order. We only need to check which is the minimum gap between the $n-1$ adjacent pairs. The runtime of the naïve method can be approximated with $O(nd)$ [39] where n is the cardinality of our knowledge base D and d is the dimensionality of D.

When the data set contains just a small number of points, the simple approach is best. Only when fast queries are necessary for a large number of points does it pay off to consider more sophisticated methods. There exist several approaches improving the execution time of the brute-force nearest neighbour search, but in higher dimensions, however, these algorithms have an exponentially growing space requirement [40].
In two dimensions, Voronoi diagrams provide an efficient data structure for nearest-neighbour queries. Although Voronoi diagrams can be built in higher dimensions, their size quickly grows to the limit of unusability. In moderate-dimensional spaces the *kd*-tree data structure does a very good job.

The usual procedure to extract information stored in a database is to run a query in a language the database 'understands'. For relational databases there exists the standardised query language SQL (Structured Query Language) whose basic constructs and functions work with any relational database. The query then returns as a result one ore more records or aggregated values fulfilling the applied search criteria. That is why we need to implement the search for the most similar fire configuration in SQL such that the configuration with the minimum distance to some given configuration is returned.
The simplest solution to our NNS deals with calculating the distance according to the distance measure worked out in the previous section for every data point in the database and then selecting the minimum, i.e. implementing the brute-force procedure to solve NNS. Enhanced NNS methods would implicate too much storage overhead for the current database configuration.
Besides, we show that the distance calculation only has to be carried out for the minor part of records applying intelligent database techniques. We start with a simple straightforward implementation of the distance function and will then refine it gradually until coming up with satisfactory results.

To start with, a sequential scan through the data is performed storing the computed distance to the query point for each record in a new column. Afterwards the obtained distances are sorted in ascending order before the first record with minimum distance can be returned. The SQL query accomplishing this described behaviour is listed in appendix A.1.
But the performance of this query is more than poor. For every record the minimum and maximum values of the linear variables have to be computed to carry out range normalisation. Assuming that our knowledge base includes no indices and no ordering, for each min, max and sorting operation a full table scan (FTS) has

to be executed. This would lead, theoretically, to $n2d_{lin} + 1$ total scans through the data, where n is the number of records in the table and d_{lin} the number of linear dimensions present in the table (Remember that nominal variables do not have to be normalised but always return a value of either 1 or 0 using the given value-matching metric). The last FTS is required for sorting.
Although this fits with the above mentioned estimation of runtime for NNS, $O(nd)$, it is totally unacceptable for a database application and could result even worse than the naïve text file-based approach. Furthermore, full table scans are not scalable as the knowledge base grows. As more data is included in the table, the more data has to be processed to complete the query.
This is why we now show how to gradually improve query execution time using some special database functionalities.

Maybe the query analyser of the underlying database management system (DBMS) is smart enough to detect the ever repeating tasks calculating minimum and maximum values and can eliminate them all by itself but to be on the safe side, the most obvious enhancement is to calculate the range of each linear variable before running the query and storing it in a temporal variable as implemented in appendix A.2.
Thus, the number of completed FTS can be reduced to $2d_{lin} + 1 + 1$; searching minimum and maximum value of each linear variable, doing the distance calculation and sorting the results.

A further advance can be reached after analysing how the distance is calculated for nominal variables, described in the previous subsection. The measure computing the space between nominal feature values works on a value-matching basis. Furthermore, in the established search conditions, we require the values of the nominal variable fuel model to be equal comparing each data point to the query point to be eligible. Both circumstances suggest the application of a WHERE-clause in the query which tests the nominal feature values of each data point and the query point for equality.
By doing so and furthermore applying an index on the nominal variables, the FTS of the distance calculation can be reduced to a much smaller index scan, only calculating the distance for those records whose nominal feature values are equal to the query point's nominal feature values. The higher the cardinality of the nominal variables, the more selective result the variables, the fewer records have to be examined and the better works the index.
In the case of the presence of one nominal variable, instead of a full table scan, only $n/card(var_{nom})$ records have to be scanned, assuming a uniform distribution of the nominal variable values in the knowledge base.
In the present application, the number of FTS can be now reduced to $2d_{lin} + \frac{2}{card(var_{nom})}$ where $card(var_{nom})$ is the cardinality of the nominal variable, because we only have to sort the records for which we computed the distance.

Lastly, the nominal variables can be removed from the implemented distance calculation because they do not contribute anymore to the calculated distance. Remember that the distance is 0 if the values are equal. The application of weights has to be limited to the remaining linear variables, leading to changed weight values for the variables. The resulting query is shown in appendix A.3

Not only is an index useful if the query contains a WHERE-clause, but can also help to improve the retrieval of maximum and minimum values of a column.
Generally spoken, an index is a database structure that provides a quick lookup of data in one or more columns of a table. Indices may help to speed up queries amazingly but they also carry some costs with respect to storage and maintenance. More precisely, indices occupy additional storage and have a cost for insert, update and delete statements. Each index is an additional system-managed table with need to be maintained and adapted when altering data. Thus, having a lot of indices might speed up select statements, but slow down inserts, updates and deletes.
In the case of our fire configuration knowledge base, once established, it will be exposed to no or at most very few data changes, thus justifying the extended use of indices not only for the nominal feature columns, but also for the columns accommodating linear features. If the index, mostly implemented as a B-tree, stores the linear variable in sorted ascending order, the minimum value can be retrieved directly and the maximum value using $log(n)$ data reads. By doing so, we can eliminate the full table scans during pre-processing calculation of the range for each linear variable.
Finally, using special database techniques in the current knowledge base, all FTS have been removed.

A last enhancement improving overall simulation performance consists in storing the range values for linear variables at simulation setup, assuming that the knowledge base will not change during simulation. The computation of range values can thus be avoided in every simulation step.

We have proven that, under no circumstances, all configurations contained in the knowledge base have to be evaluated in order to retrieve the most similar configuration if we implement the application with a database management system. More important, we are not bound any longer to a sorted order of configurations and can easily enter and delete new fire configuration records without further measures of precaution. Following, we sum up the numerous other advantages of the database retrieval method.
First of all, data is bundled in just one location but not the complete sample data has to be examined or even loaded into main memory upon access. We were able to decrease storage occupation by more than 50% storing the expert knowledge in a database instead of a text file (357 kB vs. 796 kB applying the variable configuration stated in table 5.1). Deploying all proposed indices still yielded a

storage improvement of 4%. Using numerical data, it is faster normally to access information from a database than to access a text file. The information contained in the database is likely to be saved in a more compact format than in the flat text file. Accessing it, thus, involves fewer disk accesses. Therefore, we expect retrieval times to find the best fitting parameter to decrease and will verify such behaviour in the next chapter conducting and resuming the necessary experiments.
Further on, a database presents a more structured approach including a labelling system for column names which reduces confusion on changing or updating data and decreases the number of unintended and undetected mistakes. Thus, the system becomes more robust. Moreover, a database management system imposes strict design parameters on developers and therefore ensures that the data retains its integrity and accuracy.
The use of triggers upon record insertion facilitates the detection of outliers and unfeasible values which can then be discarded automatically in order to minimise the occupied storage space. Triggers also are very helpful on the incorporation of historical data from real forest fires because information can be pre-processed automatically if not in the right format or measuring unit.
Introducing a database for information retrieval further on introduces some standard as standard SQL can be used to execute queries. This removes the need for an additional parser. Thus code can be saved in the simulation application because no text files have to be parsed to find line and column boundaries.
Also, using standard SQL, the underlying DBMS can be changed or replaced without problems or supplementary adaptations.
A database, in addition, allows multiple users to access and use data simultaneously, which greatly improves efficiency of systems and really allows for parallelisation of the application instead of executing all procedures busy with information retrieval on the master.
To finish, a DBMS is easily expandable (add further columns or tables) in the case that new requirements occur. This includes the possibility of modelling relations between different objects, something that is impossible with the exclusive use of text files.
Summarising, the database solution offers a user-friendly and interactive front-end which makes the handling (searching, deleting, inserting, updating) of data more easy. Information can be organised, processed and managed in a structured and controlled manner. This eases overall maintenance and management of the sample data, saves time, and eliminates error sources.

One mayor disadvantage of this enhanced retrieval method is the need to setup a database server. In reality, this should be less problematic as most system environments where fire propagation is studied or simulated (investigation centres, universities, fire departments...) count with one or another database management system where a new database can easily be included. Otherwise, a free product is simple to install and run.

The last question one could stumble upon treats the fact that our knowledge base, at the moment, is made up of merely a single table and that the involvement of a complete DBMS therefore would be unnecessary. We just saw that storing data in a text file causes a lot of inconveniences and maybe a simple spreadsheet would do. But we think more general and will include more information in the knowledge base in the long run which could be helpful to improve prediction results (see 7.2). Once disposing of a database, we should use its advantages to their full extent.

The next chapter presents the experiments we conducted to show the correctness of our established distance function. We also show how retrieval times will decrease using the database retrieval method compared to the text file based approach, although we applied the brute-force method to implement NNS.

Chapter 6

Experimental Results

The current chapter presents some experiment scenarios to prove the earlier mentioned expectations (5.2.2) of decreased knowledge retrieval time. We concentrate on providing test cases for this specific topic because it was the objective of the work to enhance retrieval time conditions. The overall performance of the application with respect to prediction results and how applying a knowledge-guided GA reduces the error in fire prediction can be found in [8]. There can be found the exact settings of the applied Genetic Algorithm, too.

6.1 Experimental Framework

For all following test scenarios we worked with one synthetical fire plot. This is perfectly reasonable as we are only interested in the different retrieval times of the naïve approach and the database method. Using a synthetical plot we can also show the correctness of the new knowledge retrieval approach by simply checking the results of both methods for equality.
The knowledge base was implemented as a MySQL database using the configuration shown in table 5.1. Applying the same knowledge configuration assures results which are comparable to the naïve approach presented in section 5.1. Further on, all 'tables' from the naïve approach (one text file per fuel model) were stored in a single table to be the most similar to the database implementation. In a final step, the search in the text files was decoupled from parameter order always reading the complete number of records to find the appropriate knowledge. To obtain the most equal testing conditions, for all test cases only retrieval times were measured disregarding the time needed for connection or disconnection in case of the database or the time required to open and close the text file.
Next, the test cases are described and which results they yielded.

6.2 Enhanced Retrieval Times

The first experiment is conducted to show the gradually decrease of knowledge retrieval times along with the refinement of the SQL query as described in section 5.2.2. Figure 6.1 shows the obtained results.
Taking the retrieval time of the naïve approach as a starting point, simply deploying the database approach did not result in an enhanced performance but showed the same retrieval time. Calculating ranges of linear variables once before main query execution and implementing a WHERE-condition for nominal variables improved retrieval performance by 6%. Applying all proposed indices for nominal as well as linear attributes yielded time savings of 16%. Unfortunately, these savings go at the expense of occupied storage. As described in section 5.2.2, indices need additional storage. Having applied all indices, the database needs nearly the same storage as the text file approach (769 kB vs. 796 kB).

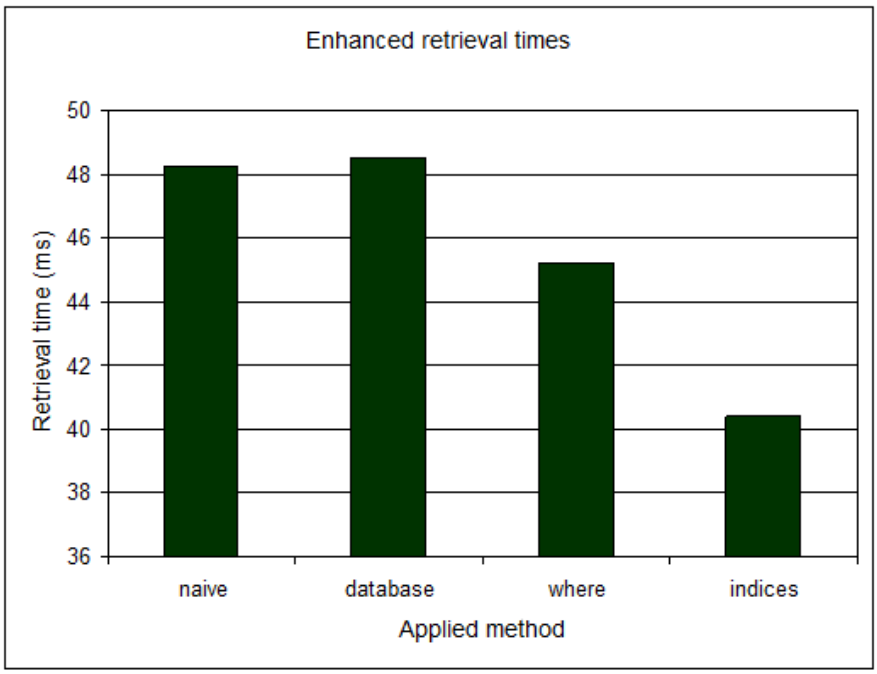

Figure 6.1: Development of retrieval times gradually improving the SQL distance function implementation.

At a first glance, these results represent the expected behaviour of decreased retrieval time but do not seem to be overwhelming. Taking advantage of a database may really pay off if the database contains more than the approximately 12,500 records from the current configuration. This is why we conducted a scalability test next.

6.3 Scalability

In order to observe how the retrieval methods react to an increased storage load and to measure the scalability of the single approaches we amplified the knowledge contained in the text file and in the database by the factors 2, 4, and 8 and again measured retrieval times for each approach. Figure 6.2 depicts the results and clearly shows the advantages of the enhanced database approach using indices.
While the retrieval time of the naïve approach grows with the same factor as data was augmented, implementing the WHERE-condition for nominal attributes and calculating ranges outside of the main query scales much better. Performance gains up to 68% were obtained using approximately half the storage of the naïve approach (1.39 MB vs. 3.09 MB; all values apply for enlargement factor 4).

Figure 6.2: Development of different retrieval approaches enlarging the amount of data.

The most surprising and at the same time promising results were obtained for the approach which includes the deployment of indices for the variables. Increasing the amount of data by factor 8, the needed retrieval time merely rises 4 ms (see figure 6.3), representing a scaling factor of 1.09. In this case, performance improvements up to 88% were obtained.
Once again, this performance boost goes at the expense of occupied storage caused by the additionally required storage for the indices and nearly reaches the amount of storage occupied by the naïve approach (5.92 MB vs. 6.19 MB).

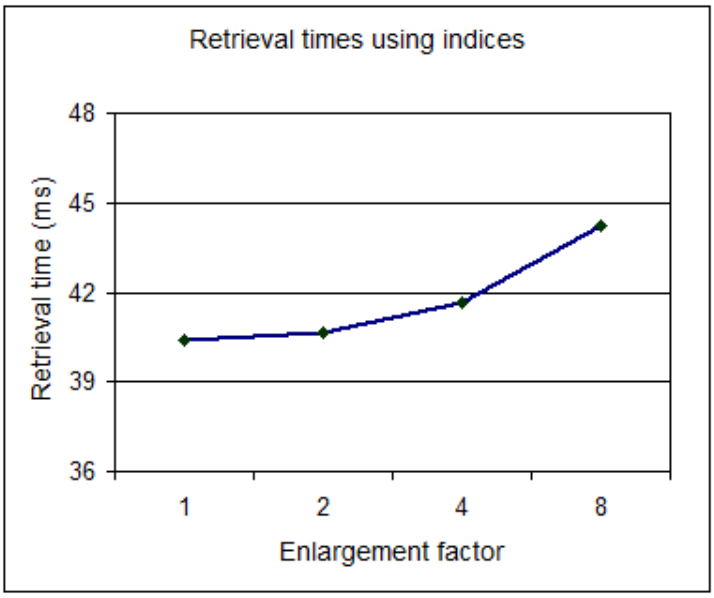

Figure 6.3: Retrieval times for the most advanced distance calculation implementation for an increasing amount of data.

6.4 Experiment Conclusions

Experiments showed that the retrieval time of the database method performed equally good or, in most cases, much better compared to the text file based approach. This ratio constantly gets better the more sample data is available, i.e. the bigger the knowledge base gets.
Obviously, the performance of the enhanced knowledge retrieval method clearly depends on its specific implementation. On the one hand, simply deploying a database and applying the distance calculation one-to-one in SQL without further database specific enrichments, yields rather similar retrieval times to the naïve approach and only starts to pay off slowly for a significantly augmented amount of data. On the other hand, having improved the query implementation by avoiding full table scans and using indices, results in nearly constant retrieval times, even for an increased amount of data. Summarising, the database implementations scale much better than the text file does.

Conducting the experiments, we were also able to show that the elaborated distance function is correct and the database approach retrieves the proper results. As the naïve approach, implementing the computational method to retrieve knowledge to guide the Genetic Algorithm, was proven to be correct in [8], we now simply compared results for equality.
In some cases, however, differing wind conditions were found comparing the results of the naïve approach with the ones from the database method. This happened e.g.

conditions (also see section 5.1.1). The naïve approach in this case retrieves the first value found with respect to parameter order. The database approach, having calculated the same distance for various records, is likely to select an arbitrary parameter with minimum distance. This slight difference in retrieval behaviour is negligible because the retrieved values are not used as fixed values to guide the GA but form the centre of a new, much smaller, range as described in section 4.3.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In the course of the present work a more robust data storage and retrieval system was proposed which serves for holding and searching the expert knowledge used to guide the Genetic Algorithm applied in the data-driven forest fire prediction to calibrate input variables.
We analysed the drawbacks of the naïve approach presented in [8] and suggested the deployment of a database management system in order to maintain data in a structured manner and to dispose of a user-friendly, interactive and less error-prone data management and retrieval method. Having elaborated an adequate distance function, allows us to retrieve domain knowledge paying no more attention to parameter order. We also checked the correctness of retrieved values against the simple approach with approved outcomes, thus deducing the proper functioning of the distance function.
Experiments show that we accomplished the objective of implementing a highly scalable solution which reduces storage occupancy to the half and, at the same time, diminishes retrieval times up to remarkable 68%. Disregarding storage optimisation, data retrieval performance could even be decreased up to nearly 90%.
Thus, now we are able to store more expert knowledge and can provide more precision without fearing that the retrieval of knowledge could become an application's bottleneck. Summarising, applying the database retrieval approach, the predictions of forest fires can be fastened and improved further.

7.2 Future Enhancements

Disposing of an effective and scalable knowledge storage and retrieval system, a next step comprises the further discretisation of knowledge obtained from synthetical burnings. Therefore, the granularity of the variables slope and wind speed should be refined to obtain supplementary combinations of fire configurations used to generate fire propagation knowledge. Whereas the variable fuel model already is discretised to a maximum. Neither is a further refinement of the variable wind direction suggested because most fire simulators only consider the eight primary wind directions.
To increase precision, we should also include historical data from real fires which seems most valuable. Although an exclusive use of real fire data is not recommended because it is not possible to cover all possible configuration combinations, it presents an essential and truthful source of knowledge. If the solely use of data from real forest fires is aspired, some additional quality assurance method has to be employed to secure a satisfactory level of data diversity and not to guide the GA in a wrong direction and slow down its convergence.
We additionally propose to investigate the inclusion of knowledge from further input variables in the knowledge base, e.g. moistures of dead and living fuel. Earlier, we already suggested breaking down the variable fuel model into its respective components. Because at the moment some implementations of the NNS simply deliver no result if the corresponding fuel model is not present in the database, this could be a way to get around this drawback. If one fuel model is not present in the knowledge base, a similar configuration from the same fuel model group could be returned.
In 4.3 we already suggested some ways to improve the performance and the results of the Genetic Algorithm. However, applying more knowledge guidance to the GA and enhancing it further, how much randomness can be stolen from the operations of the GA before it stops being a random methodology with its intended purposes?
At the moment, the knowledge base consists of one single table. In the long run, we will include more information in the database which could be helpful to improve prediction results. Recent investigations found out that, e.g. depending on the shape of the fire front the fire should be simulated differently.
A last idea covers the investigation of possibly eliminating the GA and directly using results from the knowledge base to calibrate input variables. In doing so, we expect notable execution time savings. At the same time we are aware that such approach heavily depends on data quality in the knowledge base.

Appendix A

SQL queries

This appendix comprises three SQL statements elaborated in section 5.2.2 which show the gradually refinement of the implemented Nearest Neighbour Search using the distance function established in 5.2.1[1].

Listing A.1: Straightforward SQL implementation of distance function.

```
SELECT *,
        SQRT
                ((4/10 * ((abs(fuelModel - QP_fuelModel)/
                abs(fuelModel - QP_fuelModel)) is not null)) +
                ((3/10) * POWER(abs(slope - QP_slope)/
                ((SELECT MAX(slope) FROM fireData) -
                (SELECT MIN(slope) FROM fireData)), 2)) +
                ((2/10) * POWER(abs(fireDir - QP_fireDir)/
                ((SELECT MAX(fireDir) FROM fireData) -
                (SELECT MIN(fireDir) FROM fireData)), 2)) +
                ((1/10) * POWER(abs(fireSpeed - QP_fireSpeed)/
                ((SELECT MAX(fireDist) FROM fireData) -
                (SELECT MIN(fireDist) FROM fireData)), 2)))
AS totalDist
FROM fireData
ORDER BY totalDist
LIMIT 1;
```

[1]The term *QP_xxx* in the below stated queries refers to the particular variable value of the query point QP.

Listing A.2: SQL query calculating ranges apart.

```
SELECT @range_slope :=
        (SELECT MAX(slope)
         FROM fireData) -
        (SELECT MIN(slope)
         FROM fireData);

SELECT @range_fireDir :=
        (SELECT MAX(fireDir)
         FROM fireData) -
        (SELECT MIN(fireDir)
         FROM fireData);

SELECT @range_fireSpeed :=
        (SELECT MAX(fireDist)
         FROM fireData) -
        (SELECT MIN(fireDist)
         FROM fireData);

SELECT *,
        SQRT
                ((4/10 * ((abs(fuelModel - QP_fuelModel)/
                abs(fuelModel - QP_fuelModel)) is not null)) +
                ((3/10) * POWER(abs(slope - QP_slope)/
                (@range_slope), 2)) +
                ((2/10) * POWER(abs(fireDir - QP_fireDir)/
                (@range_fireDir), 2)) +
                ((1/10) * POWER(abs(fireSpeed - QP_fireSpeed)/
                (@range_fireSpeed), 2)))
AS totalDist
FROM fireData
ORDER BY totalDist
LIMIT 1;
```

Listing A.3: SQL query with nominal variables placed in WHERE-condition.

```
SELECT @range_slope :=
        (SELECT MAX(slope)
         FROM fireData) -
        (SELECT MIN(slope)
         FROM fireData);

SELECT @range_fireDir :=
        (SELECT MAX(fireDir)
         FROM fireData) -
        (SELECT MIN(fireDir)
         FROM fireData);

SELECT @range_fireSpeed :=
        (SELECT MAX(fireSpeed)
         FROM fireData) -
        (SELECT MIN(fireSpeed)
         FROM fireData);

SELECT *,
        SQRT
                (((3/6) * POWER(abs(slope - QP_slope)/
                (@range_slope), 2)) +
                ((2/6) * POWER(abs(fireDir - QP_fireDir)/
                (@range_fireDir), 2)) +
                ((1/6) * POWER(abs(fireSpeed - QP_fireSpeed)/
                (@range_fireSpeed), 2)))
AS totalDist
FROM fireData
WHERE fuelModel = QP_fuelModel
ORDER BY totalDist
LIMIT 1;
```

Bibliography

[1] Forest Fires - Prediction & Analysis. *Web Page,* http://www.borealforest.org/world/innova/fire_prediction.htm, Visited July 2008.

[2] Science and Innovation - Forest Fires. *Web Page,* http://www.borealforest.org/world/innova/forest_fire.htm, Visited July 2008.

[3] Wildfire - Wikipedia, the free encyclopedia. *Web Page,* http://en.wikipedia.org/wiki/Forest_fire, Visited July 2008.

[4] The Curt Jester: Octobr 2006 Archives. *Web Page,* http://www.splendoroftruth.com/curtjester/archives/2006/10/, Visited July 2008.

[5] A. Bachmann and B. Allgöwer. Uncertainty propagation in wildland fire behaviour modelling. *International Journal of Geographical Information Science,* Vol. 16, Issue 2, pp. 115-127, March 2002.

[6] CSERD: What is Computational Science? *Web Page,* http://www.shodor.org/cserd/Help/whatiscs, Visited July 2008.

[7] SIAM: Graduate Education for Computational Science and Engineering. *Web Page,* http://www.siam.org/students/resources/report.php, Visited July 2008.

[8] M. M. Denham. Predicción de incendios forestales basada en algoritmos evolutivos guiados por los datos. *Master Thesis,* Universitat Autònoma de Barcelona, Spain, July 2007.

[9] Encyclopedia Collection - Forest Encyclopedia Network. *Web Page,* http://www.forestencyclopedia.net/, Visited July 2008.

[10] G. Bianchini. Wildland fire prediction based on statistical analysis of multiple solutions. *PhD Thesis,* Universitat Autònoma de Barcelona, Spain, July 2006.

[11] R. Rothermel. A mathematical model for prediction fire spread in wildland fuels. *USDA FS, Ogden TU Res.,* Pap. INT.115, 1972.

[12] Wildland Fire Management and Planning: Free Online Course Materials - USU. *Web Page,* http://ocw.usu.edu/Forest__Range__and_Wildlife_Sciences/Wildland_Fire_Management_and_Planning, Visited July 2008.

[13] FireModels.org - Fire behaviour and fire danger software. *Web Page,* `http://www.firemodels.org/`, Visited July 2008.

[14] *fireLib* User Manual and Technical Reference. *Web Page,* `http://www.fire.org/downloads/fireLib/1.0.4/doc.html`, Visited July 2008.

[15] H.E. Anderson. Aids to determining fuel models for estimation fire behaviour. *Intermountain Forest and Range Experiment Sation Ogden,* UT 84401, General Technical Report INT.122, 2002.

[16] Ch. Giraud-Carrier and T. Martinez. An efficient metric for heterogeneous inductive learning application in the attribute-value language. *Intelligent Systems,* pp. 341-350, 1995.

[17] B. Abdalhaq. A methodology to enhance the prediction of forest fire propagation. *PhD Thesis,* Universitat Autònoma de Barcelona, Spain, June 2004.

[18] R. Salvador, J. Piñol, S. Tarantola, and E. Pla. Global sensitivity analysis and scale effects for a fire propagation model used over mediterreanean shrublands. *Elsevier, Ecological Modelling 136,* pp. 175-189, 2001.

[19] G.A. Trunfio. Prediction wildfire spreading through a hexagonal cellular automata model. *Cellular Automata for Research and Industry, University of Amsterdam, The Netherlands, LNCS 3305,* Springer Verlag, Berlin, pp. 385-394, 2004.

[20] J. D. Beezley, S. Chakraborty, J. L. Coen, C. C. Douglas, J. Mandel, A. Vodacek, and Z. Wang. Real-time data driven wildland fire modeling. *ICCS 2008, Part III, LNCS 5103,* Springer Verlag, Berlin Heidelberg, pp. 46-53, 2008.

[21] R. Ursem. Diversity-guided evolutionary algorithms. *Parallel problem solving from nature, Granada, Spain, LNCS 2439,* Springer Verlag, pp. 462-471, 2002.

[22] F. Li and T. M. Lindquist. Knowledge guided genetic algorithm for optimal contracting strategy in a typical standing reserve market. *Power Engineering Society General Meeting, IEEE,* Vol. 2, pp. 859-863, July 2003.

[23] A. Berry and P. Vamplew. PoD can mutate: A simple dynamic directed mutation approach for Genetic Algorithms. *AISAT: International Conference on Artificial Intelligence in Science and Technology,* Hobart, Tasmania, Australia, November 2004.

[24] Q. Zhang. Knowledge incorporation in Evolutionary Computation [Book Review]. *Computational Intelligence Magazine, IEEE ,* No. 4, Vol. 1, pp.58-59, November 2006.

[25] D. M. Tate and A. E. Smith. Expected allele coverage and the role of mutation in Genetic Algorithms. *Proceeding of the Fifth International Conference on Genetic Algorithms,* Morgan Kaufmann Publishers, San Mateo, pp. 31-36, 1993.

[26] F. Divina and E. Marchiori. Knowledge-based evolutionary search for inductive concept learning. In *Knowledge Incorporation in Evolutionary Computation, Y. Jin (Ed.),* Springer Verlag, 2004.

[27] Q. Zhang and J. Sun. Iterated local search with guided mutation. *IEEE Congress on Evolutionary Computation CEC 2006,* Vancouver, Canada, pp. 924-929, 2006.

[28] C. Young, Y. Zheng, C. Yeh, and S. Jang. Information-guided genetic algorithm approach to the solution of MINLP problems. *Industrial & Engineering Chemistry Research,* No. 5, Vol. 46, pp. 1527-1537, 2007.

[29] Similarity Measurement *Web Page,* `http://people.revoledu.com/kardi/tutorial/Similarity`, Visited July 2008.

[30] J. Lumijärvi, J. Laurikkala, and M. Juhola. A comparison of different heterogeneous proximity functions and Euclidean distance. *MEDINFO 2004,* Ed. M. Fieschi et al., Amsterdam, IOS Press, 2004.

[31] S. Aksoy and R.M. Haralick. Feature normalization and likelihood-based similarity measures for image retrieval. *Pattern Recognition Letters,* No. 5, Vol. 22, pp. 563-582, 2001.

[32] L. Al Shalabi and Z. Shaaban. Normalization as a preprocessing engine for data mining and the approach of preference matrix. *Proceedings of the International Conference on Dependability of Computer Systems, IEEE Computer Society,* pp. 207-214, 2006.

[33] A. Hinneburg, C. Aggarwal, and D. Keim. What is the nearest neighbor in high dimensional spaces? *VLDB,* pp. 506-515, 2000.

[34] D. Wettschereck and D.W. Aha. Weighting features. *Case-Based Reasoning, Research and Development, First International Conference,* Springer Verlag, Berlin, pp. 347-358, 1995.

[35] Ch. Ling and H. Wang. Computing optimal attribute weight settings for nearest neighbor algorithms. *Lazy learning,* Kluwer Academic Publishers, pp. 255-272, 1997.

[36] R. Kohavi, P. Langly, and Y. Yun. The utility of fearture weighting in nearest-neighbor algorithms. *Proceedings of the European Conference on Machine Learning (ECML-97),* 1997.

[37] D.R. Wilson and T.R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research,* Vol. 6, pp. 1-34, 1997.

[38] E. Chávez, G. Navarro, R. Baeza-Yates, and J.L. Marroquín. Searching in metric spaces. *ACM Comput. Surv.,* No. 3, Vol. 33, New York, USA, pp. 273-321, 2001.

[39] Nearest neighbor search - Wikipedia, the free encyclopedia. *Web Page,* `http://en.wikipedia.org/wiki/Nearest_neighbor_search`, Visited July 2008.

[40] F. Bajramovic, F. Mattern, N. Butko, and J. Denzler. A comparison of nearest neighbor search algorithms for generic object recognition. *Proceedings of the advanced concepts for intelligent vision systems (ACIVS),* pp. 1186-1197, 2006.