A Methodology to Enhance the Prediction of Forest Fire Propagation

Department d'Informatica

Unitat d'Arquitectura d'Ordinadors

i Sistemas Operatius

A thesis submitted by **Baker Abdalhaq** in fulfillment of the requirements for the degree of Doctor per la Universitat Autònoma de Barcelona.

Barcelona (Spain), June 2004

A Methodology to Enhance the Prediction of Forest Fire Propagation

Thesis submitted by **Baker Khaldoun Baker Abdalhaq** in fulfillment of the requirements for the degree of Doctor per la Universitat Autònoma de Barcelona. This work has been developed in the Computer Science Department of the Universidad Autònoma de Barcelona and was advised by Dra. Ana Cortés Fité,

Bellatera June, 2004

Thesis Advisor

Ana Cortés Fité

To my parents

Acknowledgment

Writing a doctoral thesis is hard work but the collaboration of all my colleagues and my family made it easier. I take the opportunity to thank all the people who have helped me through the course of graduate study.

First I would not only like to thank my advisor Ana Cortés, for her invaluable guidance advice and encouragement, but also for creating a family atmosphere in the group, and I would like to thank Tomás Margalef for his help and his questioning that enriches the work. Thank to Emilio Luque for his discussions that inspires the work.

Many thanks are due to my colleagues in the unit AOSO, German Bianchini for his participation in the experiments, Josep Jorba providing his simulator and Xiao Yuan Yung for his discussions. I wish to also acknowledge my colleagues outside the unit, I would like to thank my colleagues in Coimbra, Xavier Domingos Viegas and his team that provided us with the possibility of carrying out experiments in the laboratory of Lousâo, Jorge André for his scientific curiosity and his deductions about some ideas of the thesis. Many thanks to my colleagues in CREAF, Imma Oliveras digitalizing the experiments videos and currying out the fire experiment and in helping to carry out the experiments in Lousâo and Josep Piñol's discussions of the main ideas of Thesis. I would like to thank the technical staff, Daniel Ruiz and Jordi Valls for their great job of making the cluster work properly and of solving the emerging problems in the research platform.

I would like to acknowledge the indirect help of the GNU[1] movement over the world by providing the environment and tools, such as the operating system, the compilers, editors and the world processors, which are essential to the investigation. I would also like to acknowledge the financial support of "Autònoma solidaria" and "SPREAD" project.

[1]This thesis has been written using LyX

My family, my wife and daughters have made a great sacrifice; moving from Palestine to Spain with all the associated difficulties such as the language and leaving the house and the surrounding family and neighbors. I would like to thank my wife and daughters for that. Special thanks to my parents that have planted and did not wait for harvest time. I would like to thank my sister Huda and my brother Saad for their caring and supporting me throughout my education.

My stay in Spain could not continue without the help of my new friends Muhannad Fatayer, Hesham Annajjar and Muhammad Abdalah, thanks to the hard times I faced when I first came to Spain I have made good friends for my whole life.

Contents

Chapter 1

Introduction

> "La aventura va guiando nuestras cosas
> mejor de lo que acertáramos a
> desear...que ésta es buena guerra, y es
> gran servicio de Dios quitar tan mala
> simiente de sobre la faz de la tierra"
>
> *Miguel de Cervantes*

With computers, scientists and engineers have made numerous discoveries that they would not have made otherwise. In fact, computers have revolutionized the way that many scientists do their work. Traditionally, science was done in laboratory as a combination of theory and physical experimentation (which included hand calculations), but computers have made possible a new and powerful way of doing science. Numerical simulation is the process of modeling mathematically a physical phenomenon, and then running an experiment with the mathematical model. Computational mathematics or computational scientists play a major role in this new way of doing science, creating, evaluating, and refining the mathematical models used to simulate the physical phenomena. Thus, computational science and *Grand Challenge* problems conform a new way of seeing "science".

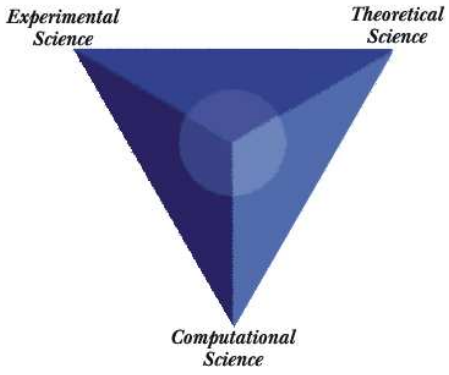

Figure 1.1: Three methods of investigation in natural sciences and engineering. (Computing includes, in this figure, numerical computations, simulation and visualization.)

1.1 Computational Science and Engineering

The power and availability of modern computers have made numerical calculations, computer simulations and other use of high performance computers to an important tool in almost all disciplines of science and engineering. Many researchers claim that *computing has become a third main method for doing investigations, besides theory and experiments,* [92, 20, 31]. There is probably no consensus about the relative importance of computing compared to the other two parts of this triad (Figure 1.1). However, there is no doubt that computing is being used in more and more disciplines as a main tool. Computing may be used *instead of theory and experiments* by providing insight in many phenomena that are too complex to be handled by analytical methods, too expensive, too dangerous or impossible to study through experiments [31]. Computing may also be a *supplement to theory and experimentation* resulting in an interplay among the three main methods.

At present, the computational approaches used by scientists and engineers are very similar [31], so *computational science* is often used as a shorthand for *computational science and engineering.* A very short (informal) definition of computational science is given by D. E. Stevenson in his article *Science, Computational Science and Computer Science: At a Crossroads* [102]:

> "We describe computational science as an interdisciplinary approach to doing science on computers."

The difference between the term scientific computing and computational science is presently not completely understood by the author. Golub and Ortega give the following working definition of "scientific computing" [45]:

> "Scientific computing is the collection of tools, techniques, and theories required to solve on a computer mathematical models of problems in science and engineering."

Golub and Ortega describe the difference in the following way:

> "... The techniques used to obtain such solutions [of mathematical models that represent some physical situation] are part of the general area called scientific computing, and the use of these techniques to elicit insight into scientific or engineering problems is called computational science (or computational engineering)."

From Native point of view [82], the difference expressed in the second part of this quotation may become a bit clearer when another definition is considered.

> ..... "use of HPC technology to advance the state of knowledge in a particular applications discipline"

Thus, informally, scientific computing may be perceived as more focused on "obtaining a solution on a computer", while CS&E can be said to be more focused toward using such solutions to increase the understanding of problems in science or engineering. In other words, in the term computational science it is important to realize that science is a variable x, representing any natural science or engineering discipline. Thus we have computational chemistry, computational physics, computational solid mechanics etc. Computational science is not the science of how to do computations, covering topics such as computational complexity theory (algorithm analysis, computability, NP-completeness etc.). Those topics are part of the well-established field theoretical computer science (TCS). CS&E experts need a certain body of knowledge in such topics just as they need it in mathematics, but computational science must not be misunderstood as a new name on such relatively traditional "computational topics" within computer science.

Computational science is a scientific endeavor (application) that is supported by the concepts and skills of mathematics (algorithms) and computer science (architecture) (see figure 1.2).

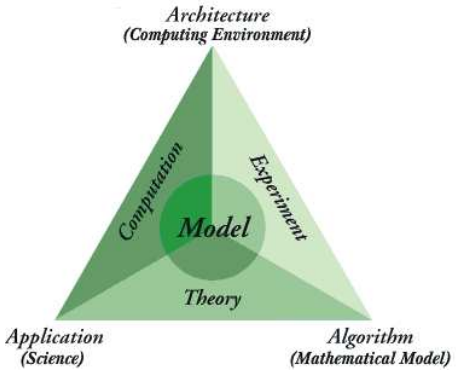

Figure 1.2: Computational Science

The major steps in scientific problem solving in computational science are basically the same as for ordinary problem solving except for an additional step that converts the devised plan into a computational method. In this context, the steps in scientific problem solving become (similar to Polya's list)

1. Problem formulation - This step is an exploratory, familiarization step. Understanding the problem, determining the relevant physical/scientific principles, developing an intuitive, conceptual picture of the problem are all things done at this level. This is where the gee whiz aspect of science occurs (what happens if I push here?)

2. Modeling- Develop a model (usually mathematical) that describes some aspects of the problem. Mathematically, this is really an abstraction of the physical problem, and as such involves approximations (i.e. certain physical aspects are considered unimportant or irrelevant).

3. Computational method- Develop a procedure for solving the mathematical formulation of the problem. Historically, this has often involved a good deal of mathematical analysis. There are very few problems, which can be solved exactly, whether the solution is attempted by approximate analytical techniques or by numerical computations.

4. Solution implementation- Carry out the calculation either by evaluating the approximate analytic solution or by coding the numerical computations.

5. Assess/analyze the results in context of computational method, mathematical model, and original problem – determine how relevant the results are to the original problem. Because the results represent a calculational approximation to the mathematical approximation of the problem, it is important to form a judgment as to whether the results are meaningful or not. For example, these results might imply something that is utter nonsense physically.

These steps exist for both traditional and computational science; the devil is in the details. For example, in step 2 above one needs to use one or several numerical "recipes" to begin the solution of the mathematical model generated. Many numerical recipes are too complex to calculate by hand and/or require repetitive calculations – iterations – to get close to an answer. At this point we can use the technologies of computer science to implement our algorithm or mathematical model on some suitable sized computer using some computational software tool. Needless to say, this whole process is itself "iterative" – solutions to preliminary algorithmic approaches to the problem generate a better algorithm, perhaps with the need for increasing computational power and/or precision. In other words, computer modeling can be done instead of a strictly mathematical model and the solution method would then be a computer simulation (an example of this is lattice gas automata). The mathematical analysis of the computer modeling would be impossible.

There is a nice, historical example that illustrates the differences between traditional science and computational science- the development of the model for the solar system.

1. Experimental science- Tycho Brahe (1546-1601) was a Danish astronomer who, over a twenty year period, painstakingly recorded the positions of celestial bodies (including planets) before the invention of the telescope. In order to do this, he invented things such as the sextant, the mural quadrant, and the ring armillary. Notice that these instruments have had wider uses than the original intent Experimental scientists observe how nature behaves, designing experiments, building equipment to improve their measurements, refining their experiments if unexpected results are measured.

2. Computational science- Johann Kepler (1571-1630) was a German astronomer who was Brahe's assistant at the end of Brahe's life. Kepler

inherited his entire observational record. Kepler set himself the goal of finding the patterns in the data. Over the years, he filled book after book with calculations on the data. Eventually, he came up with what are known as Kepler's laws of planetary

(a) Planets move around the sun in an ellipse with the sun at one focus of the ellipse.

(b) The line between the planet and the sun sweeps out equal areas in equal times.

(c) The square of a planet's orbital period is proportional to the cube of is average distance from the sun.
Kepler's laws are consistent with Brahe's observations but do not explain why the planets move in this way.

3. Theoretical science- Isaac Newton (1642-1727) was an English physicist who discovered the laws of motion and gravitation. These laws describe the motion of every object, not just planets. They explain Kepler's laws as resulting from gravitation.

Theoretical scientists use mathematics to give a succinct, abstract representation of physical phenomena in order to explain and predict how nature behaves. Note that these predictions/explanations follow from the mathematical model. To the extent that these models describe reality (recall that these models are still approximations) they give correct results. Discrepancies often indicate refinements of the models are needed. Theoretical scientists deal with abstractions and generalizations of observations and calculations, i.e. they deal with general knowledge instead of specific knowledge.

Computational scientists use both theoretical and experimental knowledge to develop computer models of reality. These models are then used to simulate the behavior of nature- they can be verified by comparing with nature. They are simulated versions of nature. Note that in this simulation certain aspects of nature could be changed, creating a virtual reality.

1.2 Grand Challenge Problems

Historically, studies of specific problems and corresponding breakthroughs have led to new scientific disciplines. Physics, Chemistry and Astronomy emerged out

of Natural Philosophy following the important discoveries by Newton, LaVoisier and Galileo respectively. The type of problems in which computational science could achieve breakthroughs are generally referred to as "grand challenges". A nice definition of grand challenges stems from Office of Science and Technology Policy:

> Grand Challenges are... fundamental problems in science and engineering, with potentially broad social, political, and scientific impact, that could be advanced by applying high performance computing resources.

As expected, there is a long list of grand challenge problems, including electronic structures of materials, turbulence, genome sequencing and structural biology, global climate modeling, speech and languages studies, pharmaceutical design, pollution and dispersion, and many more. In particular, the Grand Challenge face up in this thesis is Forest Fire Prevention and Mitigation.

1.3 Prediction of Forest Fire Propagation: A Grand Challenge Problem

Forest fire is one of the most critical environmental risks in all the Mediterranean countries due to the high temperatures and low precipitation rates, especially during the summer. This is an important problem throughout the world, but in these areas it is especially dangerous. Every year, intensive forest fires burn thousands of hectares, destroying many trees and natural resources. Moreover, it implies a progressive turning of land into desert with all the associated problems (figure 1.3).

For all these reasons, it is very important to fight against such forest fires using all the available resources in order to minimize their effects as much as possible. The fight against fire must be carried out at two main different levels:

1. Forest fire prevention: At this level, the administration must promote the education of civil society to avoid risks that can provoke a wild forest fire. However, there are many other things that must be done to minimize fire effects as much as possible. For example, it is necessary to work on the planning of the terrain to prepare natural fire-breaks, to decide which terrain can be urbanized, to prepare evacuation plans, to design

(a) (b)

Figure 1.3: (a) land before fire (b)land after fire

roads and ways of reaching all the places in the country and so on. If the administrative decisions are made in the proper way, fire effects can be minimized and the fight against the fire during a real emergency can be made easier.

2. Forest fire fighting: During a forest fire emergency, it is necessary to use the available resources in the best possible way. However, fighting against the fires implies the coordination of several groups (planes, helicopters, firemen, and volunteers) for the adequate use of combative resources (figure 1.4). It is very important to predict fire behavior so as to avoid accidents and to make rational decisions. For instance, if we know the fire propagation speed, direction and intensity, we will be able to decide what front we have to fight on and the effective way of fighting.

In both cases, fire prevention and real emergency, it is very important to have tools that are able to predict forest fire propagation, taking into account particular conditions. One of the goals of fire-model developers is to provide physical or heuristic rules that can explain and emulate fire behavior and, consequently, might be applied to creating robust prediction and prevention tools. In the following section, we will describe forest-fire simulator as a tool for predicting fire behavior.

Figure 1.4: fire fighting groups in act

1.4 Forest Fire Simulators

The process of developing a simulator software often is done in team made up of members bringing a variety of different skills. They use the following process in order to solve problems.

1. Identify an appropriate real-world phenomenon; Scientists begin the solution process by defining the problem of interest. It is important to identify clearly the aspect of the problem that will be considered. The information that is "known" about the problem must be stated. It will be used to ensure that the model represents the original problem as closely as possible. Then the questions to be answered must be posed clearly. The scientists may then assemble an inter-disciplinary team.

2. Construct a mathematical model; The mathematical model is usually a very large collection of equations and inequalities. Moving from the original problem to the mathematical model almost always requires making simplifying assumptions. Perhaps it will be modeled in 2 dimensions rather than 3, for example. These assumptions will need to be reconsidered when

the results of the numerical experiments are obtained.

3. Design a numerical algorithm; An algorithm is a set of step-by-step instructions for accomplishing a task. The algorithm is intended to produce the solution to a problem or sub-problem by operating on the model.

4. Build a computer code that implements the numerical algorithm; Often, steps in the numerical algorithm are general enough that they can be implemented in computer code in more than one way. The computational scientist will need to decide how to implement the algorithm efficiently on the type of computer that will be used to solve the problem. Many of the steps in the algorithm may be implemented by using existing software; others will require new code to be written.

5. Perform numerical experimentation (simulation) with the intention of:
   a) evaluating (validating) and revising (if necessary) the model. If the results replicate behavior that scientists know to be valid, then they will have more confidence that the predictions of the model will also be of value. Sometimes, results from simulations will lead to additional questions about the phenomenon being studied, and the mathematical model will need to be modified to reflect a new direction of research. Occasionally, researchers will realize this point that they forgot to include some piece of information (for example, that a particular value can never be negative) that is common sense to person, but must be stated explicitly in a mathematical model.
   b) generating mathematical conjecture and subsequent new theory concerning numerical and theoretical properties of both the model and the algorithm[112, 113].

The following sections are organized according to these development steps.

1.4.1 Forest Fire Models

Models are being increasingly used to make predictions in many fields of the environmental science. Models, however, have a series of limitations of which the lack of accuracy and/or a proper validation have the most dramatic consequences. However, other restrictions generally occur when applying models. Frequently, information on a high number of input variables is required to run a model, this need being specially restrictive for models with a spatially explicit

expression (since these require data for variables in each site). Fire propagation models are a good example of data demanding (data restricted) models.

It must be considered that the forest fire propagation is a very complex problem that involves several aspects that must be considered:

1. Meteorological aspects: The meteorological conditions affect fire propagation in a direct way. Temperature, wind, moisture, and so on modify fire behavior and propagation in a significant way. It must be taken into account that these conditions are not static, but these change due to the macro-meteorological conditions or the day-night cycle. Therefore, the forest fire propagation prediction should consider prediction of the meteorological conditions as well. In a more accurate analysis, it must be pointed out that the fire itself modifies the temperature, wind conditions and so on.

2. Vegetation features: Vegetation features influence the fire behavior in a direct way. However, there are points related to meteorological conditions that modify the features of the vegetation. For example, the moisture contents of the vegetation influences fire behavior, but this content depends on meteorological conditions (past and current).

3. Topographical aspects: The topographical aspects of the terrain are also very significant to predict the fire behavior. But the particular topographical conditions also modify the meteorological conditions. For example, the meteorological wind is modified by the topography of the terrain in such a way that the particular wind at each point must be evaluated and therefore, it must be analyzed as a wind field with a particular value in each point.

For all these reasons it can be concluded that the forest fire propagation prediction is a very complex problem involving several disciplines that must co-operate to provide accurate models and solutions that predict the fire propagation in a realistic way. Research on these models involves researchers in physics, chemistry, biology or ecology. In other worlds, forest fire propagation prediction is a Grand Challenge problem.

There are several models in the literature to describe the behavior of forest fire propagation. First of all, it must be pointed out that the propagation models include two separate models: the global model and the local model. These two models consider two different scales. On the one hand, the global

model considers the fire-line as a whole unit (geometrical unit) that evolves in time and space. On the other hand, the local models consider the small units (points, sections, arcs, cells, ...) that constitutes the fire-line. These local models take into account the particular conditions (vegetation, wind, moisture, ...) of each unit and its neighborhood to calculate the evolution of each unit.

The local fire-spread model calculates the movement of each individual section of the fire-line and then the global model calculates the total fire-line applying an aggregation process. The local fire-spread model takes into account the static and dynamic conditions of the terrain (vegetation, topography, wind, moisture, and so on). The dynamic conditions (mainly moisture and wind) must be evaluated before the local model can calculate the movement of the section.

The global model allows the partitioning of the fire-line into a set of sections. In each of these sections certain, local balance conditions must be observed [6, 7]. Under these conditions the movement of the fire-line can be considered as the separate movement of the different sections, and then it is possible to compose the fire-line in the next time step by aggregating the particular movement of the different sections.

In the literature, there are several approaches to solving the global models. These approaches can be classified in the following categories:

1. The fire-line is considered as a set of units (points, sections, arcs, ...). It is assumed that each section has its own desegregate movement and then the new position of the fire-line is determined by aggregating the new position of each section.

2. Physical approach based on Huygens principle. The fire-line is considered as a set of points. Each point is considered as an ignition point that generates a virtual fire-line, which evolves in the same way as a real fire. The new fire-line is obtained as the covering of the virtual fire-lines.

3. Cellular models based on Dijkstra's Algorithms. The terrain is divided in a discrete mesh of cells that are characterized by the average values of the parameters. From each cell, the fire can propagate to the neighbor cells. There are different models that consider different mesh geometry or different neighborhoods.

Once we have developed the mathematical model, we need to design and implement the simulator. In the next section we will describe how this is done.

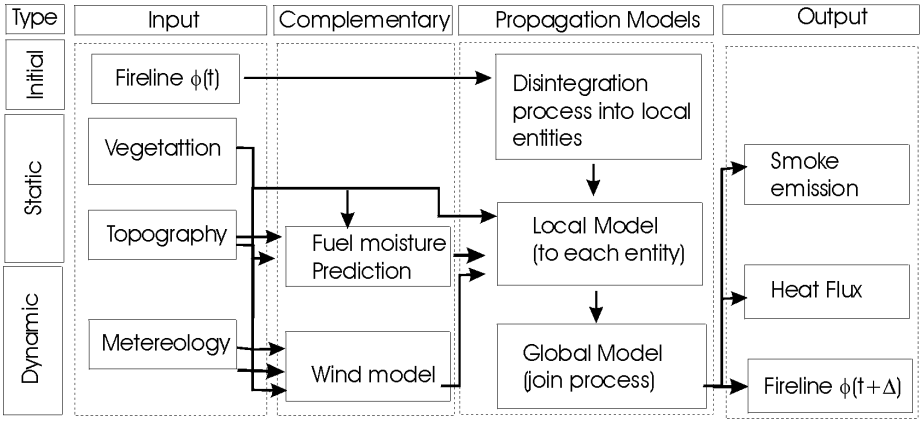

Figure 1.5: "Ideal" components of a fire simulation system

1.4.2 Simulator Software Development

The main goal of any implementation based on fire propagation models is to provide simulation tools that can be integrated into an information system that provides the user with an accurate information concerning forest fire propagation. This means that theoretical models developed by scientist must be coded and must run on a computer. To accomplish this objective, it is necessary to apply numerical methods and algorithms that solve the proposed models. This work implies direct co-operation between scientists and computer scientists.

The global and local models and the required environmental information must be integrated to obtain a simulation system that provides the space-time forest fire evolution. The general "ideal" structure of such systems is shown in figure 1.5[7].

The main components are the following:

1. Input information databases, concerning the physical environment, including: 1) Ignition point or current status of the fire-line, 2) vegetation maps that include the characteristics of the vegetation of each region, 3) topographic information of the terrain where the fire is burning, and 4) meteorological information, usually the wind field.

2. Propagation models: The global and local models described above.

3. Complementary models: These models include those parameters with a dynamic behavior.

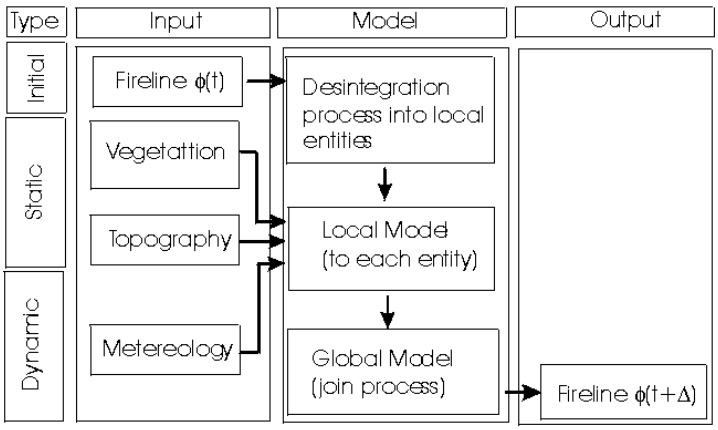

Figure 1.6: Components of a fire simulation system

4. Output: Mainly is the prediction of forest fire propagation, in addition to other outputs like heat flux and smoke emission.

The diagram at figure 1.5 includes all these components. However, the current state of forest fire research does not allow us to include all the components in a real system. There is active research in all these fields [25, 80, 106, 91, 110, 97, 40] but there are no final results that can be included in the simulation systems. Therefore, the real current simulation systems have a simplified structure (as that shown in figure 1.6). However, it must be pointed out that in the near future, research results will be introduced in these simulation systems.

1.4.3 Error Sources in the Simulation Software

Most of the existing wild-land fire models are not able to exactly predict real-fire spread behavior. Fire simulators [68, 74, 58, 35], which merely are a translation of the fire models' equations into code, cannot provide satisfactory fire spread predictions. The disagreement between real and simulated propagation basically arises because of :

- Uncertainties stemming from the respective mathematical models; scientists study fire behavior to create sophisticated fire models. A model is a simplified description of an actual system, useful for studying system behavior. The models attempt to simplify the phenomena without losing the main characteristics. Overly simple models can lose accuracy and

overly complicated models can be unpractical to use or even to realize. The models try to capture the main characteristics of the physical system or phenomena. No model will capture all the characteristics of the phenomena, if so it will be the system it self. It is important to understand what aspects of the system the model is intended to describe, and what the model limitations are as a result of simplification.

- Limitation of numerical solutions; Mathematical equations in the models are solved by numerical solutions. Usually, the mathematical models contain a complicated differential equations that need approximated numerical solutions to be solved. These methods have approximation errors (or *truncation errors*)[86]. The discrepancy between the true answer and the answer obtained in a practical calculation is called the *truncation error*. Truncation error would persist even on a hypothetical, "perfect" computer.

- Processor limitations; translating the mathematical models with their numerical solutions into code is done using a computer language. These languages and the underlying processors have numerical accuracy limitations. The representation of real numbers in digital machines and the ability of the processor to process the numbers have limits. Computers store numbers not with infinite precision but rather in some approximation that can be packed into a fixed number of bits. In floating-point representation, a number is represented internally by a sign bit, and exact integer exponent, and an exact positive integer mantissa. Arithmetic among numbers in floating-point representation is not exact, even if the operands happen to be exactly represented. The smallest (in magnitude) floating-point number which, when added to the floating-point number 1.0, produces a floating-point result different from 1.0 is termed the *machine accuracy*. Many arithmetic operations among floating numbers should be thought of as introducing an additional fractional error of at least equals to machine accuracy. This type of error is called *roundoff error*.

We can see that there will always be limitations on the simulators, so a perfect simulator is impossible to create. However, assuming that the simulator we have is good enough, we use it to predict the fire behavior. In the following section, we will describe the classical way of using fire simulators to predict fire behavior.

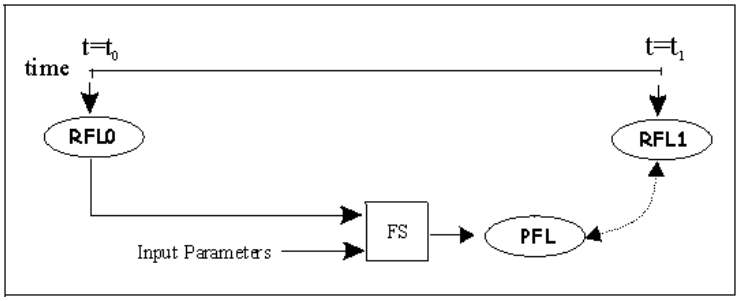

Figure 1.7: Classical prediction of wild-land fire propagation.

1.5 Classical Fire Prediction

Once we have developed the fire propagation model and created a simulator software based on the mathematical model, the resulting software can be used to predict the fire propagation patterns in a variety of conditions. Classical prediction simply consists of using any existing fire simulator to evaluate the fire position after a certain time. The simulator is fed with all the required parameters (vegetation, meteorological, ignition point ...etc). Then, the simulator is executed to predict the fire line after a certain period of time. We should point out that the simulator cannot be run with the absence of one of these input parameters.

This classical approach is depicted in figure 1.7 where FS corresponds to the underlying Fire Simulator, which is seen as a black-box. RFL0 is the real fire line at time t_0 (initial fire line), whereas RFL1 corresponds to the real fire line at time t_1. If the prediction process works, after executing FS (which should be fed with its input parameters and RFL0), the predicted/simulated fire line at time t1 (PFL) should concord with the real fire line (RFL1). However, this classical approach to predicting fire spreed has certain limitations. These limitations will be discussed in the following section.

1.6 Discussion of the Classical Prediction Problem: Thesis Contribution

The error in the classical approach of prediction not only stems from the simulator as described in section 1.4, but, even if we have a perfect simulator, we

still have difficulties to provide this simulator with quality input parameters. In the following subsection we will describe this difficulty.

1.6.1 Input Data Uncertainty

One of the most common sources of deviation from real propagation is imprecision in input parameters. There are certain parameters that cannot be measured directly, but this must be estimated from indirect measures (for example, moisture content in vegetation). Other parameters can be measured in certain particular points but the value of such parameters must then be interpolated to the whole terrain. For example, wind can be measured in certain meteorological stations (figure 1.8), but it is necessary to then estimate the wind for all the points within the terrain; therefore some sort of model is required to make this highly specific form of estimation. Moreover, the wind changes dynamically during forest-fire propagation due to changes in the meteorological conditions, so some model is required that will effectively predict the evolution of the wind. Other source of uncertainty and error in input parameters is that maps used (of vegetation characteristics and humidity) have finite cell size. It will be very costly to make maps with small cell size. So the simulator will need to work with parameters that are the average of large cell. In addition to that, the maps can be actualized in certain time periods, so the parameters fed to the simulator have to be measured at the past. For all these reasons, wind is usually estimated and predicted with a high degree of uncertainty.

Similarly, uncertainties in other input parameters can also have substantial impact on the result errors. For example, an error in the wind direction will affect the fire propagation direction. For this reason, we intend to search for parameters that, if fed to the simulator, will best describe the real fire line in the near past. We expect that these parameters will remain valid for a period of time (*predictability limit*), so we can use them to forecast within that time. This can be realized by defining a measurement of concordance between the real fire line and simulated. Then maximizing the concordance will provide us with the desired parameters. Therefore, we propose to add an optimization process on the classical prediction approach to tune input parameters. While the fire extends and time passes the parameters may change, which creates *a prediction error*. The error can be controlled by the continual assimilation of data, with intervals shorter than the predictability limit and continual execution of the optimization process.

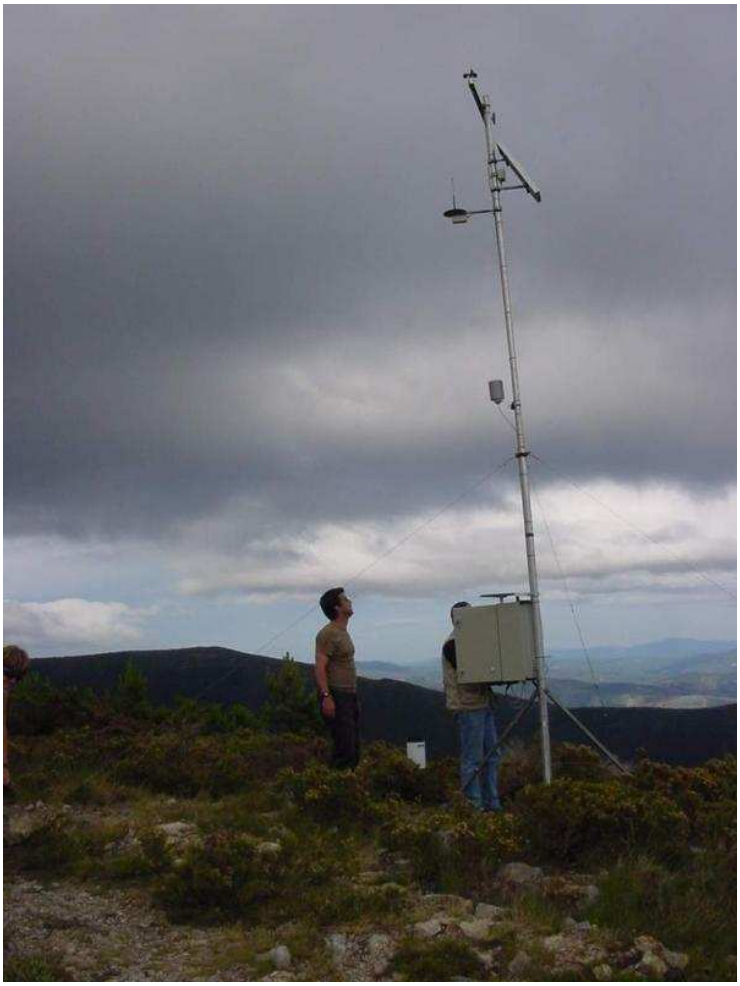

Figure 1.8: meteorological station

1.6.2 Real Time Constraint

In order to obtain operative predictions during a real fire emergency, this needs to be restricted by the real time constraint. Usually, simulators are complicated programs which consists of many calculations that need a great amount of computer power, so the proposed additional optimization process need to be finished in as few a number of executions as possible so as to minimize the need of computer power and the time of optimization. Due to the increasingly large size and inherent complexity of most man-made simulators, the problem to optimize here is an NP-complete problem, purely analytical means are often insufficient for optimization. We need to use modern heuristic optimization methods to search input parameters domains in the most efficient way.

The computing systems based on parallel and distributed systems offer the required power to apply these techniques and provide successful results in an acceptable time. Currently, distributed systems are widely available at a reasonable cost, and the software technology is mature enough to allow their intensive use.

On the one hand, using the classical approach of prediction with a light-weight simulator we can meet the real time constraint but we will loose the precision. On the other hand, optimizing the huge number of input parameters will be a time consuming problem even in a computer cluster. As in all decisions in life, and in engineering, there are some compromises. In this case, we have not just a compromise, but a suitable compromise. For instance, we can reduce the optimization time without loosing precision by fixing the parameters that have little impact on the simulation result to nominal or expected values and spend the time on a more productive issue, namely optimizing the most sensible parameters.

In order to apply this method to a fire emergency, we need to obtain the status of the fire in real time, assimilate this information and use it as a reference to calculate the *prediction error.* This information will then be used to optimize the input parameters of the fire spread simulation in order to enhance the prediction. The real fire line must be available in real-time and in machine recognizable format (a polygon in two dimensions or a set of burned cells). This requires an on-line forest fire (measurement) system(see figure 1.9). The system aims to use the images (for example taken from a helicopter), the GPS position of the camera, and information from a geographic Information System (GIS) to locate the fire and to estimate in real-time their properties. The system needs

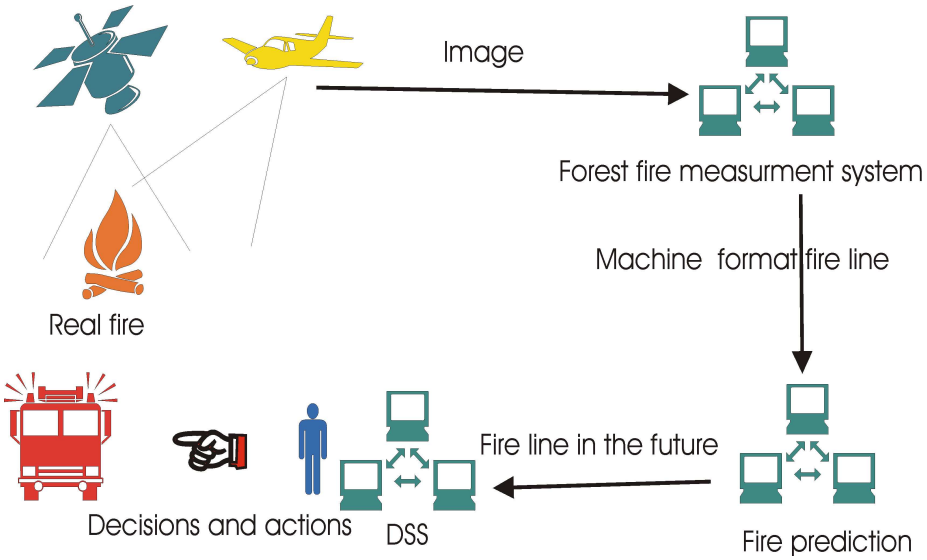

Figure 1.9: The relation of the fire prediction system with the other systems

to provide in real-time the evolution of the fire-front [78, 77, 70, 76].

After the delivery of the real fire line in the desired format, a prediction system has to start processing all the available data (the fire line, ignition line, vegetation, terrain, meteorological data) to provide an accurate prediction also under the real-time constraint. The prediction of the future behavior of the fire will be manipulated by humans and/or by a Decision Support System (DSS) to act against the hazard in the most proper way, to prevent accidents and to optimize resources.

1.7 Outline of Thesis

In the following chapter we introduce a theoretical description of the enhanced prediction method (Chapter 2). Then, we discuss the possible ways of parallelizing the method and a description of the implementation is stated in chapter 3. After that an illustration of experimental comparison of the implemented optimization techniques is reported in chapter 4. We discuss, in chapter 5, the theory and experiments of possible ways to accelerate the optimization methods. Finally, in chapter 6, an application of the complete methodology on real cases is included. In the following we describe in more details each chapter.

In chapter 2, we will describe a pragmatic approach that intended to improve the prediction quality of forest-fire simulators with the existence of all imperfections in real life (described in the introduction). As mentioned, enhanced prediction method is based on searching for values of input parameters that enhance the prediction of the simulators. Therefore, search methods occupy an important part of the Thesis. Thus, a theoretical discussion of search methods is introduced in chapter 2.

Chapter 3 discusses the way we have parallelized the method to reduce the time of execution and make it possible to execute the method in reasonable time. In addition, a full description of the implementation of the method is reported.

In chapter 4, we illustrate our experimental study to tune and compare several optimization techniques that could be used in the proposed methodology.

Chapter 5 describes the ways to accelerate the optimization method so that we can reach the optimal solution in less iteration and, therefore in less time. In the same chapter we illustrate the experimental study performed.

In chapter 6, we apply this methodology on real fire lines extracted from laboratory experiment, which were specifically designed to test our methodology.

Finally, in chapter 7, we address the main conclusions and propose future directions that can extend and enhance this research.

Chapter 2

Enhanced Prediction Method

"si tienes miedo, quítate de ahí, y
ponte en oración en el espacio que yo
voy a entrar con ellos en fiera y
desigual batalla."

Miguel de Cervantes

In this chapter, we will introduce a new methodology to enhance the prediction of the forest fire propagation in order to overcome the difficulties of obtaining precise input parameters. As has been mentioned, the methodology consists of optimizing a black-box function. We will therefore discuss the possible methods that can be used to approach such problems. For this purpose, the chapter is organized as follows: The chapter will start with a description of the enhanced methodology to predict forest fire propagation. Then, a formal description of the optimization process will be introduced followed by a discussion of the main parts of the optimization process. Finally, there is a discussion about optimization methods available in the literature.

2.1 Wild-land Fire Prediction

The prediction method proposed is a step forward with respect to the classical methodology described in section 1.5. The approach proposed focuses its effort on overcoming the input-parameter uncertainty problem. Using this approach the interaction of the end user with the simulation/prediction or (desision support system) DSS systems will be changed. The operator who uses the software

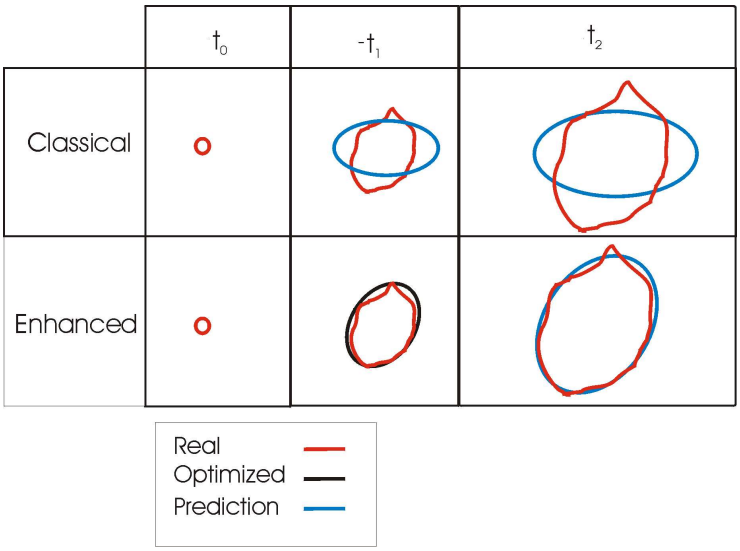

Figure 2.1: Readjusting the prediction using the optimized parameters

can provide ranges instead of a single guess for the values as input parameters. These ranges reflect the possible values of the parameters that the operator estimates. Our approach introduces the idea of applying an optimization scheme to calibrate the set of input parameter with the aim of finding an optimal set of inputs, thus improving the results provided by the fire-spread simulator. The aim of the optimization process is to find a set of input parameters that, if fed to the simulator, best describe fire behavior in the past. Therefore, we expect that the same set of parameters could be used to improve the prediction of fire behavior in the near future. We assumed that these parameters will remain valid for a period of time (*predictability limit*). In figure 2.1 we illustrate how the enhanced prediction works differently from the classical prediction. In the classical prediction we use some estimated parameters to predict the fire line in the future, but in the enhanced prediction we use the parameters of the optimized fire line (at time t_1) to predict the fire at time t_2. It is clear from the figure that we cannot use the enhanced prediction at time t_1. We need two time periods to carry out the prediction. One period is needed to find the optimized set of parameters. Then we use the optimized parameters to predict the fire line in the future. This process will be repeated to re-adjust the optimized parameters because the some parameters are very dynamic.

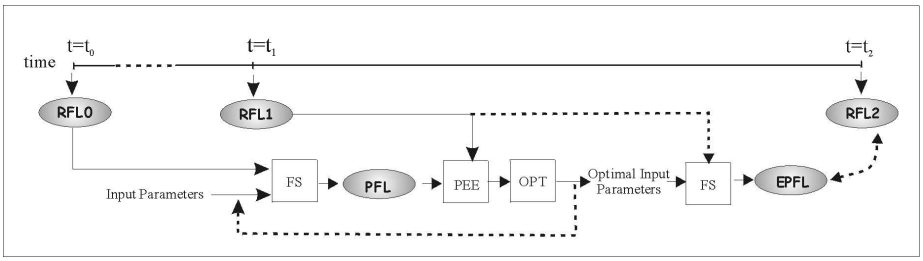

Figure 2.2: Enhanced wild-land fire prediction method

The resulting scheme is shown in figure 2.2. As we can see, in order to evaluate the goodness (concordance) of the results provided by the simulator, a prediction error estimation should be included (PEE box). This function will determine the degree of matching between the predicted fire line and the real fire line. We can also observe from figure 2.2 the way in which the optimization strategy is included into the framework to close a feedback loop. This loop will be repeated until a "good" solution is found or until a predetermined number of iterations has been reached. At that point, a "suboptimal" set of inputs should be found, which will be used as the input set for the fire simulator, in order to obtain the position of the fire front in the very near future (t_2 in figure 2.2). Obviously, this process will not be useful if the time incurred in optimizing is superior to the time interval between two consecutive updates of the real-fire spread information (for the example of figure 2.2, the interval time between t_1 and t_2). Consequently, we are interested in accelerating the optimization process as much as possible. Although similar ways of approaching this problem can be found in the literature [18, 26], no analysis of its applicability under real time constraints has been carried out.

Optimization is an iterative process that starts from an initial set of guesses for the input parameters and, at each iteration, generates an enhanced set of input parameter guesses. The optimization process will stop when a satisfactory solution is reached or the time of execution is spent. Typically, any optimization technique involves a large number of simulation executions, all of which usually require considerable time.

This process will be repeated when a new real fire line is optioned from the fire. The real fire line needs to be provided in a time interval less than the *predictability limit*. While the fire is burning, the process will be repeated to readjust the prediction and minimize the *prediction error*.

Since the fire simulator (FS) box of figure 2.2 has been widely analyzed in chapter 1, in the following sections, we will describe in more details the optimization (OPT) and prediction-error estimation (PEE) boxes shown in figure 2.2.

2.2 Optimization Problem

Formal optimization is associated with the specification of a mathematical objective function (called L) and a collection of parameters that should be adjusted (tuned) to optimize the objective function. This set of parameters is represented by a vector referred to as θ. Consequently, one can formulate an optimization problem as follows:

Find θ^* that optimizes $\underset{\theta \in S}{L(\theta)}$

where $L : R^p \rightarrow R^1$ represents some objective function to be minimized (or maximized). Furthermore, θ represents the vector of adjustable parameters (where $\theta^*$is a particular setting of θ), and $S \subseteq R^p$ represents a constrain set defining the allowable values for the θ parameters. Put simply, the optimization problem deals with the aim of defining a process to find a setting for the parameter vector θ, which provides the best value (minimum or maximum) for the objective function L. The term "search problem" and "optimization problem" are considered synonymous. The search for the best feasible solution is the optimization problem. This search is carried out according to certain restrictions of the values that each parameter can take. The whole range of possibilities that can be explored in obtaining the optimization goal is called the search space, which is referred to as S. In the following section, we describe how the objective function is particularized for our particular problem and certain key points to be considered with respect to the search space (S) defined by the fire problem.

2.2.1 Objective Function

As we mentioned in the previous section, we are interested in complex model optimization regardless of how the model itself works. Under this assumption, the underlying model/simulator is identified as a complex black-box function about which no information is provided. However, there is the possibility of measuring the quality of the results provided by the simulator for any input vector (θ). Consequently, the objective function (L) involves both executing

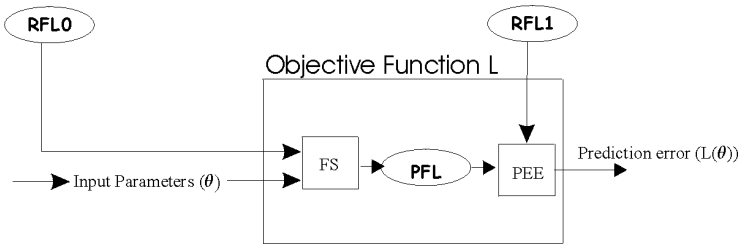

Figure 2.3: Objective function

the simulator and the evaluation of the quality of the results sequentially for each input vector θ. If we identify each one of these components in figure 2.2, we obtain the scheme depicted in figure 2.3. As we can observe, the objective function L corresponds to executes the underlying fire simulator once, followed by executing the prediction error estimator. The final value obtained is identified to $L(\theta)$ and its return is the value to minimize (*prediction error*).

Since our objective consists of finding the combination of input parameters that minimizes the deviation of the simulator prediction from the real scenario as fast as possible, we need to compare the simulated fire-lines against the real fire-line and, according to the results of this comparison, assign a quality measurement to the underlying scenario.

The objective function figure 2.3 takes a candidate solution (i.e a set of input parameters) as inputs. Then it executes the simulator using the input parameters and any inputs that we consider as fixed, such as the terrain. The obtained simulated fire line is compared with the real fire line and assigns a real value that describes the concordance (or disagreement) of the simulated fire line with the real fire line. This value is what we call the *prediction error*.

The prediction error estimator must be a function, whose value has to correctly respond to the following rules:

- give a zero prediction error value when the simulated fire line exactly concords with the real fire line and a positive value otherwise.
- better simulations should provide a smaller prediction error.

The fire lines are usually described in geometrical lines or as areas in two dimensional euclidean surface. Therefore, the prediction error will be a measurement made by matching two geometrical shapes.

2.2.2 Search Space

Fire simulators (as described above) needs many input parameters to be functionally usable. The simulator inputs are usually supplied as maps of vegetation and wind, which means that in each cell of the map there are several real valued parameters. As described in section 1.4, forest fire model parameters can be divided into three categories, meteorological, vegetation and topological. The topology is considered as static and can be obtained from the available maps. So we will not include the topology parameter in the optimization process.

The simplest case is when we have homogeneous vegetation and wind field. In these cases, the simulators need no less than 10 different parameters at each point. In mathematics, there is infinity number of possible solutions, but in computers, every thing is digital and finite, so, if we are going to implement some sort of algorithms to find the optimum, we have to consider the available computing precision, if we guarantee precession of six decimal places, each variable could then take on 10'000'000 different values [79]. Thus with only 10 parameters the search space will be 10E70 different combinations.

2.3 Optimization Taxonomy

Optimization problems are classified [111] according to the attribute variable type into continuous and discrete, and according to the presence of the constraints into unconstrained and constrained, and according to problem complexity into linear and nonlinear. The optimization problem we have can be classified as **a continuous unconstrained nonlinear optimization problem**. We will discuss the issues related with this type of problems in this thesis.

Figure 2.4 shows a taxonomy of optimization methods as focused in this thesis. The main criteria in classifying optimization strategies concerns the degree of analytical formality implied when developing them. On the one hand, analytical methods uses calculus as a tool to analyze the objective function $L(\theta)$. This method requires a well defined description of the objective function and its derivatives. On the other hand, non-analytical approaches can be further divided into two more categories: traditional optimization methods and modern heuristics. Within the traditional optimization method we can include for example, exhaustive search, and local search. Some representative examples of modern heuristics methods are genetic algorithm, simulated annealing and taboo search. The main difference between them is that the traditional meth-

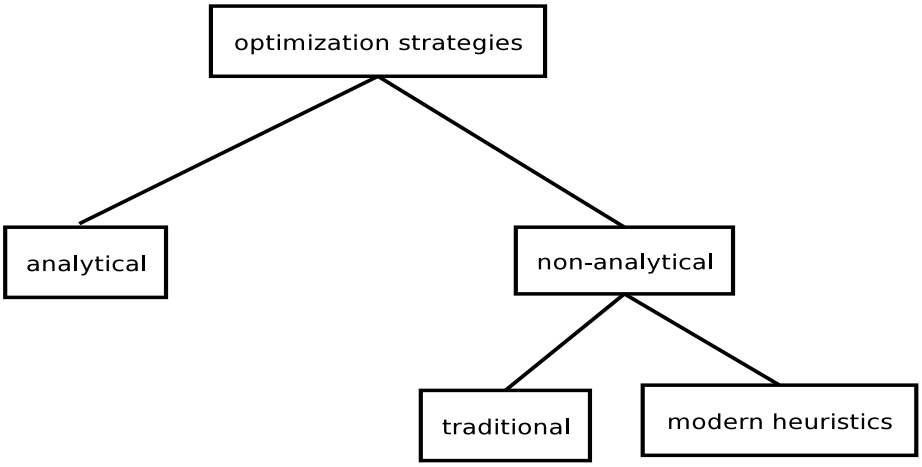

Figure 2.4: A classification of optimization methods

ods, although guarantee finding the global optimum, they are very expensive for solving typical real-world problems, and they have a tendency to get stuck in local optima.

2.4 Analytical Methods

The traditional analytical method is the method based on calculus derivation of the objective function. The method is designed to find local minima and maxima of a function using the following theory:

Theorem. Suppose that a is a local maximum or a local minimum of the function L. Then either $L'(a) = 0$ or $L'(a)$ does not exist.

Therefore, by finding the derivative of the function we can find all its maxima and minima. We then only need to search for the global optima throughout these points. Finding the derivative of the function in real-world problems is not an easy task. Problems, such as the problem we have, do not have explicit function definition but are a complex set of functions and algorithms that have many loops and if-statements inside. However, if we were able to estimate the corresponding function, it usually would have a complex derivative that will make it difficult to find its roots.

2.5 Traditional optimization Methods

There are many traditional algorithms designed to screen the search space for an optimum solution, but none of these traditional methods are robust. Every time the problem changes, the algorithm must also changes. The classical methods of optimization can be very effective when appropriate to the task in hand. It pays to know when and when not to use each one. As mentioned in section 2.3 optimization methods can be divided into two main categories: Those dealing with partial solutions and those dealing with complete solutions. Exhaustive search and local searches are examples of traditional methods that deal with complete solutions. Greedy algorithms, divide and conquer and branch and bound are examples of algorithms that deal with partial solutions.

2.5.1 Exhaustive Search

As the name implies, exhaustive search checks each and every solution in the search space until the best global solution has been found. This means if you do not know the value that corresponds to the evaluated worth of the best solution, there is no way to be sure that you have found the best solution using exhaustive search unless you examine everything. The size of the search space is enormous. If we look at the example in the section 2.2.2 we have 10e+70 possible solutions if we need 1 second for each objective function evaluation which implies execution of a fire simulator. We need 1e+71 seconds which is equal to 3.1709791984e+63 years of computation to reach the optimal solution.

2.5.2 Local Search

Instead of exhaustively searching the entire space of possible solutions, we might focus our attention within a local neighborhood of some particular solution. this procedure can be explicated in four steps:

1. Pick a solution from the search space and evaluate its merit. Define this as the current solution.

2. Apply a transformation to the current solution to generate a new solution and evaluate its merit.

3. If the new solution is better than the current solution then exchange it with the current solution; otherwise, discard the new solution.

4. repeat steps 2 and 3 until no transformation in the given set improves the current solution.

The key to understanding how this local search algorithm works lies in the type of transformation applied to the current solution. At one extreme, the transformation could be defined to return a potential solution from the search space selected uniformly at random. In this case, the current solution has no effect on the probabilities of selecting any new solution, and in fact the search becomes essentially enumerative. Actually, it is possible for this search to be even worse than enumeration because you might re-sample points that you have already tried. At the other extreme lies the transformation that always returns the current solution and this is not productive.

From a practical standpoint, the right thing to do lies somewhere in between these extremes. Searching within some local neighborhood of the current solution is a useful compromise. In that way, the current solution imposes a bias on where we can search next, and when we find something better we can update the current point to this new solution and retain what we learned. If the size of the neighborhood is very small, then we might be able to search that neighborhood very quickly, but we might also get trapped at a local optimum. In contrast, if the size of the neighborhood is very large then we have less chance of getting stuck, but the efficiency of the search may suffer. The type of transformation that we apply to the current solution determines the size of the neighborhood, therefore, we have to choose the transformation wisely in light of what we know about the evaluation function and our representation.

The majority of numerical optimization algorithms for NLP (Non-Linear Problems) are based on some sort of local search principle. However, there is quit a diversity of these methods. One reason that there are so many different approaches to NLP is that no single method is superior to all others. In general, it is impossible to develop a deterministic method for finding the best global solution in terms of L that would be better than exhaustive search.

Bracketing methods[79]. This method seeks to find the $L(\theta^*) = 0$. Suppose we know how to bracket θ^* with two other numbers a and b. Then one very simple way to find θ^* is the method of bisecting the range between a and b, finding this midpoint, m, determining the value of $L(m)$ then resetting the left of right limits of the range to m, depending on whether or not $L(m)$ is positive. If we continue to iterate this procedure, we eventually converge arbitrarily close to a value θ^* such that $L(\theta^*) = 0$.

Note that this bisection is a local search method. It relies on two bracketing points, a and b, but it still updates only one point at a time and has one current best solution at any iteration. There are many other similar methods that will converge much faster. For example *Regula Falsi,* which uses a secant line and find the point that intersects the x-axis. Also, *Newton's method* which uses the tangent line in the same manner.

Gradient methods.[79] The basic idea is that we need to find a directional derivative so that we can proceed in the direction of the steepest ascent or descent, depending on whether we are maximizing or minimizing. If the evaluation function is sufficiently smooth at the current candidate solution, the directional derivative exists. Our challenge is to find the angle around the current solution for which the magnitude of the derivative of the evaluation function with respect to some step size s is maximized. By applying calculus, we can derive that this maximum occurs in the direction of the negative gradient $-\nabla\mathcal{L}(\theta)$

The method of steepest descent, then, is essentially this: Start with a candidate solution θ_k, where $k = 1$. Then, generate a new solution using the update rule:

$$\theta_{k+1} = \theta_k - \nabla\alpha_k L(\theta_k),$$

where $k \geq 0, \nabla\mathcal{L}(\theta_k)$is the gradient at θ_k,and α_k is the step size. The bigger α_k is, the less local the search will be. The method is designed to ensure reductions in the evaluation function for small enough steps, so the idea is to find the right step size to guarantee the best rate of reduction in the evaluations of candidate solutions over several iterations.

2.5.3 Greedy Algorithms

Greedy algorithms attack a problem by constructing the complete solution in a series of steps. The reason for their popularity is simplicity. The general idea behind the greedy approach is to assign the values for all the decision variables one by one and, at every step, make the best available decision. Of course, this approach assumes a heuristic for decision-making that provides the best possible move at each step, the best "profit", thus the name greedy. However, it is clear that the approach is also shortsighted since taking the optimum decisions at each separate step does not always return the optimum solution overall. There really are no efficient greedy algorithms for NLP, but we can design an algorithm that displays some greedy characteristics. For example, to optimize a function

of, say, two variables, we could set one of the variables at a constant and vary the other until we reach an optimum. Then, while holding the new value of the second constant, we could vary the first until a new optimum is reached, and so on. Naturally, this line search can be generalized into n dimensions, but this process performs poorly if there are interaction between the variables.

Strictly speaking, line searches are not really greedy algorithms because they only evaluate complete solutions. Nevertheless, the intention of choosing the best available opportunity with respect to single dimension at a time does follow the general notion of greedy algorithms.

Greedy methods are conceptually simple, but they normally pay for that simplicity by failing to provide good solutions to complex problems with interacting parameters. The fire simulator optimization is a case of problems with interdependencies between the parameters.

2.5.4 Divide and Conquer

Sometimes it is a good idea to solve a seemingly complicated problem by breaking it up into smaller simple problems. You might be able to then solve each of those easier problems and find a way to assemble an overall answer out of each part. This "divide and conquer" approach is really only cost-effective if the time and effort required to complete the decomposition, solve all of the decomposed problems, and then reassembling an answer is less than the cost of solving the problem as it originally stands with all its inherent complexity. Also, you caution is required when you assemble the solution from the decomposed pieces, so you actually do obtain the answer you were looking for. Sometimes, the chance for assembling an overall solution disintegrates as you break the problem apart.

The original problem is replaced by a collection of subproblems, each of which is further decomposed into sub-subproblems, and so forth, often in a recursive manner. The process continues until the problems are reduced to being trivial.

2.5.5 Branch and Bound

Branch and bound is one such heuristic that works on the idea of successive partitioning of the search space. We first need some means for obtaining a lower bound on the cost for any particular solution (or an upper bound depending on whether we are minimizing or maximizing). The idea is then that if we have a solution with a cost of, say, c units, we know that the next solution to try has a lower bound that is greater than c, and we are minimizing, we don't have to

compute just how bad it actually is. We can forgot about that one and move on to the next possibility.

It helps to think about the search space as being organized like a tree. The heuristic of branch and bound prunes away that area of interest.

2.6 Modern Heuristic Optimization Methods

We have discussed a few traditional problem-solving strategies. Some of them guarantee finding the global solution, others do not, but they all share a common pattern. Either they guarantee discovering the global solution, but are too expensive (i.e., too time consuming) for solving typically real-world problems, or else they have a tendency of "getting stuck" in local optima. Since there is almost no chance to speedup algorithms that guarantee finding the global solution, i.e., there is almost no chance of finding polynomial-time algorithms for most real problems (as they tend to be NP-hard), the other remaining option aims at designing algorithms that are capable of escaping local optima.

2.6.1 Taboo Search

Taboo Search (TS) [44, 65, 66] is an iterative procedure, which was originally proposed for solving discrete combinatorial optimization problems. It was first suggested by Glover (1986) [42] and since then it has become increasingly used. It has been successfully applied to obtain optimal or sub-optimal solutions to such problems as scheduling, timetabling, traveling salesperson, and layout optimization.

Taboo Search is based on the premise that problem solving, in order to qualify as intelligent, must incorporate adaptive memory and responsive exploration. The adaptive memory feature of TS allows the implementation of procedures that are capable of searching the solution space economically and effectively.

The emphasis on responsive exploration in Taboo Search, whether in a deterministic or probabilistic implementation, derives from the supposition that a bad strategic choice can yield more information than a good random choice. In a system that uses memory, a bad choice based on strategy can provide useful clues about how the strategy may profitably be changed. (even in a space with significant randomness a purposeful design can be more adapt at uncovering the imprint of structure.)

Algorithm 1 Sketch of Taboo Search algorithm

TS Algorithm

```
k = 1 , Generate initial solution θ
WHILE not finished
  Identify N(θ) ∈ S  (Neighborhood set)
  Identify T(θ) ∈N(θ) (taboo set)
  Identify A(θ) ∈T(θ) (Aspiration set)
  Choose θ' ∈
N(θ) − T(θ)) ∪ A(θ), for which    L(θ') is maximal
  θ' ⇐ θ
  k= k+1.
END WHILE
```

Responsive exploration integrates the basic principle of intelligent search, i.e. exploiting good solution features while exploring new promising regions. taboo search is concerned with finding new and more effective ways of taking advantage of the mechanisms associated with both adaptive memory and responsive exploration. The development of new designs and strategic mixes makes TS a fertile area for research and empirical study.

Quoting Glover and Laguna (1993, p. 70)[43], "taboo search has its antecedents in methods designed to cross boundaries of feasibility or local optimality standardly treated as barriers, and to systematically impose and release constraints to permit exploration of otherwise forbidden regions".

TS begins in the same way as ordinary local or neighborhood search, proceeding iteratively from one solution (parameter vector) to another until a chosen termination criterion is satisfied. Each vector of the search space ($\theta \in S$) has an associated neighborhood $N(\theta) \subset S$, and each solution $\theta' \in N(\theta)$ is reached from θ by an operation called *move*. There is a set of k moves, $M = \{m_1, ..., m_k\}$, and the application of these moves to a feasible solution ($\theta \in S$) leads to k usually distinct, solutions $M(\theta) = \{m_1(\theta), ..., m_k(\theta\}$. The subset of feasible solution belonging to $m(\theta)$ is known as the neighborhood of $\theta(N(\theta))$ (see figure 2.5).

The method commences with a (possibly random) solution $\theta_0 \in S$ and determines a sequence of solution $\theta_0, \theta_1, ..., \theta_n \in S$. At each iteration θ_{j+1} is selected from the neighborhood $N(\theta_j)$ and the aspirant set $A(\theta_j) \subset T(\theta_j)$ of taboo neighbors. Then θ_{j+1} is the neighbor of θ_j which is either an aspirant or not taboo and for which $L(\theta_{j+1})$ is maximal; that is $L(\theta_{j+1}) \geq L(\theta')\forall\theta' \in N(\theta_j) - T(\theta_j) \cup A(\theta_j)$.

A sketch of the taboo Search algorithm is provided in algorithm box 1.

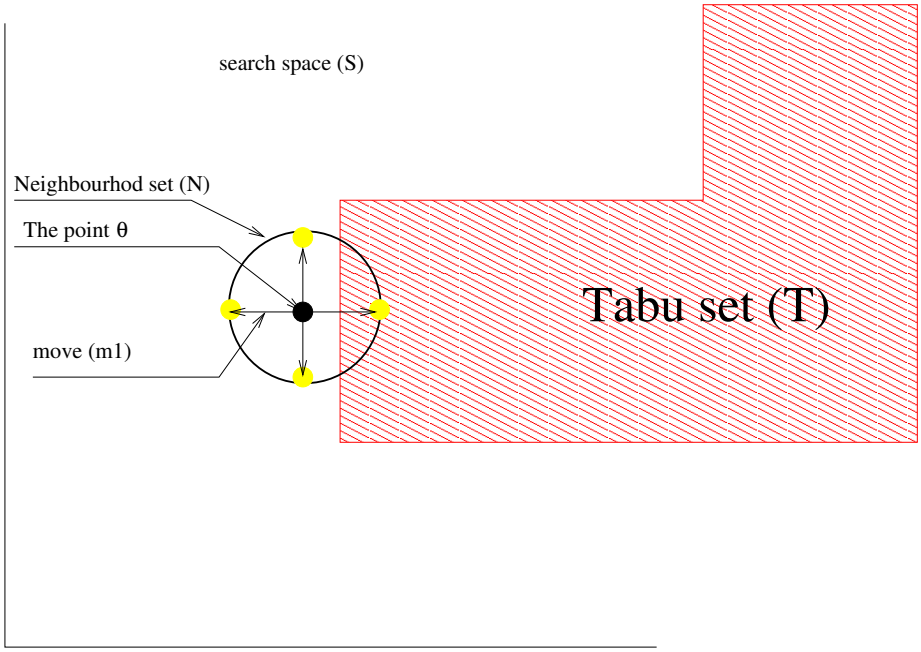

Figure 2.5: Tabu search

Note that it is possible, and even desirable, to avoid convergence at a local optimum, that $L(\theta_{j+1}) < L(\theta_j)$. The conditions for a neighborhood to be taboo or an aspirant will be problem-specific. For example, a move m_r may be taboo if it could lead to a solution which has already been considered in the last q iterations (regency or short-term condition) or which has been repeated many times before (frequency or long-term condition). A taboo move satisfies the aspiration criteria if, for example, the value of $L(\theta')$ with $\theta' \in T(\theta_j)$ satisfies $L(\theta') > L(\theta_i) \forall i, o \leq i \leq j$.

2.6.2 Simulated Annealing

Simulated Annealing is a stochastic search technique that avoids getting trapped in local optima by also accepting, in addition to transitions corresponding to an increase in function value, transitions corresponding to a decrease in function value. The latter is done in a limited way by means of a probabilistic acceptance criterion. In the course of the maximization process, the probability of accepting deteriorations in the course of the maximization process, the probability of accepting moving away from local optima and exploring the feasible region S in its entirety.

Simulated annealing originated from an analogy with the physical annealing process of finding low energy states of a solid in a heat bath. Pincus (1970)[84] developed an algorithm based on this analogy for solving discriminations of continuous global optimization problems. Many applications to date have been of continuous global optimization problems.

As with any search algorithm, simulated annealing requires the answers for the following problem-specific questions:

- What is a solution?
- What are the neighbors of a solution?
- What is the cost of a solution?
- How do we determine the initial solution?

These answers yield the structure of the search space together with the definition of neighborhood, the objective function, and the initial starting point. Note, however, simulated annealing also requires answers for additional questions:

- How do we determine the initial "temperature" T?

Algorithm 2 Sketch of Simulating Annealing algorithm

```
SA Algorithm
Step 0. Set n = 0, choose T_0 ∈ [0, ∞] and θ ∈ S
Step 1. Select Y_{n+1} according to
        the probability distribution R(θ_n, .).
Step 2. If L(θ_{n+1}) ≥ L(y_{n+1}), set θ_{n+1} ⇐ y_{n+1},
        If L(y_{n+1}) < L(θ_{n+1}), set θ_{n+1} ⇐ y_{n+1}
        with probability exp((L(y_{n+1}) − L(θ_{n+1}))/T_n).
         Otherwise, set θ_{n+1} ⇐ θ_n.
Step 3. Set T_{n+1} = g(T, n),
        increment n and return to Step1.
end.
```

- How do we determine the cooling ratio $g(T, n)$?
- How do we determine the termination condition?
- How do we determine the halting criterion?

The temperature T must be initialized before executing the procedure. Should we start with $T = 100$, $T = 1000$, or something else? how should we choose the termination condition after which the temperature is decreased and the annealing procedure reiterates? Should we execute certain number of iterations or should we use some other criterion instead? Then, how much or by what factor should the temperature be decreased? By one percent or less? When should the temperature be decreased? And finally, when should the algorithm halt, i.e., what is the "frozen" temperature?

Most implementations of simulated annealing follow a simple sequence of steps:

STEP 1: $T \Leftarrow T_{max}$
select θ_c at random

STEP 2:pick a point θ_n from the neighborhood of θ_c
if $L(\theta_n)$is better than $L(\theta_c)$
then select it $(\theta_n \leftarrow \theta_c)$
else select it with probability $exp(-\Delta L(\theta)/T)$
repeat this step $K_t$times

STEP 3: set $T \Leftarrow rT$
if $T \geq T_{min}$

then goto STEP 2
else goto STEP 1

Here we have to set the values of the parameters T_{max}, K_t, r, and T_{min}, which correspond to initial temperature, the number of iterations, the cooling ratio, and the frozen temperature, respectively.

In the area of numerical optimization, the issues of generating the initial solution, defining the neighborhood of a given point, and selecting particular neighbors are straightforward. The usual procedure employs a random start and Gaussian distributions for neighborhoods. Bit implementations differ in the methods for decreasing temperature, the termination condition, the halting condition, and the existence of a post-processing phase (e.g., where we might include a gradient-based method that would locate the local optimum quickly). Note that continuous domains also provide for an additional flexibility: the size of the neighborhood can decrease together with the temperature. If parameters σ_i(standard deviation) decrease over time, the search concentrates around he current point resulting in better fine tuning.

2.6.3 Genetic Algorithm

Genetic Algorithms (GA) are numerical optimization algorithms inspired by both natural selection and natural genetics. The method is a general one, capable of being applied to an extremely wide range of problems. Unlike some approaches, their promise has rarely been over-sold and they are used to help solve practical problems on a daily bases. The algorithms are simple to understand and the required computer code is easy to write.

In many ways, the thought of extending the concept of natural selection and natural genetics to other problems was tried from the very beginning. Computer scientists have visions of systems that mimicked one or more of the attributes of life. The idea of using a population of solutions to solve practical engineering optimization problems was considered several times in the 1950's and 1960's. However, GAs were in essence invented by one man - John Holland - in the 1960's. More recently others, for example De Jong, in a paper entitled "Genetic algorithms are NOT function Optimizers" [32], have been keen to remind us that GAs are potentially far more than just a robust method for estimating a series of unknown parameters within a model of a physical system. However in the context of this text, it is this robustness across many different practical optimization problems that concerns us most.

So what is a GA? A typical algorithm might consist of the following:

1. a set of guesses (or population) of the solution to the problem;
2. a way of calculating how good or bad the individual solutions within the population are;
3. a method of mixing fragments of the better solutions to form new, (on average better solutions); and
4. a mutation operator to avoid permanent loss of diversity within the solutions.

With typically so few components, it is possible to start getting the idea of just how simple it is to produce a GA to solve a specific problem. There is no complex mathematics, or this three are few hard and fast rules to what exactly a GA is.

Rather than starting from a single point (or guess) within the search space, GAs are initialized with a population of guesses. These are usually random and will be spread throughout the search space. A typical algorithm then uses three operators, selection, crossover and mutation (chosen in part by analogy with the natural world) to direct the population (over a series of time steps or generations) toward convergence at the global optimum.

Typically, these initial guesses are held as binary encoding (or strings) of the true variables, although an increasing number of GAs use "real-valued" (i.e. based-10) encoding, or encoding that have been chosen to mimic in some manner the natural data structure of the problem. This initial population is then processed by the three main operators.

Selection attempts to apply pressure upon the population in a manner similar to that of natural selection found in biological systems. Poorer performing individuals are weeded out and better performing, or fitter, individuals have a greater than average chance of promoting the information they contain within the next generation.

Crossover allows solutions to exchange information in a way similar to that used by a natural organism undergoing sexual reproduction. One method (termed single point crossover) is to choose pairs of individuals promoted by the selection operator, randomly choose a single locus (point) within the binary strings and swap all the information (digits) to the right of this locus between the two individuals (see figure 2.6).

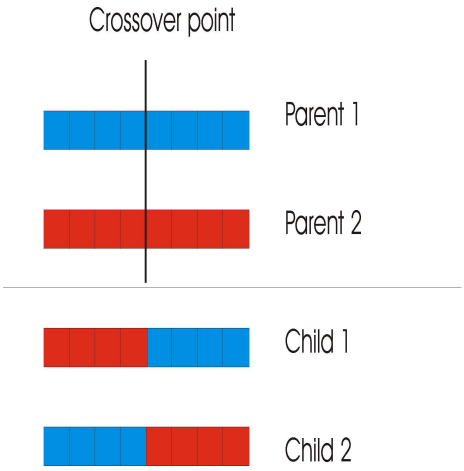

Figure 2.6: Crossover

Mutation is used to randomly change (flip) the value of single bits within individual strings. Mutation is typically used very sparingly.

After selection, crossover and mutation have been applied to initial population, a new population will have been formed and the generational counter is increased by one. This process of selection, crossover and mutation is continued until a fixed number of generations have elapsed or certain convergence criterion has been met.

On a first encounter, it is far from obvious that this process is ever likely to discover the global optimum, let alone from the basis of a general and highly effective search algorithm. However, the application of the technique to numerous problems across a wide diversity of fields has shown that it does exactly this. The ultimate proof of the utility of the approach possibly lies with the demonstrated success of life on earth.

From the above discussion, we have seen that Genetic Algorithms (GAs) seek to mimic natural evolution's ability to produce highly functional objects. Natural evolution produces organisms. Genetic algorithms produce sets of parameters, programs, molecular designs, and many other structures. Genetic algorithms usually solve problems by following the 4 steps shown in algorithm box 3.

A key issue is what constitutes better individuals or better chromosomes. This is determined by a fitness function. The fitness function takes a chromosome (set of genes) as input and returns a number representing the fitness of that

Algorithm 3 Sketch of Genetic Algorithm

1. Randomly generating the first population of individual potential solutions.

2. Evaluate the fitness function for each population member.

3. While not(an acceptable solution is found or exhaustion sets in) Obtain a new generation:

3.1 *Elitism*: best individuals are copied into the new generation.

repeat

3.2 *Selection*: select two "parent" individuals with a bias toward better individuals to produce children.

3.3 *Crossover*: each of two parents is divided into two parts and one part from each parent is combined into a child.

3.4 *Mutation:* a single "parent" is randomly modified to generate a child.

until a new population has been completed

end while

chromosome. The fitness values will be used to discriminate which members of the population will be considered/discarded to generate the new population. The genetic algorithm will be consecutively applied to obtain generations until an acceptable solution has been found.

Genetic algorithms differ in their representation of solutions. Bit string representations were used in the first genetic algorithms [56], but arrays of floating point numbers, special symbols that generate circuits [57], robot commands [59], and many other symbols may be found in the literature. For an excellent review of genetic algorithms and related techniques, see [12]. In our case, a chromosome is defined as the set of input parameters needed by the simulator to provide the output fire line. Since all involved parameters are represented by floating point numbers, we took these patterns as a solution representation for all elements of our chromosomes.

In the following section, we will discuss the our version of the heuristic techniques and how they can be fitted in one framework.

2.7 Evolutionary Methods Theory

In this section, we will introduce our vision of the heuristic algorithms. This vision will help us in the implementation. The main advantage of this approach is that it will not be necessary to implement different data structures and classes every time we want to implement different algorithm. According to this vision it will be easier to understand the implementation of the algorithms. The maintenance of the software will be more efficient. In future, it will be easier to implement other algorithms. The resulting software can be re-adapted to be used for purposes other that first intended purpose.

Evolutionary algorithms mimic the metaphor of natural biological evolution. Evolutionary algorithms operate on a population of potential solutions applying the principle of survival of the fittest to produce ever better approximations to a solution. At each generation, a new set of approximations is created by the process of selecting individuals according to their level of fitness in the problem domain and breeding them together using operators borrowed from natural genetics. This process leads to the evolution of populations of individuals that are better suited to their environment than the individuals that they were created from, just as in natural adaptation.

Evolutionary algorithms model natural processes, such as selection, recombination, mutation, migration, locality and neighborhood. Evolutionary algorithms work on populations of individuals instead of single solutions. In this way, the search is performed in a parallel manner.

From the above discussion, it can be seen that evolutionary algorithms differ substantially from more traditional search and optimization methods. The most significant differences are:

- Evolutionary algorithms search a population of points in parallel, not a single point.
- Evolutionary algorithms do not require derivative information or other auxiliary knowledge; only the objective function and corresponding fitness levels influence the directions of search.
- Evolutionary algorithms use probabilistic transition rules, not deterministic ones.
- Evolutionary algorithms are generally more straightforward to apply

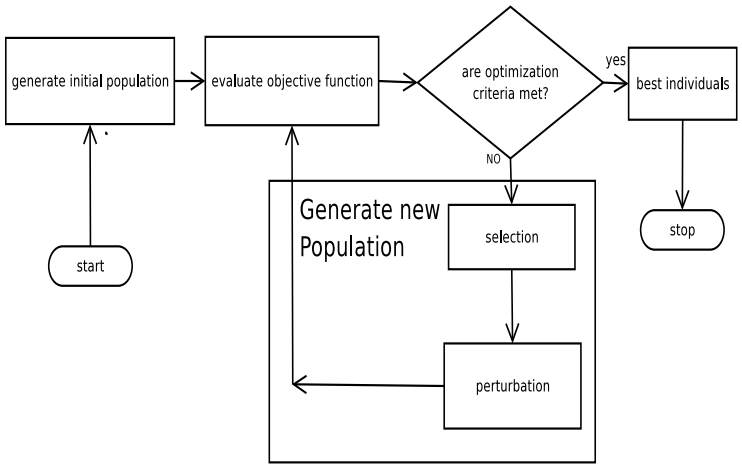

Figure 2.7: evolutionary algorithm

- Evolutionary algorithms can provide a number of potential solutions to a given problem. The final choice is left to the user. (Thus, in cases where the particular problem does not have one individual solution, for example, a family of Pareto-optimal solutions, as in the case of multi-objective optimization and scheduling problems, the evolutionary algorithm is then potentially useful for identifying these alternative solutions simultaneously.)

Figure 2.7 shows a simple sketch of the evolutionary algorithm. The main parts of the evolutionary algorithms are:

Selection where the algorithm chooses certain solutions in a stochastic way to use them as a base to create new solutions. The selection is biased toward good solutions. It is necessary some times to choose solutions that are not the best to avoid trapping in local minima.

Perturbation this part is applying some operations to introduce new solutions, these solutions are usually from the neighborhood of the current solutions.

Genetic algorithm, taboo search and simulated annealing can be adapted to the evolutionary approach. Genetic algorithm by nature works on a population of individuals at a time. The GA uses mutation and crossover for this purpose. The selection in the genetic algorithm depends on the competition among the candidate solutions. The perturbation is mutation and crossover. Where muta-

tion is some kind of searching the neighborhood of the solution and crossover is combining more than one solution to create some others.

Simulated annealing also can fit in this framework. The original SA algorithm explores the search space by examining one solution at a time, but it can be adapted to fit the evolutionary approach by exploring a set of neighbors at a time. This set of neighbors is analogical to the current generation in the genetic algorithm. Selection in SA depends on choosing better solution, if not found it may choose the next solution according to decreasing probability. It selects a solution from the neighborhood of the current solution. If in the neighborhood there is a better solution it selects it, otherwise, it selects another solution according to decreasing probability. The creation of the neighborhoods can be seen as an operator of perturbation. It is analogical to mutation in the genetic approach.

In the case of taboo search, it searches in the neighborhood as with simulated annealing, but the selection is different. It uses a memory instead of probability to avoid local minima. Selection in TS depends on memory. It uses experience from the previous iterations. Perturbation is similar to the case of SA, it is creating the neighborhood and it also may use cross over. In all the three cases we can use the same perturbation operators.

These methods participate the same idea of exploring not only the best solution to avoid trapping in local minima. All of these methods have algorithm parameters that need to be tuned to make a trade-off between exploring the entire search space and, at the same time, concentrate on the promising regions. For instance, in the genetic search we can modify the mutation and crossover probabilities. We can also change the strategy of the selection to give more stress on the good solutions or to relax the selection to explore more regions. Some ideas from simulated annealing can also be used in GA such as changing the mutation probability or mutation radius during the iterations. In the case of simulated annealing, we can do the same by choosing the starting and ending temperature and the cooling strategy. The size of the taboo memory and the taboo moves do the same in the case of taboo search.

In summary, the three modern heuristic approaches can be seen as one iterative search approach that have as input a population of solutions and as output another population of solutions, and it uses selection and perturbation operators to explore the search space. Taking in account exploring all the search space and at the same time concentrating in the neighborhood of the good solutions.

Table 2.1 summarizes the differences between the three algorithms. The se-

	Genetic Algorithm	taboo search	Simulated Annealing
selection	competence between solutions	according the memory	decreasing probability
perturbation	mutation, crossover	neighborhood	neighborhood

Table 2.1: evolutionary approach

lection in the genetic algorithm depends on the competence between the current generation of the individuals: better individuals have more chance to reproduce new solutions. In the case of taboo search, the memory decides the next region to explore. The solutions that have been explored will not be chosen. Simulated annealing starts as random search choosing any solution from the neighborhood, then it restricts the selection to better solutions at the end of the iterations.

2.8 Chapter Conclusions

In this chapter we have developed a methodology to enhance the prediction methods. We have theoretically shown that heuristic optimization methods are strong candidates to be used in the enhanced prediction method. At the end of the chapter, we developed a theory that can help in simplifying the implementation and parallelization of the heuristic optimization methods. In the next chapter, we show how to parallelize the proposed methodology.

Chapter 3

Parallelizing the Enhanced Prediction Method.

"Pues aunque mováis más brazos que
los del gigante Briareo, que lo habéis
de pagar. "

Miguel de Cervantes

As we have previously commented, the main goal of this work consists of carrying out parameter optimization and prediction faster than real fire evolution so that the prediction can be useful in deciding which actions need to be taken in tackling the emergency. Consequently, the response time of the prediction framework (simulator plus optimization) will be bounded by hard real time constraints. The response time of the program must be faster than the real fire. Although modern heuristic optimization techniques may reduce the time of search, we still could make the execution time faster. The computational time consumed for optimizing process clearly depends on two elements: the function to be optimized (objective function), which involves the execution of the underlying simulator, and the number of times the optimization process should be iterated to converge. In this chapter, we focus on how to overcome these two sources of prediction delay by taking advantage of the computing systems based on parallel and distributed systems. Distributed systems are currently widely available at a reasonable cost, and the software technology is mature enough

to allow their intensive use. Therefore, we have parallelized the proposed enhanced prediction method in order to reduce the execution time involved in each iteration of the whole process.

In the following sections, we will start with the analysis of different parallel programing paradigms. Then, we will describe the parallel enhanced methodology.

3.1 Parallel Programming Paradigms

It has been widely recognized that parallel applications can be classified into some well defined programming paradigms. A few programming paradigms are used repeatedly to develop many parallel programs. Each paradigm is a class of algorithms that have the same control structure [53].

Experience to date suggests that there are a relatively small number of paradigms underlying most parallel programs. The choice of paradigm is determined by the available parallel computing resources and by the type of parallelism inherent in the problem. Computing resources may define the level of granularity that can be efficiently supported on the system. The type of parallelism reflects the structure of either the application or the data and both types may exist in different parts of the same application. Parallelism arising from the structure of the application is named as functional parallelism. In this case, different parts of the program can perform different tasks in a concurrent and cooperative manner. But parallelism may also be found in the structure of the data. This type of parallelism allows the execution of parallel processes with identical operation but on different parts of data.

There is a wide range of parallel program paradigm classification but the following are the most popular used in parallel programming [22]: Master/Worker (Task-Farming), Single Program Multiple Data (SPMD), Data Pipelining, Divide and Conquer and Speculative Parallelism [22].

In the following subsections, we briefly describe each one of these parallel programming paradigms.

3.1.1 Master/Worker

A master-worker application consists of two entities: a master and multiple workers. The master is responsible for decomposing the problem into small tasks and distributes these tasks among a farm of worker processes, as well

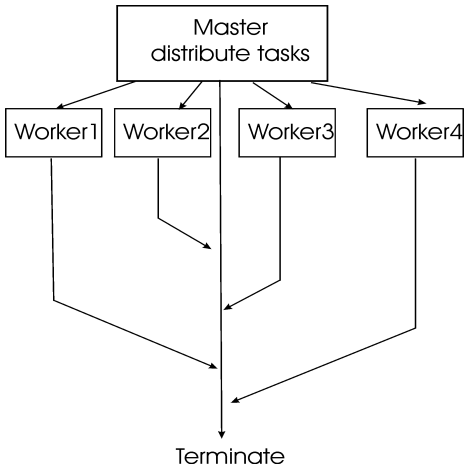

Figure 3.1: A master/worker structure

as for gathering the partial results in order to produce the final result of the computation. The worker processes receive a message from the master with the next task, process the task, and send the results to the master. The master process may carry out some computations while tasks of a given batch are being completed. After that, a new batch of tasks is assigned to the master and this process is repeated several times until completion of the problem (after K cycles or iterations). Figure 3.1 shows how a Master-Worker application proceeds.

3.1.2 Single Program Multiple Data (SPMD)

In the SPMD paradigm each process executes basically the same piece of code but on a different part of data. This involves splitting data among the available processors. This type of parallelism is also referred to as geometric parallelism, domain decomposition, or data parallelism.

Many physical problems have an underlying regular geometric structure, with spatially limited interactions. This homogeneity allows the data to be distributed uniformly across the processors, where each one will be responsible for a defined spatial area. Processors communicate with neighboring processors and the communication load will be proportional to the size of the boundary of the element (see figure 3.2). SPMD can be very efficient if the data is well distributed by the processes and the system is homogeneous. This paradigm is very sensitive to the loss of some process. Usually, the loss of a single process

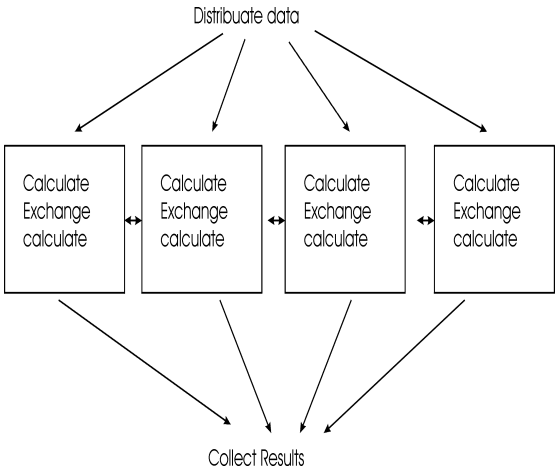

Figure 3.2: Basic structure of a SPMD program.

is enough to cause a deadlock in the calculation in which none of the processes can advance beyond a global synchronization point.

3.1.3 Data Pipelining

This program paradigm is based on a functional decomposition approach: the tasks of the algorithm, which are capable of concurrent operation, are identified and each processor executes a small part of the total algorithm. Figure 3.3 presents the structure of this model. processes are organized in a pipeline, each process corresponds to a stage of the pipeline and is responsible for a particular task. The communication pattern can be very simple since the data flows between the adjacent stages of the pipeline. For this reason, this type of parallelism is also sometimes referred to as data flow parallelism. The communication may be completely asynchronous. The efficiency of this paradigm is directly dependent on the ability to balance the load across the stages of the pipeline. The robustness of this paradigm against reconfigurations of the system can be achieved by providing multiple independent paths across the stages. This paradigm is often used in data reduction or image processing applications.

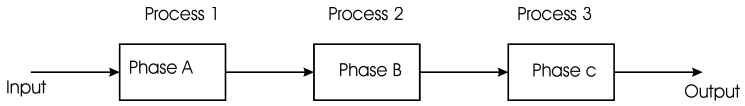

Figure 3.3: Data pipelined structure

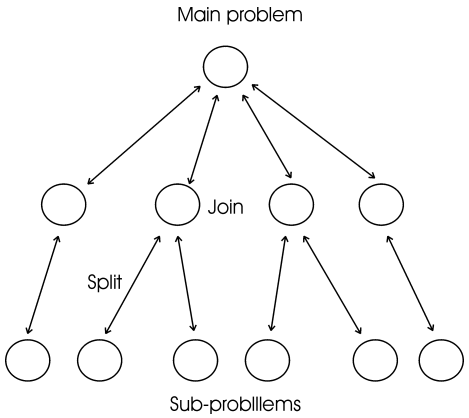

Figure 3.4: Divide and conquer as a virtual tree.

3.1.4 Divide and Conquer

Under this program paradigm, a problem is divided into two or more subproblems. Each of these problems is solved independently and their results are combined to give the final result. Often, the smaller problems are just smaller instances of the original problem, giving rise to recursive solutions. One can identify three generic computational operations for divide and conquer: split, compute and join. The application is organized in a sort of virtual tree, as is depicted in figure 3.4

3.1.5 Speculative Parallelism

This paradigm is employed when it is quite difficult to obtain parallelism through anyone of the previous paradigms. Some problems have complex data dependencies, which reduces the possibilities of exploiting the parallel execution. In these cases, an appropriate solution is to execute the problem in small parts but use some speculation or optimistic execution to facilitate the parallelism.

3.2 Parallel Enhanced Prediction Approach

Recalling the optimization approach described in section 2.7, the method depends on using several candidate solutions to create new solutions using an optimization method. The optimization method depends on the values of objective function of each solution to create new set of solutions. So we need to evaluate the objective function for each candidate solution before feeding it to the optimization method. While the evaluation of the objective function for the candidate solutions is independent in one iteration, we can evaluate it in parallel (see figure 3.5). We can assign for each available processor a candidate solution to execute the objective function, then we need to collect these solutions. Once the objective function is calculated for all the candidate solutions of the current iteration, the optimizer function is called to process them and create a new set of candidate solutions. Once more, the cycle starts again by distributing the new set of solutions to the farm of processors.

This process can be described as iterative master/worker. In addition to the master task of distributing and collecting the tasks to and from workers it executes the optimization algorithm, and, obviously, the workers will execute the prediction error, which consists of executing the simulator and comparison method (see section 2.2.1).

As described earlier the objective function consists of executing a fire simulator in addition of prediction error evaluation. This indicates that the objective function will hold many execution times. So, we expect that this approach to parallelize the method would have a good scalability because it has a reasonable granularity and a great portion of the execution time can be parallelized.

3.3 Black-Box Optimization Framework (BBOF)

In this section, we describe the implementation of the optimization framework called BBOF (Black-Box Optimization Framework), which works in an iterative fashion, where it moves step-by-step from an initial set of guesses to a final value that is expected to be closer to the true (optimal vector of parameters) than the initial guesses. This goal is achieved because, at each iteration of this process, a preset optimization technique is applied to generate a new set of guesses, which should be better than the previous one.

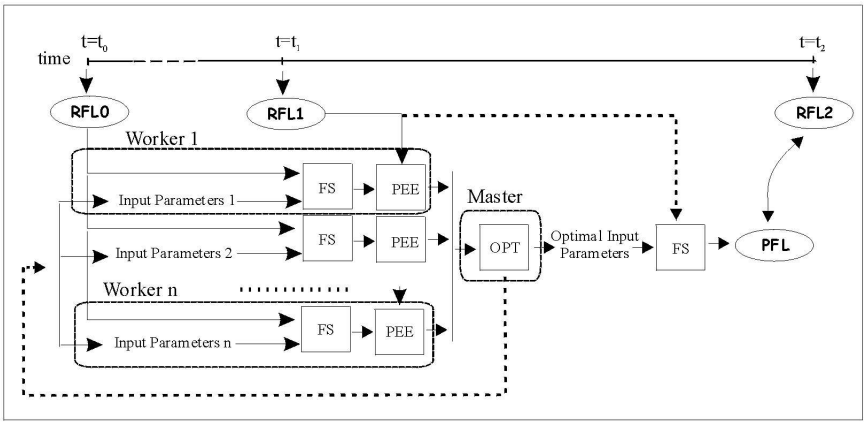

Figure 3.5: Parallel enhanced prediction approach

3.3.1 BBOF - Design Issues

Basically, we can decompose the implemented OF (Optimization Framework) into two separate blocks: the optimizable block and the optimizer block. We focus now on optimizable blocks. Optimizable blocks can be identified with black-boxes objective functions due to the complexity of the objective function. Objective functions usually consists of several iterative solutions, table selections and if conditions. This feature makes it impossible to reduce it to a simple mathematical expression and to know its derivatives, furthermore, the implemented optimizer blocks seeks a global optimum.

Since the two above-mentioned blocks are independent of each other, we can experiment in a "plug&play" fashion with different complex objective functions and global optimization techniques at the optimizable and optimizer blocks, respectively.

This basic architecture of our OF system is shown in figure 3.6. This system works in an iterative way, where starting from a set of guesses of the input parameters (vector) needed by the black-box function, we can obtain a set of results that allow the optimizer block to generate a new set of vectors of parameters. In other words, this process proceed with the following iterative steps:

1. Create a set of initial vectors randomly or distributed around some expected vector of parameters.

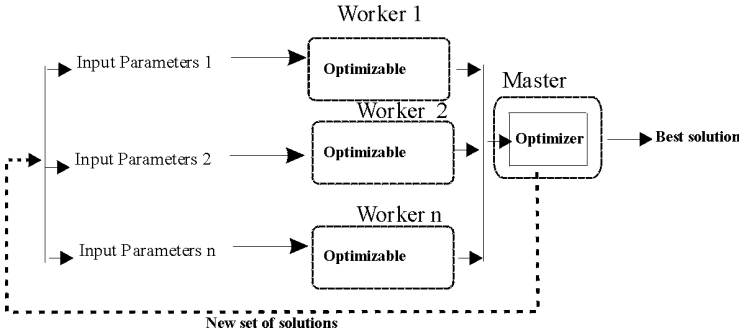

Figure 3.6: Master and worker process interaction.

2. Evaluate the objective function of the given vectors.

3. Apply a method (optimization technique) to move from one set of vectors to a better set.

4. Repeat 2 and 3 until satisfying exit conditions. (Number of iterations, accepted value ... etc).

The black box can be any system that has a vector of parameters as input and one value or more as output to be optimized. The method to generate a better set of vectors for next iteration can be a range of methods such as genetic, taboo, random, analytical etc... Since this process has an inherent parallel structure, the implemented OF tool has been developed taking advantage of existing middle-ware tools, such as MPI, PVM ..etc.

We can easily match each element of this master-worker scheme with the main components of the iterative optimization framework BBOF. In particular, since the evaluation of the objective function for each guess is independent of each other, they can be identified as the work (tasks) done by the workers. The responsibility for collecting all the results from the different workers, and for generating the next set of guesses by applying a given optimization technique, will be concentrated on the master process.

For the master-worker communication purpose, BBOF uses the MPI (Message Passing Interface) library. BBOF is a set of C++ abstract-based classes that must be re-implemented in order to fit the particular function to be optimized and the specific optimization technique. The most relevant classes within BBOF are the following:

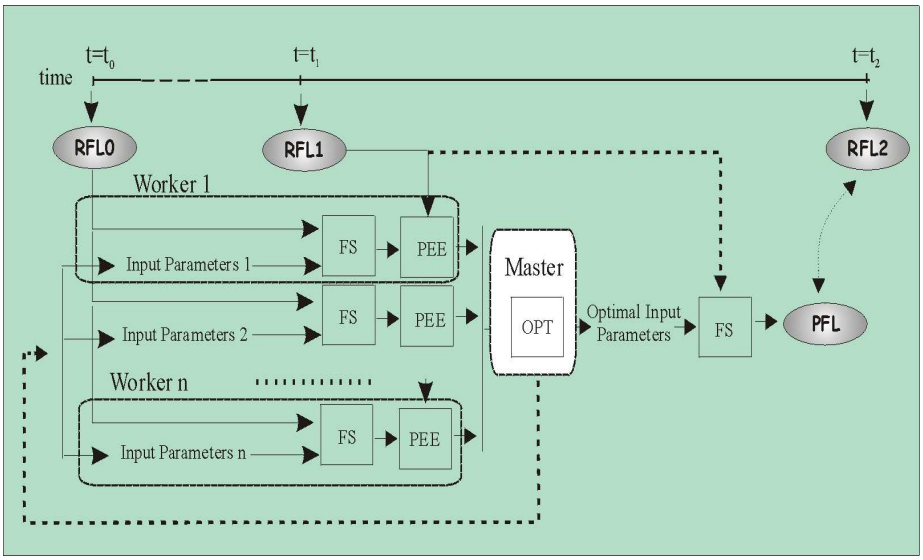

Figure 3.7: Master

BBOF_objective_function: this class corresponds to the objective function which is going to be optimized. This is the only function that the user has to change if he wants to optimize his own objective function. It takes an individual, evaluates the objective function according to the parameter vector it consists, and modifies the individual to hold the objective function evaluation result. This class executes on the worker.

BBOF_guess: this class is identified with a vector of real numbers and the value of the objective function. In other words, this class represents the information related to the tasks to be executed by the workers.

BBOF_master_set: is a set of BBOF_guess classes (vector guesses) and the current iteration number. One important method within this class is the method denoted by **BBOF_optimization_technique**, which is the optimization technique to be applied. This class keeps the best individual and it reports makes report about the current set of guesses (such as average, standard deviation ...etc).

3.3.2 BBOF - The Master Process

As we have commented in our case the master process deals with the execution of the selected optimization technique. In the following subsection we describes in detail each one of the implemented optimization strategies.

3.3.2.1 Classical Optimization Techniques

Mathematical-Statistical To implement an analytical approach we need information about the derivatives of the function that we want to optimize. In this work, we assumed that we only have the system as a black-box and we do not know anything about the function except that we can feed some parameters and get a result. To overcome this problem, we propose finding an approximation of the systems. We create a model of the response surface by means of statistics. In this work we have implemented a parabolic function that is smooth and has one global minimum and it is easy to derivate. It can be written formally as the following

$$f(X) = \beta_0 + \beta_{11}x_1 + \beta_{12}x_1^2 + ... + \beta_{1n}x_1^n + ... + \beta_{p1}x_p + \beta_{p2}x_p^2 ... + \beta_{pm}x_p^n$$

where f is approximation of L.

In order to find the optimum of this function using calculus, we need to derive it for each variable x_i and to find the zero of the derivative. The formula describing this derivation form and the equation we should solve is the following:

$$\begin{aligned} f'(X) &= 0 \\ \frac{\partial f(X)}{\partial x_i} &= \beta_{i1} + 2\beta_{i2}x_i + ... + n\beta_{i2}x_i^{n-1} = 0, \forall i \in \{1..p\} \end{aligned}$$

To find the coefficients of the approximated function, we evaluate a large set of vectors to obtain a significant set of values, which allow us to better obtain the function. Subsequently, we apply the multiple linear regressions on this set to find the coefficients of the function.

This approach can help us, if and only if, the cost function can be approximated to a good precision to some function that is calculus friendly. In this case, the BBOF only performs one iteration starting from a wide set of parameters

Algorithm 4 Analytical

1- Evaluate a large set of randomly generated vectors.
2- Consider the input as function parameter and their elevation to the power from 1 to n as variables of the function X where
3- consider the output of black-box as the result of the function (Y vector).
4- Compute the fitted function of some degree (n) using multiple regrision.
5- Get partial derivatives as illustrated above.
6- Find the root of the partial derivatives from 4.
Evaluate the roots that have been calculated from 5.

vectors. The obtained results from the evaluation of the objective function for all these vectors are used to obtain the parabolic mathematical expression of the function. The steps shown in algorithm box 4.

To calculate the fitted function we used the function from the GSL (GNU Scientific Library) library, `gsl_fit_linear` that computes the best-fit parameters c of the model $y = Xc$ for the observations y and the matrix of prediction variables X. The best-fit is found by singular-value decomposition of the matrix X. Any components which have a zero singular value (to machine precision) are discarded from the fit. The formulation can be used to fit to any number of functions and/or variables by preparing the matrix X appropriately. To fit to $n - th$ order polynomial in x, we used the following matrix

$$X_{ij} = X_i^j$$

where $i = 0...n$ for all the n parameters we have.

After finding the best-fit parameters and formulating the derivative parameters by the formula above, we pass these parameters to the function in the same library to solve the roots of the functions as follows:

1. For the function in the order of 2, we used the following formula to calculate the best parameter.
$$x_i^* = \frac{-\beta_{i1}}{2\beta_{i2}}$$

2. For the function of the order 3, we have derivatives as quadratic equations and solve them using the GSL function `gsl_poly_solver_quadratic`,

this function finds the root of the quadratic equation,

$$\begin{aligned} ax^2 + bx + c &= 0 \\ a &= 3\beta_{i3} \\ b &= 2\beta_{i2} \\ c &= \beta_{i1} \end{aligned}$$

the number of real roots (either zero or two) is returned. The number of roots found depends on the sign of the discriminate. This will be subject to rounding and cancellation errors when computed in double precision, and will also be subject to errors if the coefficients of the polynomial are inexact. These errors may cause a discrete change in the number of roots. However, for polynomials with small integer coefficients, the discriminate can always be computed exactly.

In the case of polynomial of order greater than 3, we use the general polynomial solver `gsl_poly_complex_solver` this function computes the root of the general polynomial

$$\begin{aligned} P(x) &= a_0 + a_1 x + a_2 x^2 + ... + a_{n-1} x - 1 \\ a_i &= (i+1)\beta_{pi+1} \end{aligned}$$

Using balanced-QR [39] reduction of the companion. The function uses an iterative method to find the approximate location of roots of higher order polynomials.

Random We implement Random Search by implementing the procedure that creates a new set vector (OFNewSet) so that it takes the old set of solutions and searches in it for one vector that provides the objective function value better than the one reached so far, if it finds one that satisfies this criteria it changes the best so far by that one. Then it creates another set from the same size randomly, by generating a random number for each parameter within the range of that parameter. It also keeps the best vector reached so far and reports it in the output report. The random function is equally distributed.

3.3.2.2 Modern Heuristic Optimization Techniques

Genetic Recalling from section 2.6.3 that GAs are characterized for emulating natural evolution, Therefore, under GAs scenarios, we refers to a vector of parameters as chromosome, a gene is identified with one parameter and the population is the set of vectors (chromosome) used for one iteration of the algorithm. Furthermore, four transmissions operators are defined: elitism, selection, crossover and mutation, whose particular implementation in our approach is now described:

Elitism: The two best solutions to the problem discovered are copied to the next generation. We choose two because our implementation of crossover needs two parents to produce two children and the population size is assumed even. It can be implemented by copying one elite individual and the population need to be odd. Or by producing one offspring from two parents by crossover. Our implementation maintains the same number of vectors in each generation.

Selection: We implement the biased selection (or fitness-proportional, or roulette wheel) where the individual with greater fitness has more probability of being chosen. For this purpose, a random number between zero and the summation of all fitness is generated. Then, the fitness of individuals are added in turn and when the partial sum equals or exceeds the random number previously obtained, the corresponding individual is selected.

Crossover: In particular, we implement arithmetic crossover. Crossover proceeds by cutting the pair of parents obtained by the selection operation at a random locus (picked by throwing a random number between 0 to the length of the chromosome). The two obtained parts of the chromosome will each be equally copied to one of two children in the same locus as they occupied in the parent chromosome. Furthermore, the remaining parameters (genes) of each child will be obtained by evaluating the average corresponding locus values in the parents' genes. Formally, for the first child

$$\begin{aligned} \theta_{c1i} &= \begin{cases} \theta_{p1i} & i < crosspoint \\ \frac{\theta_{p1i}+\theta_{p2i}}{2} & i \geq crosspoint \end{cases} \\ \theta_{c2i} &= \begin{cases} \frac{\theta_{p1i}+\theta_{p2i}}{2} & i < crosspoint \\ \theta_{p2i} & i \geq crosspoint \end{cases} \quad \forall i = 1...n \end{aligned}$$

where θ_{c1i} is the parameter i of the first child, θ_{c2i} is the parameter i of the second child,

θ_{p1i} is the parameter i of the first parent, θ_{p2i} is the parameter i of the second parent, and n is the number of vectors parameters. Crossover is not necessarily implemented to each individual, it can be subject to some probability, as in the case of mutation, but the probability of crossover is usually large. A random number is generated between 0 and 1000 if the number is less than the probability of crossover, the algorithm applied the crossover, otherwise the individual is copied to the next generation and it may suffer some mutation. Typical probabilities of crossover are 0.4 to 0.9. In our implementation we use more probability it reaches to 0.99.

Mutation: It can be defined as changing some aspects of the solution. For every parameter there is the same probability to be mutated. We have implemented several types of mutation. The first method is to add (or subtract) a small number to (from) the selected parameter. Therefore we apply the formula $\theta_i = \theta \pm \varepsilon$ where ϵ is a small real number which can be called the mutation distance. The other method is to change the selected parameter to another value randomly generated. The new value can be generated from the range of available values of the parameter using a random distribution function. The function can equally be distribution or any other distribution like normal distribution. Using equally distribution the new value do not depends on the current value, but in normal distribution the new value depends on the current value by using the current value as the mean.

Experiments of genetic algorithms (detailed in the next chapter) show that the progress is not linear. Initial progress is rapid, however this progress is not maintained. If early during a run one particularly fit individual is produced, fitness proportional selection can allow a large number of copies to rapidly flood the subsequent generations. Although this will give rapid convergence, the convergence could quite possibly be erroneous or only to a local optimum. Furthermore, during the later stages of the execution, when many of the individuals will be similar, fitness proportional selection will pick approximately equal numbers of individuals from the range of fitness present in the population. Thus, there will be little pressure distinguishing between the good and the very good individuals.

What is needed is a method whereby particularly good individuals can be stopped from running away with the population in the earlier stages, yet a degree of selection pressure maintained in the final stages. This can be achieved by various mechanisms one being the use of linear fitness scaling.

Linear fitness scaling works by pivoting the fitness of the population members around the average population fitness. This allows an approximately constant proportion of copies of the best individuals to be selected compared with average individuals. To achieve this, the fitness of every population member will have to undergo a scaling just before selection. This scaling needs to be dynamic. The fitness will need to be drawn closer together during the initial stages and pulled apart during the later generations. The required scaling is achieved using the linear transformation:

$$f_i^s(g) = a(g) f_i(g + b(g))$$

where f_i is the true fitness of an individual, i, and f_i^s the scaled fitness. As already stated, the mean fitness of the population is assumed to remain unchanged, so:

$$f_{max}^s(g) = c(g) f_{avg}(g)$$

where f_{max}^s is the scaled fitness of the best individual.

This implies that:

$$\begin{aligned} a(g) &= \frac{(c-a) f_{ave}(g)}{f_{max}(g)} - f_{ave}(g) \\ b(g) &= (1 - a(g)) f_{ave}(g) \end{aligned}$$

In the program, setting c to zero stops the scaling process.

Typical values for this constant, c, are in the range 1.0 to 2.0. When $c = 2$, the chance of best individuals to go to the next generation is twice the chance of average individuals to go to the next generation.

Taboo Search Technique As has been previously mentioned, taboo Algorithm was originally developed for discrete optimization. We propose a continuous alternative implementation which has been implemented in our BBOF system. Let us analyze in detail how each one of the parameters involved in the Taboo Algorithm has been individually implemented, starting by the move

definition.

Move Definition We define the move in the real numbers optimization as a determined way of changing the vector.

We have a deterministic move implementation which is adding or subtracting certain real number (ϵ_i) from one parameter of the vector so that the count of available moves is double the count of the parameters. If (ϵ_i) is large, the algorithm will not reach an acceptable vector and if (ϵ_i) is small the algorithm spends a large number of iterations to reach the optimal or sub-optimal vector. So the number (ϵ_i) needs to be large at the beginning and when we find the optimal of that precision we need to decline (ϵ_i) to give more accurate results. The determination of (ϵ_i) depends on the type of the parameter, its range, the precision needed on this parameter and its contribution in yield of the system. This method adds another set of parameters to the algorithms parameters that need to be set by the user.

An alternative way to define the move is changing one parameter upward or downward by randomly choosing a value that is greater than the current value and less than or equal to the upper limit of the available parameter range. It is similar to the previous one, but instead of defining a determined move distance we use a random distance. We call this *move* as indeterministic move. This move adds a random factor to the algorithm.

Neighborhood Definition In the context of move definition, the Neighborhood of a vector is applying all available moves on the vector (see figure 3.8).

Taboo Set Definition We use two types of taboo time: long time and short time. The long time keeps a set of vectors that have been visited and chosen as selected during preceding iterations. This set is implemented as queue (first in first out) and it has a predetermined length (LL). During the navigation in the search space oldest vector that has been visited before LL iterations goes out and new vector comes in the list. Long-term taboo time avoids looping in the search space.

The short-term taboo time is to mark as taboo for some iterations. The move that have been made recently and/or its anti-move (anti-move means the move that is given moving at the opposite direction on certain parameter so

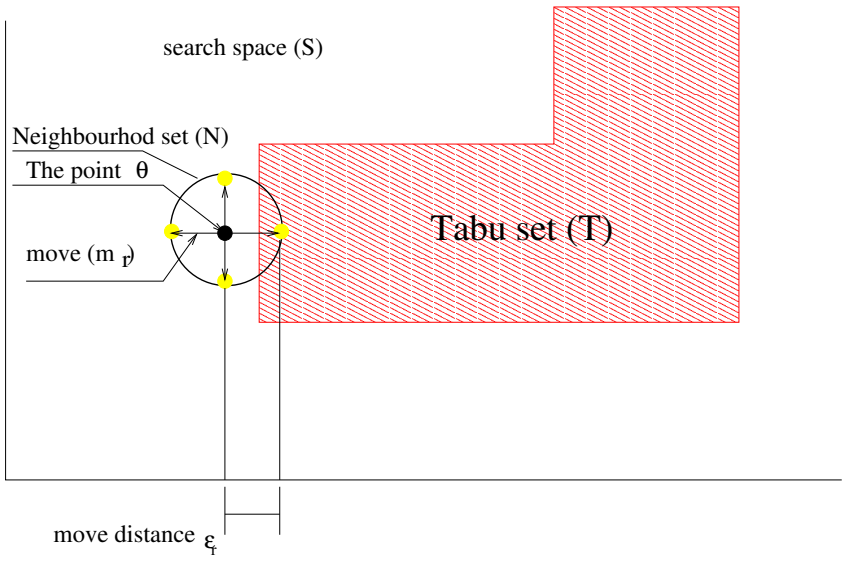

Figure 3.8: taboo Neighborhood definition with two variables

that it may cancel the effect of the move i.e.; for example if the move is adding to θ_i the anti-move is subtracting from θ_i).

Aspiration Set We simply use the criteria that if the solution is better than the best so far, the algorithm chooses it without checking if it is taboo or not.

Frequency We keep track of the number of times some move have been carried out so that we can avoid concentrating on the same parameter and discount others. The frequency counter can influence the sorting of neighbors so that those neighbors generated using less frequent moves will have more probability. This process helps in exploring the search space.

It is clear from this that the algorithm needs to evaluate the neighbors of a point and these evaluations are independent. So the algorithm fits in a master-worker paradigm and BBOF. The master will choose the next point by means of taboo algorithm and keeps taboo memory structures. Workers evaluate the objective function. To increase the number of workers, we can find the neighbors of the current neighbors in advance, and evaluate them, and even enter them in the process of choosing the next point (widen the neighborhood). In the case of

Algorithm 5 Sketch of Simulating Annealing Algorithm

```
Randomly generating the first set of n guesses, set T= T0.
Evaluate the objective function for each guess
initialize C to a vector from the first set.
While not (stop criteria).
Generate a new set of guesses:
for each guess V
     calculate dE = L(V) - L(C)
     if (dE<0) or(random<exp(dE/T))
         change C with V
     else set V = C.
calculate neighbors of the set.
Evaluate the objective function
         for each vector within the new set
Reduce T.
end while.
```

undetermined moves, simply create more random neighbors.

Simulated Annealing We have adapted simulated annealing to our framework. The sketch version of the algorithm is shown as algorithm 5. Note that the inner 'while' loops to the current neighborhood set using the current temperature. The algorithm generates new set of solutions by substituting better solutions with old solutions or by using probability depending on the current T to allow worst solutions in the new set. The probability of substituting solutions depends on the difference of the two objective function values and the probability. If the solution has better objective function it substitutes the current solution, otherwise it substitutes the current solution according to decreasing probability.

Neighbors in any vicinity are calculated by randomly changing the value of one parameter. The SA algorithm, which is outlined in algorithm box 5, is based on a value referred to as temperature (T). T is initialized to a certain value T_0 and is gradually reduced until a certain T_n is reached. In our particular implementation of the SA algorithm, T_0 and T_n have been set to 1000 and 10, respectively. The function that indicates the reducing *(cooling) factor* of T_0 is as follows:

$$T = T_0 - \frac{k(T_0 - T_n)}{n}$$

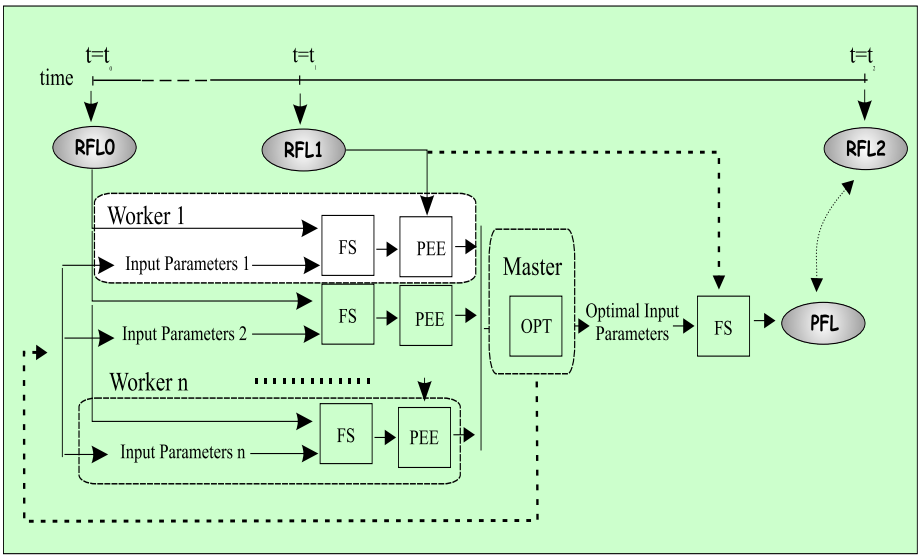

Figure 3.9: Worker

where i is the current iteration number and n the maximum number of iterations over which this process will be repeated. When dE (the difference between the objective function value of the current state and the candidate state) is less than zero, the algorithm accepts the new state; otherwise, if the random number is less than $e^{dE/T}$, a new random number is generated within the range [0,1]. Temperature (T) determines the tolerance of accepting other vectors (states) that have higher energy (higher objective function); when the temperature is high, there is more possibility of accepting a vector. i.e., if a given vector exhibits a higher objective function and the temperature is lower, there will be less tolerance to changing the current vector. In fact, temperature plays a crucial role in the algorithm.

3.3.3 BBOF - The Worker Process

Figure 3.9 shows the BBOF parts belonging to the worker process. As we can see, the worker will execute a fire simulator (FS box) followed by a Prediction Evolution Error (PEE box).

If we consider one dimensional prediction simulator, obviously the prediction error will be the scalar difference between the simulated and real fire propagation speeds, but this is not our case. We have treated two dimensional fire simulators.

These simulators produce as output a two dimensional map describing the fire line. Each fire-line describes a burned area. We have used two simple simulators: the first is called ISS[58] and the second is an example that uses Fire-Lib.

ISS has in its core Rothermel's set of equations to calculate the speed of propagation of each segment of the fire line and treatestreats the fire line as a polygon that moves and changes in time. ISS has as output a set of points describing the burned area as a polygon.

We have also used fire-Lib, Which is a C-language function library for predicting the spread rate, intensity, flame length, and scorch height of free-burning surface fires. It is derived directly from the BEHAVE [8] fire behavior algorithms for predicting fire spread in two dimensions, but is optimized for highly iterative applications such as cell- or wave- based fire growth simulation. fire-Lib was developed to give fire growth modelers a simple, common, and optimized application programming interface to use in their simulations. While fireLib contains 13 functions, as few as 4 function calls are required to create a simple yet efficient and functional fire growth simulator. Fire-Lib was written by Collin D. Bevins, Systems for Environmental Management. Program development was funded cooperatively by the Fire Behavior Research Work Unit of the USDA Forest Service Rocky Mountain Research Station, and by Systems for Environmental Management. With the distribution of the fire-Lib, there is a simple simulator example that uses cells as input and output. With minor changes, we have used this simulator to make some of our experiments. This simulator is light and is suitable for the applications that have time restriction, so we used this simulator to make our real case experiments.

To compare the simulated and real fire-lines, we used two ways to compare the predicted and simulated fire line: (1) Hausdorff distance and (2) XOR area.

Hausdorff Distance [89] is used in the literature for pattern recognition. It is simple to apply when we have the fire lines as two sets of points. The Hausdorff distance measures the degree of mismatch between two sets of points by measuring the distance of a point of one set that is farthest away from any point of the other, and vice versa. Formally, the directed Hausdorff distance h between two sets of points M and I at a specific point in I is:

$$h(M, I) = max_{m \in M}(min_{i \in I}(distance(m, i)))$$

Thus, the Hausdorff distance H is defined as:

$$H(M,I) = max(h(M,I), h(I,M))$$

XOR Area The other measurement is the area of the XOR between the real and simulated burned areas. This XOR includes the areas that are burned in one of the propagations but not in the other one. If there is a perfect match between the real and simulated fire, this XOR is zero. When the result of the XOR is greater, this means that there are burned areas that were not predicted, or that unburned areas were expected to burn. The greater the XOR, the worse the prediction of propagation. This area can be used as a measurement of the prediction error.

To calculate the XOR area we first create a polygon that describes the XOR and then calculate its area. We have used the "General Polygon Clipping library" version 2.31 by Alan Mutra (gpc)[81] to create the XOR polygon. gpc is a C library implementation of a polygon clipping algorithm. The techniques used are descended from Vatti's polygon clipping method [105] . Subject and clip polygons may be convex or concave, self-intersecting, contain holes, or be comprised of several disjoint contours. It extends the Vatti algorithm to allow horizontal edges in the source polygons, and to handle coincident edges in a robust manner. Four types of clipping operation are supported: intersection, exclusive-or, union or difference of subject and clip polygons. The output may take the form of polygon outlines or tristrips. After using this library to create the XOR of the two fire lines we use the following formula to calculate the area of the resulting polygon:

$$A = \frac{1}{2} \sum_{i=0}^{N-a} (x_i y_{i+1} - x_{i+1} y_i)$$

Considering a polygon made up of line segments between N vertexes $(x_i, y_i), i = 0...N-1$ The last vertex $(x_N, y_N)$is assumed to be the same as the first. We calculate the holes in the polygon in the same way, then subtract them from the result.

Taking into account the particular implementation of the FireLib simulator, where the terrain is treated as a square matrix of terrain cells. We define the prediction error as the number of cells that are burned in the simulation but are not burned in the real fire map, and vice versa. This expression is the XOR function. Figure 3.10 shows an example of how this prediction error is evaluated

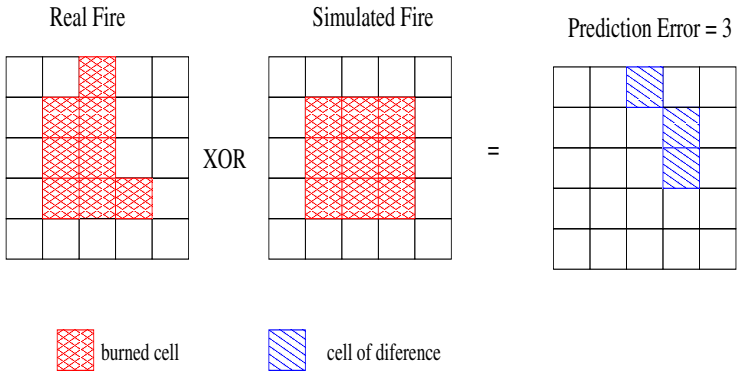

Figure 3.10: Evaluation of the XOR function as a fitness function for a 5x5 cell terrain.

for a 5x5 cell terrain. The obtained value will be referred to as the prediction error that, for this example, is 3.

3.4 Chapter Conclusions

In this chapter we have shown that a suitable way to parallelize the prediction method is master/worker paradigm. We have described the implemented BBOF that have been used to build the enhanced prediction method. We stated issue on adaptation of optimization methods to the problem. Some possible implementations of the objective function have also been described. In the next chapter we will tune the implemented optimization methods and make some comparison studies to fined the points of weakness and strangeness of these techniques.