

Between classical and ideal: enhancing wildland fire prediction using cluster computing

Baker Abdalhaq · Ana Cortés · Tomàs Margalef · Germán Bianchini · Emilio Luque

Received: 1 September 2003 / Revised: 1 March 2004 / Accepted: 1 May 2004
© Springer Science + Business Media, LLC 2006

Abstract One of the challenges still open to wildland fire simulators is the capacity of working under real-time constraints with the aim of providing fire spread predictions that could be useful in fire mitigation interventions. We propose going one step beyond the classical wildland fire prediction by linking evolutionary optimization strategies to the traditional scheme with the aim of emulating an “ideal” fire propagation model as much as possible. In order to accelerate the fire prediction, this enhanced prediction scheme has been developed in a fashion on a Linux cluster using MPI. Furthermore, a sensitivity analysis has been carried out to determine the input parameters that we can fix to their typical values in order to reduce the search-space involved in the optimization process and, therefore, accelerates the whole prediction strategy.

Keywords Fire prediction · Evolutionary optimization · Cluster computing

B. Abdalhaq (✉) · A. Cortés · T. Margalef · G. Bianchini · E. Luque
Computer Science Department, ETSE, Universitat Autònoma de Barcelona 08193-Bellaterra (Barcelona), Spain
e-mail: baker@aows10.uab.es

A. Cortés
e-mail: ana.cortes@uab.es

T. Margalef
e-mail: tomas.margalef@uab.es

G. Bianchini
e-mail: german@aows10.uab.es

E. Luque
e-mail: emilio.luque@uab.es

1 Introduction

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 extremely dangerous. Every year intensive forest fires burn thousands of hectares destroying a great number of trees and natural resources. Moreover, this implies a progressive turning of land into desert with all the associated problems.

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. This fight against the fire must be carried out at two main different levels. These two levels are:

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 the 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 roads and ways of reaching all the places in the country and so on. If the administrative decisions are made in the proper way, the effects of fire can be minimized and the fight against 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, this fighting against the fires implies the coordination of several groups (plane and helicopter pilots, firemen, and volunteers) for the adequate use of combative resources.

In both cases of fire prevention and real emergency, it is important to have tools that are able to predict the forest-fire propagation taking into account the particular conditions.

Models are being increasingly used to make predictions in many fields of environmental science. Models, however, have a series of limitations from which the lack of accuracy and/or a proper validation have the most dramatic consequences.

One of the goals of fire-model developers is to provide physical or heuristic rules that can explain and emulate fire behavior and, consequently, that might be applied to creating robust prediction and prevention tools. However, as knowledge of several factors affecting fire spread is limited, most of the existing wildland fire models are unable to exactly predict real-fire spread behavior. Fire simulators [2, 4–6], which only are a translation of the fire model equations into code, cannot therefore provide good fire spread predictions. The disagreement between real and simulated propagation basically arises because of:

- uncertainties stemming from the respective mathematical models,
- approximations in numerical solutions,
- processor limitations and
- the difficulty of providing the model with accurate input values.

There are two different philosophies to approaching the challenge of developing fire-propagation models. On the one hand, certain scientists study fire behavior to create sophisticated fire models in order to “exactly” (“ideally”) reproduce the real fire behavior. However, overly complicated models can be impractical to use or even to realize. On the other hand, there are certain trends to simplify the phenomena without losing the main characteristics. The drawback, in this case, is that excessively simple models can lose accuracy.

Mathematical models usually contain complicated differential equations that need approximated numerical solutions to be solved. Translating these numerical solutions into code is carried out 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 processors to process these numbers have limits [22].

Classical prediction simply consists of using any existing fire simulator to evaluate fire position after a certain time. The simulator is fed with all the required parameters (vegetation, meteorological information, ignition point ...etc.) and then, the simulator is executed to predict the fire line after certain period of time. The simulator cannot be run with the absence of one of these input parameters. The prediction obtaining using this approach usually disagrees to the reality. As we have previously commented, one reason of the disagreement between real and simulated propagation stems from the difficulty of providing the model with accurate input values.

Uncertainties in the input variables needed by the fire propagation models can have a substantial impact on the result errors and have to be considered.

A way to overcome this problem consists of applying optimization techniques to the input parameters of the simulator/model, with the aim of finding an input setting so that the predicted fire propagation matches the real fire propagation. Due to the increasingly large size and inherent complexity of most man-made simulators, purely analytical means are often insufficient for optimization [1]. We need to use heuristic optimization methods to search input parameters domains in most efficient way.

Additionally, we should bear in mind that, for a fire prediction be useful, it should provide a new fire-line situation within a time frame that is substantially before that of the real time being predicted. Otherwise, the prediction will be of no value. Therefore, fire prediction can be seen as a process under real-time constraints.

Approaching this problem using the classical prediction approach we can meet the real-time constraints in detriment to the precision results. However, this kind of environmental risks need reliable predictions in order to tackle the emergency in a productive way. Thus, we propose going one step beyond the classical wildland fire prediction by linking evolutionary optimization strategies to the traditional scheme with the aim of emulating an “ideal” fire propagation model as much as possible. Similar ways of approaching this problem can be found in the literature [8, 9], however, no analysis of its applicability under real time constraints has been carried out.

The current state of the art in the computational field offers the required background to be applied to approach this problem. Computing systems based on parallel and distributed platforms (cluster computing) offer the required computing power to apply these techniques and to provide successful results in an acceptable time. However, since evolutionary-optimization techniques work in an iterative way by improving the obtained solution at each iteration, they could be a time consuming problem even using clusters computing. Therefore, techniques to improve the convergence speed of the optimization technique should be provided. In this work, we apply a sensitivity analysis to the input parameters in order to assess their impact on output and, consequently, to determine which parameters are worth spending time on tuning and which are better to avoid spending effort on, maintaining them instead at their typical values.

The rest of the paper is organized as follows. In Section 2, we briefly reported the fire modeling problem. The classical and the proposed enhanced prediction methods are described in Section 3. How to speed up the fire prediction process taking advantage of cluster computing and techniques for accelerating the optimization are outlined in Sections 4 and 5 respectively. The framework used for the experimental study

is reported in Section 7 is set in Section 6. Finally, the main conclusions are listed in Section 8.

2 Fire simulation

It should be recalled 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 the fire propagation in a direct way. Temperature, wind, moisture, and so on modify the fire behavior and propagation in a significant way. It must be taken into account that these conditions are not static, but change due to the macro-meteorological conditions or the day-night cycle. Therefore, the forest-fire propagation prediction should consider the 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: The features of the vegetation influence in a direct way the fire behaves. 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 contents depends on meteorological conditions (past and current).
3. Topographical aspects: The topographical aspects of the terrain are also very significant in order to predict the fire behavior. But the particular topographical conditions also modify the meteorological conditions. For example, meteorological wind is modified by the topography of the terrain in such a way that the particular wind must be evaluated at each point and therefore, it must be analyzed as a wind field with a particular value at each point.

For all these reasons it can be concluded that the forest-fire propagation prediction is a very complex problem that involves several disciplines (physics, chemistry, biology, ecology, ...) that must co-operate to provide accurate models and solutions that predict the fire propagation in a realistic way.

There are several models in the literature to describe the behavior of forest fire propagation, which can be divided into two main categories: global and local models. These two models consider two different scales. On the one hand, the global models consider 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. Most of these models have in their core the basic fire propagation equations

Table 1 Input parameters for the Rothermel model

Parameter	Unit
Loading (l)	lb fuel/sq ft bed
Dead fuel moisture (dm)	lb water/lb fuel
Live herbaceous fuel moisture (lm)	lb water/lb fuel
Surface area-to-volume ratio (s)	sq ft fuel/cu ft fuel
Fuel bed depth (f)	feet
Dead fuel extension moisture content (df)	lb water/lb fuel
Low heat of combustion (h)	BTU/lb fuel
Total silica content (St)	lb silica/lb fuel
Effective silica content (Se)	lb silica/lb fuel
Wind speed (U)	feet/min

developed by Rothermel [18]. The input parameters needed by this model are reported in Table 1.

Once the scientists come up with either an enhanced or new fire model, they must finally collaborate with scientists from the computer science field, in order to develop fire simulation tools. That means that the theoretical models must be coded and then executed on computers. To accomplish this objective it is necessary to apply numerical methods and algorithms that solve the proposed models.

The main goal of all the implementations based on propagation models is to provide an integrated simulation system of forest-fire propagation. 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 main components of an “ideal” fire simulation system are the following [17]:

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.

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 but there are no final results that can be included in the simulation systems. Therefore, the real current simulation systems have a simplified structure, which is shown in Fig. 1 and, consequently, “ideal” fire-propagation simulators do not exist. In the following section, we will describe how to enhance the classical fire prediction scheme based on the use of “non-ideal” fire simulators, in order to provide more accurate fire spread predictions.

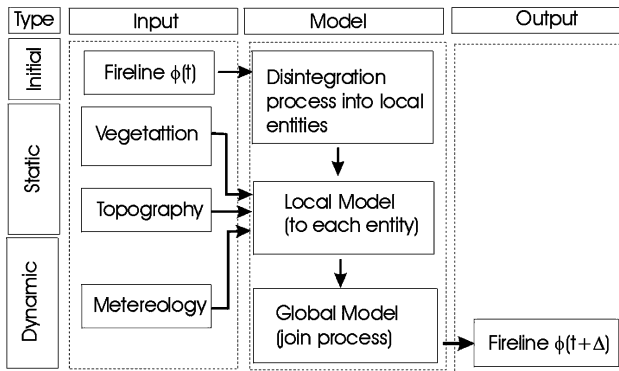


Fig. 1 Components of a simplified fire simulation system

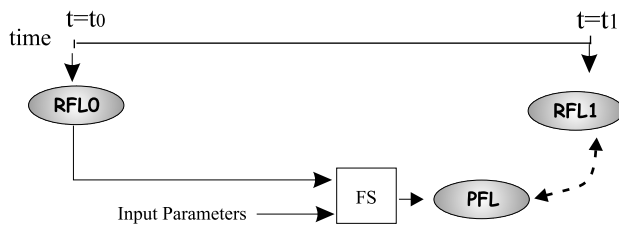


Fig. 2 Classical prediction of wildland fire propagation

3 Wildland fire prediction

As we have previously mentioned, the classical prediction scheme 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 information, ignition point, etc. ...), otherwise it will not work properly and is then executed to predict the fire line after a certain period of time.

This classical approach is depicted in Fig. 2. *FS* corresponds to the underlying fire simulator, which will be 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 the corresponding input parameters and *RFL0*) the predicted/simulated fire line at time t_1 (*PFL*) should concord with the real fire line (*RFL1*). However, due to the complexity of the fire spread modeling (as we have analyzed in the previous section), the traditional prediction scheme fairly matches the real fire propagation.

The proposed prediction method is a step forward with respect to this classical methodology. Our approach introduces

the idea of applying an optimization scheme to calibrate the set of input parameters with the aim of finding an optimal set of inputs, which improves the results provided by the fire-spread simulator. In the following, we outline how optimization has been used to enhance the classical prediction approach.

3.1 Enhanced prediction method

Formally, 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, we can formulate an optimization problem as follows:

$$\text{Find } \theta^* \text{ that solves } \min_{\theta \in S} L(\theta)$$

where L represents some objective function to be minimized (or maximized). 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 . 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 .

If we accommodate the fire prediction elements to this optimization description, the resulting scheme is the one shown in Fig. 3.

Let us analyze in more detail the correspondences between the depicted boxes and the optimization components. First of all, the vector θ corresponds to the set of input parameters to be optimized. The length of this vector will depend on the underlying simulator, but in general, as the majority of fire simulators are based on the above-mentioned Rothermel model, it will not be less than 10 components.

In order to reduce the divergence between classical prediction and real-fire propagation, we need to evaluate the goodness of the results provided by the simulator. For this purpose, a Fitness Function must be included (*FF* box). This function will somehow determine the degree of matching between the predicted fire line and the real fire-line. Both the fire simulator and the fitness function conform the objective function L together. Finally, the *OPT* box will include

Fig. 3 Enhanced wildland fire prediction method

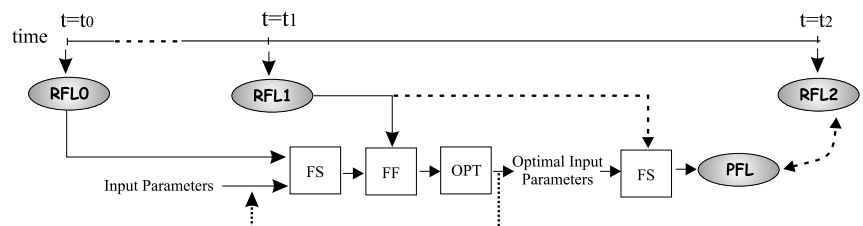
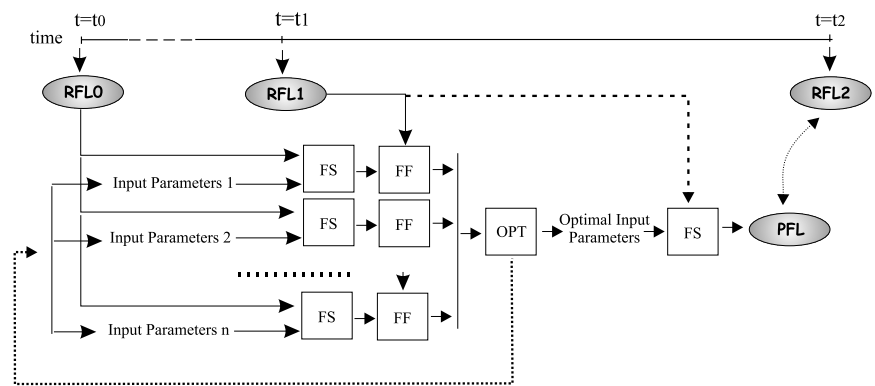


Fig. 4 Parallel implementation of the enhanced wildland fire prediction method



the particular optimization strategy selected to solve the problem. The goal of the optimization strategy consists of generating a new set of input parameters, which minimizes the underlying error prediction, taking into account the information provided by L (FS and FF boxes). The optimization process is performed in an iterative way, which is represented in Fig. 3 by the dotted arrow. This feedback loop will be repeated until either a “good” solution is found or a predetermined number of iterations has been reached. At that point, an “optimal” set of input parameters might be found, which will be used as the input set for the underlying fire simulator, in order to obtain the predicted position of the fire front (PFL) in the very near future (t_2 in Fig. 3). This process will be repeated when a new real fire-line is fed to the process in order to readjust the prediction. 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 Fig. 3, the interval time between t_1 and t_2). For this reason, it is necessary to accelerate the optimization process as much as possible.

In the following sections, we will describe two non-exclusive alternatives to deal with those real time constraints. First, we will describe how to take advantage of the computing power provided by cluster computing and parallel programming strategies to speed up the whole optimization computation time. Second, and going deeper into the particular problem we are dealing with (fire propagation), we will define alternative ways of reducing the searching space S involved in the optimization process.

4 Parallel implementation of the enhanced-prediction method

As commented above, optimization techniques typically obtain progressive improvements in the original guess of the vector to be optimized by consecutive executions of the optimization process. A great improvement in this way of proceeding is to consider not only one guess at a time, but a wide set of guesses and, based on the results obtained for all

of these, to automatically generate a new set of guesses and re-evaluate the objective function for them all, and so on.

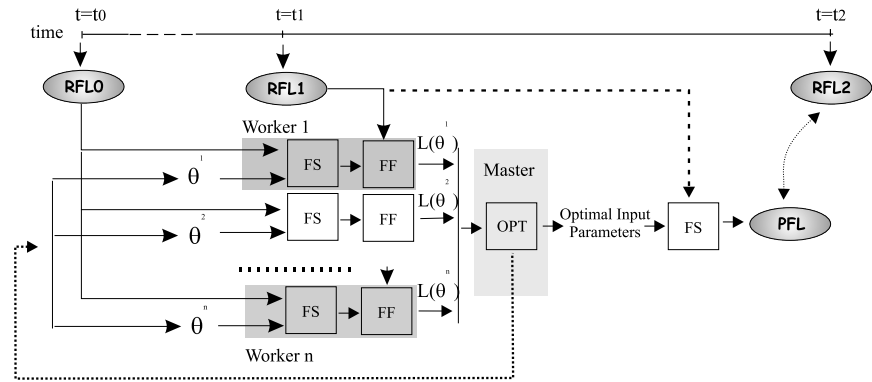
If we apply this extended optimization procedure to the enhanced-prediction scheme described in the previous section, we obtain the extended scheme depicted in Fig. 4.

For optimization purposes, we developed the optimization framework called BBOF (Black-Box Optimization Framework) [1], which consists of a set of C++ abstract-based classes that must be reimplemented in order to fit both the particular function being optimized and the specific optimization technique. Since these two elements are independent of each other, we can experiment in a “plug&play” fashion with different complex objective functions and optimization techniques. BBOF 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 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 that should be better than the previous one. In other words, this process proceeds with the following iterative steps:

1. Create a set of initial vectors randomly or distributed around some expected vector of parameters.
2. Evaluate the objective function for each vector.
3. Apply an optimization technique to move from the current set of vectors to a better set.
4. Repeat 2 and 3 until satisfying the exit conditions (number of iterations or accepted value).

This optimization scheme fits well into the master-worker programming paradigm working in an iterative scheme. An iterative master-worker application consists of two entities: a master and multiple workers. The master is responsible for decomposing the problem into small task (and distributing these tasks among a farm of worker processes), as well as for gathering the partial results in order to produce the final result of the computation. The master also runs the code of the optimization algorithm as shown in Fig. 5. The worker processes receive a message from the master with the next task, process the task, and send the results to the master. The

Fig. 5 How master-worker is matched to the BBOF elements



master process may carry out certain 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.

The three most relevant classes within BBOF are the following:

BBOF_objective_function this class corresponds to the objective function 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 of 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_guess_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 generates a report about the set characteristics and status (like average of the fitnesses and standard deviation).

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. Figure 5 graphically illustrates how this matching is undertaken.

This framework was developed to be executed on a cluster system and, for the master-worker communication purpose, BBOF uses the MPI (Message Passing Interface) library [3].

5 Optimization acceleration

The proposed enhanced prediction scheme described in Section 3 benefits from optimization to improve the fire-evolution predictions provided by the classical prediction schemes. However, as we have previously reported, in order to provide useful fire predictions, any prediction technique must accomplish certain real time constraints. Since optimization is a time-consuming process, in this section, we approach the problem of how to accelerate the whole optimization process as a non-exclusive alternative to the use of computing power provided by cluster computing to meet the real-time constraints.

Basically, the time spent by any optimization technique in finding the “optimal” solution, depends on the underlying search-space size. For our particular case (fire propagation), the search-space dimension is determined by the number of combinations of the input parameter set of the fire simulator.

As we mentioned above, the core of the most-used fire simulators is the Rothermel model [18]. The basic set of equations that make up the Rothermel model are often wrapped up by more complex formulas, which implies an enlargement of the input parameters set (as has been outlined in Section 2). Therefore, the group of input parameters of the Rothermel model makes up the minimum set of parameters we can find within the fire simulation area. In particular, this minimum number of input parameters involved in any fire simulator is at least 10 (see Table 1). In mathematics, and dealing with real numbers, there are an infinite number of possible solutions for this combinational problem. However, in computers, everything is digital and finite, so we have to consider the available computing precision. If our precision guarantees six decimal places, each variable could then take on 10,000,000 different values [10]. Thus, with only 10 parameters, the search space will be $10E70$ different combinations.

One way of reducing the search-space dimension consists of reducing the number of parameters to optimize. However, it is not feasible to eliminate any parameters in a random way because each parameter has different impact on the prediction results. For this reason, we have performed a sensitivity

analysis, in order to determine the best candidate parameters to fix to their typical value, thus concentrating, the optimization effort on the parameters that provide great impact in the simulator output. This alternative is described in the next subsection.

5.1 Sensitivity analysis

Sensitivity Analysis (SA) classically aims to ascertain how the model depends upon the information fed into it (input model/simulator parameters). The objective of any sensitivity analysis is to identify the most important factor among all inputs, which will be defined as the input that, if determined (i.e. fixed to its true although unknown value), would lead to the greatest reduction in the variance of the model output. Likewise, we can define the second most important factor, and so on, until all factors are ranked in order of importance [16].

Sensitivity methods can be classified as: (1) mathematical (2) statistical (3) graphical.

Mathematical sensitivity methods assess the sensitivity of a model output to the range of variation for an input. These methods typically involve the output evaluation for a few values of an input, representing the possible range of the input. Such methods do not address variance in the output due to the variance in the input, but can assess the impact on the output of range of variation in input values. In some cases, mathematical methods can be helpful in screening the most important inputs. These methods can also be used for verification and validation, as well as in identifying inputs that require further data acquisition or research.

Statistical methods involve running simulations in which inputs are assigned probability distributions, and process the effect of variance in inputs on the output distribution. Depending upon the method, one or more inputs are varied at a time. Statistical methods allow us to identify the effect of interactions among multiple inputs.

Graphical methods give representation of sensitivity in the form of graphs, charts, or surfaces. Generally, graphical methods are used to give a visual indication of how an output is affected by variation in inputs. These can be used as a screening method before further analysis of a model or to represent complex dependencies between inputs and outputs. These can be used to complement the results of mathematical and statistical methods for better representation.

The SA method used in this work is based on the nominal range sensitivity analysis, which is also known as local sensitivity analysis or threshold analysis [16]. This method is applicable to deterministic models and it is not usually used for probabilistic analysis. One use of nominal sensitivity analysis is as a screening analysis to identify the most important input to propagate through a model in a proba-

bilistic framework. Nominal range sensitivity can be used to prioritize data collection needs.

Basic nominal sensitivity analysis evaluates the effect on the model output exerted by individually varying only one of the model inputs across its entire range of possible values, while holding all other inputs at their nominal or base-case values. The difference in the model output due to the change in the input variable is referred to as the sensitivity or swing weight of the model to that particular input variable, in that given case.

However, there may be interdependencies among the parameters. Therefore, the effect of one parameter may depend on the values of the fixed parameters. The nominal-sensitivity analysis must therefore be repeated for each parameter for all possible cases and combinations of all the other parameters. In the particular case of fire propagation, the number of parameters is quite high and the number of combinations that must be evaluated in order to reach the sensitivity index is enormous.

5.1.1 Calculating the sensitivity index

The sensitivity of the parameters, in our case, depends on the fire-propagation model used in the core of the objective function. As we have previously commented, for a generic study, we studied the effect of the parameters of the Rothermel model. In order to evaluate the sensitivity index for each parameter, it is necessary to define a minimum and maximum value for the parameter, which are typically obtained from field and lab measurements. For all the possible combinations of the other parameters, therefore, two simulations are executed by considering the minimum and the maximum value of the parameter currently studied. The speed difference between both propagation simulations represents the effect of changing that particular parameter from its minimum to its maximum value for that particular combination of the other parameters. Let V_{ik} be the effect of varying factor i from its minimum to its maximum (difference of the speed of the minimum and the speed of the maximum) at case k .

The total effect of parameter i is defined as the addition of the effect of each possible case:

$$V_i = \sum V_{ik}$$

where k is all the possible cases (combinations of input factors). Thus, V_i will be our index of sensitivity for parameter i . This index not only reflects the effect of the parameter but also the effect of its range. Higher parameter ranges mean greater uncertainty in the measurement of that parameter. Sensitivity index is stated in Table 2.

From the sensitivity study, we can classify input parameters by their sensitivity. In this work, we have divided the parameters into 3 basic categories according to their

Table 2 Parameters ordered by sensitivity index

Parameter	Index
Fuel load (l)	0,86
Fuel depth (f)	0,77
Wind speed (U)	0,71
Humidity	0,61
Area-to-volume (s)	0,56
Effective silica content (S_e)	0,16
Heat content (h)	0,13
Total silica content (S_t)	0,03

Table 3 Parameters ordered by sensitivity index

Parameter	Sensitivity
Load parameters (l, f)	High
Wind speed (U)	High
Humidity parameters (df, lm, dm)	Medium
Area-to-volume (s)	Medium
Heat (h)	Low
Metal contents parameters (St, Se)	Low

ranking in the sensitivity: high, medium and low sensitivity. Table 3 shows the sensitivity of the input parameters of the Rothermel model according to this coarse classification. The load in the Rothermel model can be expressed by 2 parameters and they illustrate the most sensitivity. The third is wind (U), followed by humidity parameters (usually dead and living vegetation). The parameters with weakest effect are metal content (St, Se) and heat content (h). These experimental results agree with the results obtained by [19], which also use the Rothermel set of equations as a forest-fire propagation model.

6 Experimental framework

The experiments reported below were executed on a Linux cluster composed of 16 PCs with Celeron processor 433 MHz and each one has 32 MB RAM connected with Fast Ether Net 100 Mb. All the machines are configured to use NFS (Network File System) based on one server which has the same characteristics of the other machines. MPI has been

used as a message-passing interface. Subsequently, we will describe the FS, FF and OPT component of the enhanced-prediction method depicted in Fig. 3 for our particular case.

6.1 The fire simulator

For wildland fire -simulation purpose, we have used the wildland simulator proposed by Collin D. Bevins, which is based on the FireLib library. FireLib is a library that encapsulates the BEHAVE [21] fire behavior algorithm. In particular, this simulator uses a cell-automata approach to evaluate fire spread. The terrain is divided into square cells and a neighborhood relationship is used to evaluate whether a cell is to be burnt and at what time the fire will reach the burnt cells. As inputs, this simulator accepts the maps of the terrain, the vegetation characteristics, the wind and the initial ignition map. The output generated by the simulator consists of the map of the terrain where each cells is tagged with its ignition time.

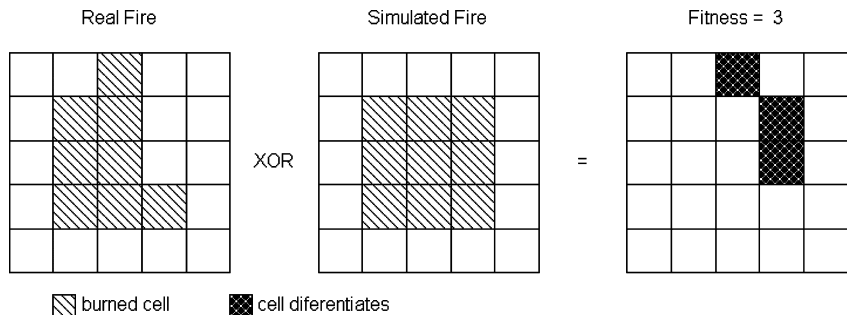
6.2 The fitness function

As we have commented, in order to measure the goodness of the predicted fire line, we need to define a fitness function. Taking into account the particular implementation of the simulator used in this experiment, where the terrain is treated as a square matrix of terrain cells, in order to ascertain whether or not the simulated fire spread exactly matches real fire propagation, we define the fitness function as the minimization of 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 known as the XOR function. Figure 6 shows an example of how this fitness function is evaluated for a 5×5 cell terrain. The obtained value will be referred to as the prediction error that, for this example, is 3. The prediction error will decrease as the simulated fire and the real fire get closer. In the ideal case where both simulated and real fire concord, the predicted error will be zero.

6.3 The optimization strategy

Genetic Algorithms (GA) are numerical optimization algorithms inspired by both natural selection and natural genet-

Fig. 6 Evaluation of the XOR function as a fitness function for a 5×5 cell terrain



ics. 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 easy to write [11, 12]. Under GA scenarios, we refers to a vector of parameters as a chromosome. A gene is identified with one parameter and the population is the set of vectors (chromosome) used for one iteration of the algorithm.

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. A sketched algorithm of a generic GA is shown in Algorithm 1.

Typically, these initial guesses are held as binary encoding (or strings) of the true variables [13], although an increasing number of GAs use “real-valued” (i.e. based-10) encoding [14], or encoding that have been chosen to mimic in some manner the natural data structure of the problem [15]. 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 this patterns as a solution representation for all elements of our chromosomes.

The initial population is then processed by the three main operators, which in our case have been implemented as follows:

Algorithm 1. Sketch of Genetic Algorithm

Randomly generating the first population of individual potential solutions.
 Evaluate the fitness function for each population member.
 While not(stop criteria)
 Obtain a new generation:
 Elitism: best individuals are copied into the new generation.
 repeat
 Selection: select two “parent” individuals with a bias toward better individuals to produce children.
 Crossover: each of two parents is divided into two parts and one part from each parent is combined into a child.
 Mutation: a single “parent” is randomly modified to generate a child.
 until a new population has been completed
 end while

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 as even. It can be implemented by copying one elite individual, and the population needs to be odd, by producing one offspring from two parents through 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. The 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 reminder parameters (genes) of each child will be obtained by evaluating the average the corresponding locus values in the parents’ genes. Formally:

$$\theta_{c1i} = \begin{cases} \theta_{p1i} & i < \text{crosspoint} \\ \frac{\theta_{p1i} + \theta_{p2i}}{2} & i \geq \text{crosspoint} \end{cases}$$

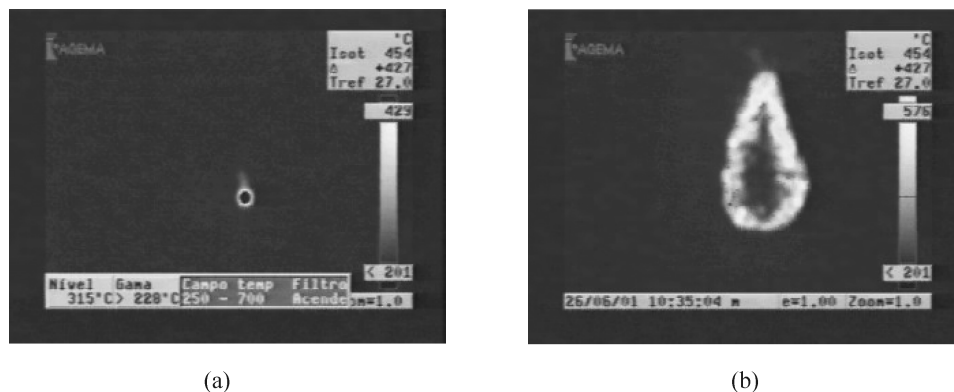
$$\theta_{c2i} = \begin{cases} \frac{\theta_{p1i} + \theta_{p2i}}{2} & i < \text{crosspoint} \\ \theta_{p2i} & i \geq \text{crosspoint} \end{cases} \quad \forall i = 1 \dots n$$

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 vector parameters.

Crossover is not necessarily implemented on each individual, it can be subject to some probability, as in the case of mutation, but the probability of crossover is usually considerable. A random number is generated between 0 and 1000, if the number is less than the probability of crossover, the algorithm applies the crossover, otherwise the individual is copied to the next generation and may undergo some mutation. Typical probabilities of crossover are 400 to 900 from 1000 or 0.4 to 0.9. When we use more probability, it reaches 0.99.

Mutation. This can be defined as changing certain aspects of the solution. For every parameter there is the same prob-

Fig. 7 Image recorded as an ignition fire (a), and fire spread after 1 minute (b)



(a)

(b)

ability of being 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 could be called the mutation distance. The other method is to change the selected parameter to another randomly generated value. The new value can be generated from the range of available values of the parameter using a random distribution function. The function can be equal distribution or any other distribution such as normal distribution. Using equal distribution, the new value does not depend on the current value; but in normal distribution the new value depends on the current value by using this as the mean of disruption.

Experiments with genetic algorithms show that the progress is not linear. Initial progress is rapid, although 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, 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 good and very good individuals.

7 Experimental results

The proposed fire-prediction model has been studied using a purposely-built experimental device. This set is composed by a burn table of $3 \times 3 m^2$ that can be inclined at any desired angle (slope) and by a group of fans that can produce a horizontal flow above the table with an arbitrary velocity. In our experimental study, the laboratory environment was set up as follows: the table inclination was set to both 30 (experiment 1) and 35 (experiment 2) grades; there was no wind and the fuel bed consisted of maritime pine (“pinus

pinaster”). Furthermore, experiment 1 was conducted by using a grid-cell of 40×40 in the fire simulator, whereas in experiment 2 the grid was set to 80×80 .

In order to gather as much information as possible about the fire-spread behavior, an infrared camera recorded the complete evolution of the fire. Subsequently, the obtained video was analyzed and several images were extracted, from which the corresponding fire contours were obtained. Figure 7(a) and (b) show the recorded images for the fire ignition and fire spread after 1 minute, respectively, for one of the two experiments performed. It should be taken into account that the slope increases from the bottom of the images toward the top. Despite the duration of this laboratory experiment (1 min 30” and 2 min from the moment of ignition), its post-mortem analysis allows us to validate the behavior of our prediction method, as will be shown in the following sections.

7.1 Classical wildland fire prediction

We used the laboratory fires described above to compare the prediction results provided by classical prediction with respect to the proposed enhanced prediction method. As previously commented, any fire-spread simulator needs to be fed with certain input parameters. Since our experimental fire was carried out under determined conditions (laboratory conditions), we have fairly good estimation values for the input parameters. Table 4 shows the values of these parameters.

We used the prediction scheme described in Fig. 2 every 30” for both experiments. That means two times for experiment 1 and four times for experiment 2. The initial time t_0 was chosen as 30”, consequently, the fire line at that time was *RFL1*. For this initial situation, we predicted the new situation of the fire front at time 1 minute by executing the fire simulator in an isolated way. The next prediction was performed by considering 1 minute as the initial time and the predicted fire line was obtained for time instant 1min 30”. This process stopped here for experiment 1, and it was repeated twice for experiment 2.

Table 4 Input parameter values measured at the laboratory fire

Parameter	Value
Wind speed (U)	0
Moisture (lm)	0.1536
Moisture of herbs (dm)	0.1536
Fuel depth (f)	0.25
Fuel load (l)	0.134
Dead fuel moist. (df)	1
Area-to-volume (s)	2500
Total silica content (St)	0.0555
Effective silica content (Se)	0.01
Heat (h)	8000

Tables 5 and 6 summarizes the obtained results. If we consider the prediction errors in terms of percentage according to the total number of cells burnt, we have 52% and 42% for the first prediction of experiment 2, and 25% and 21% for the second prediction of experiment 2 (stopping at this step). Bearing in mind the dimension of the fire ($3 \times 3m^2$), this percentage of error is considerably high.

7.2 Enhanced wildland fire prediction

The above numbers show that the classical prediction method does not provide accurate predictions, despite the study cases being well known due to its laboratory nature. Therefore, what would happen in a real fire situation where there are several types of uncertainty? We therefore applied the enhanced prediction method to the same lab fires analyzed in the previous section in order to compare the obtained predictions with that from the earlier experiments. As commented, we used the Genetic Algorithm as an optimization strategy, which will be iterated 1000 times. The input parameters to be tuned are the same as those shown in Table 4, and the optimization process will work under the assumption that there is no previous knowledge of the input parameters. The complete enhanced prediction method, depicted in Fig. 3, was applied twice, once at 30'' and again at 60'' at experiment 1 and four times at experiment 2. The first optimization pro-

cess provided an “optimal” set of input parameters, which were used to predict the new fire-line situation at 60'' and so on. The first prediction was obtained by executing the fire spread simulator once, feeding it with the real fire line at 30'' and with the “optimal” set of parameters obtained for that time. Subsequently, we continue the process of optimization to predict the fire line either at 1 min 30'' and 2 min 30''. We used the optimized parameters at 30'' as the initial generation, repeating the same process using the real-fire line at 1 minute as reference. Optimized parameters were used to predict the fire line at 1 min 30''. The result obtained in terms of improvement in prediction quality are shown in Fig. 8. This figure plots both the enhanced and classical predicted fire lines versus the real fire lines for the two performed experiments.

As we can observe from both the plotted fire lines and the obtained prediction errors (Tables 7 and 8), the proposed prediction scheme outperforms the results obtained when applying the classical scheme. In particular, the prediction errors obtained, in percentage, for the experiment 1 are 40% and 11% for both predictions, respectively. That means a reduction of 20% in the first case with respect to the classical approach and a reduction of more than 50% for the prediction at 90''. Something similar happens for experiment 2.

We can therefore conclude that the enhanced prediction method provides better results than the classical prediction scheme. In particular, we observe that the accumulation effect of the optimization method (1000 iterations at 30'' plus an additional 1000 iterations at 60'') provides better prediction quality and it is further improved if we can repeat the optimization process more times as is shown for experiment 2.

We should thus be able to both iterate the process as much as possible under real-time constraints in order to guarantee good prediction quality and accelerate the optimization process in order to converge in fewer iterations. In the following section, we will show the influence of the two alternatives above-described to accelerate wildland fire prediction: improving speed up by using cluster computing and improving optimization convergence taking into account the results obtained in the sensibility analysis.

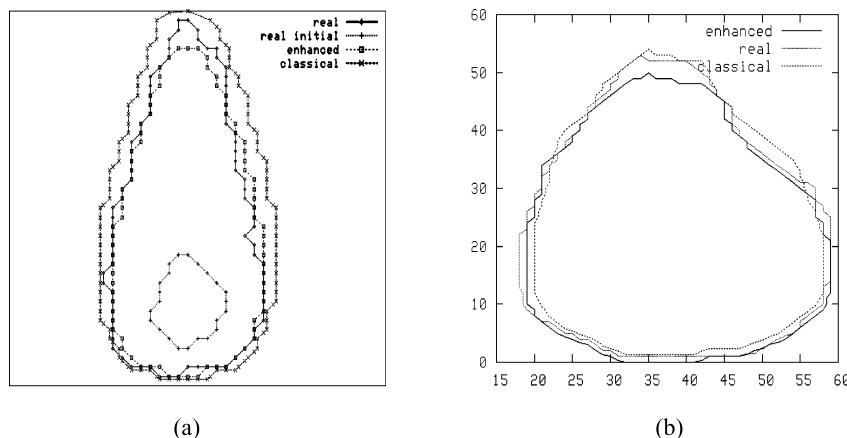
Table 5 Prediction error for the classical prediction method applied to the laboratory fire 1

Initial time	Prediction time	Total burned cells	Prediction error	Error %
30''	1 min	180	95	52%
1 min	1 min 30''	401	101	25%

Table 6 Prediction error for the classical prediction method applied to the laboratory fire 2

Initial time	Prediction time	Total burned cells	Prediction error	Error %
30''	1 min	249	106	42%
1 min	1 min 30''	539	118	21%
1 min 30''	2 min	913	126	14%
2 min	2 min 30''	1465	188	13%

Fig. 8 Predicted fire lines for the lab fire 1 (a) and 2 (b) at 90'' applying the classical and the enhanced prediction methods



7.3 Speed up improvement on cluster systems

As has been shown, in order to provide useful fire-spread prediction, it is necessary to work under real-time constrains, which must be accomplished for the proposed enhanced prediction method. For this reason, we have analyzed the speed up improvement for the prediction process as the number of processors increases.

The number of processors used were 1, 2, 4, 8 and 16. Figures 9(a) and (b) shows the evolution of the speed up for this particular example. From 4 processors we clearly observe a speed up improvement, which continues as the number of processors increases. For the maximum number of processors available, we can observe a considerable time reduction despite not accomplishing the real-time constraints required by our experiment. The prediction time for the laboratory experiment should be less than 30''. However, with the available computational resources, we have not been able to accomplish such time requirement. Figure 9 illustrates that by using more machines we can carry out more iterations in a predetermined limitation of time and obtain a better quality of prediction. Therefore, if we wish to predict fire evolution faster than real-fire spread, the use of parallel processing becomes crucial.

7.4 Improving optimization convergence

Considering the definition of the sensitivity index, if we were able to find the real value of the parameters with greatest importance, we would minimize the divergence of the simulator from reality. Therefore it is crucial to calibrate the parameters that have a greater sensitivity index while we do not know their real values. Likewise, we can say that calibrating the parameters that have little effect on the result will not significantly improve the simulator results, and this will consume processing time. The impression is therefore created that it is not worth tuning the parameters with a small sensitivity index. We suppose that fewer parameters to be optimized will make the convergence faster and, at the same time, fixing certain unimportant parameters to a given value with a reasonable error will not deviate the optimization process too far from the global minimum.

However, it is evident that if we need more accuracy, we then need to tune all the parameters regardless of their importance.

This experiment is designed to observe the effect of removing the parameters that have a small sensitivity index on the convergence of the optimization process. We used the fire-lines that were above-described. The

Table 7 Predicted error for the lab fire 1 applying the classical and the enhanced prediction methods

Initial time	Prediction time	Classical prediction error	Enhanced prediction error
30''	60''	95	72
60''	90''	101	45

Table 8 Predicted error for the lab fire 2 applying the classical and the enhanced prediction methods

Initial time	Prediction time	Classical prediction error	Enhanced prediction error
30''	60''	106	106
60''	90''	118	79
90''	120''	126	95
120''	150''	188	11

Fig. 9 Speed up for the enhanced prediction method applied to experiment 1 (a) and 2 (b) for different number of processors

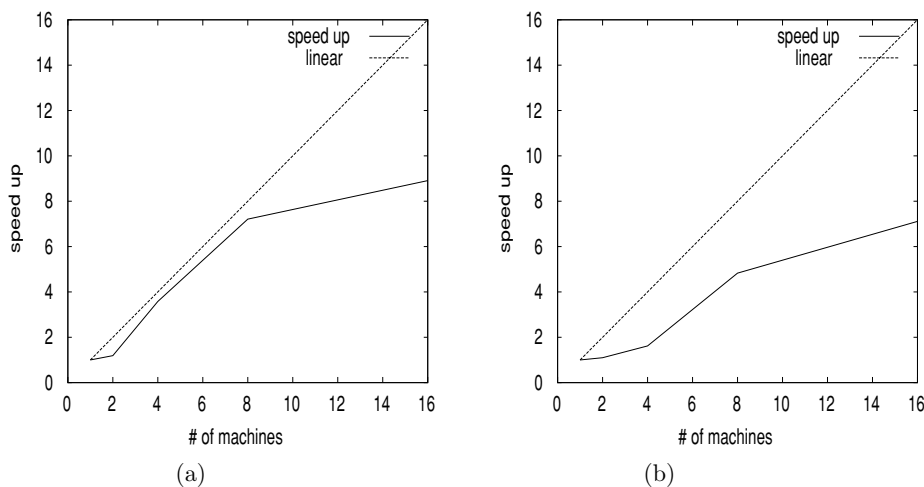
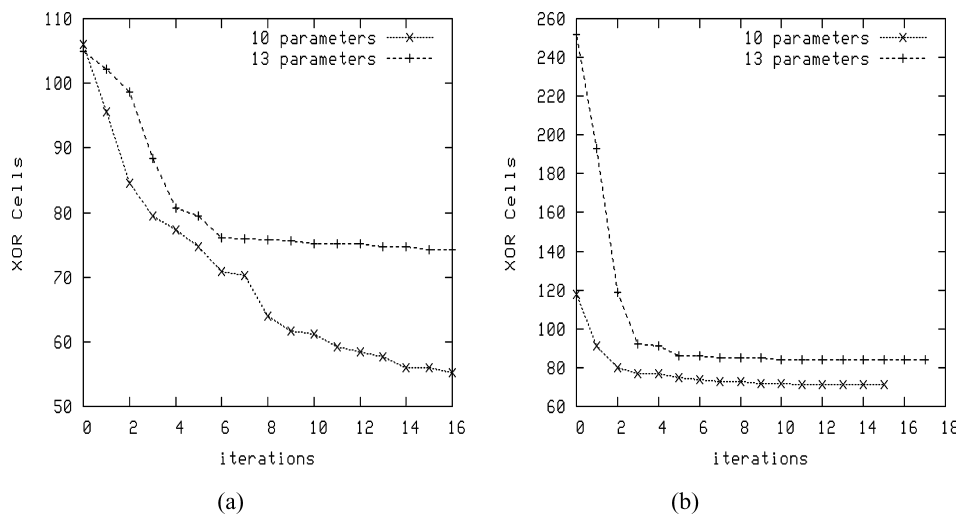


Fig. 10 Optimization
Convergence: all the parameters v.s. fixing some for the time of 1m 30'' (a) and 2 minutes (b)



optimization process was repeated 10 times for each case and the average of these repetitions is plotted on the curve.

Figures 10(a) and (b) show the convergence of the optimizing process by reducing the number of optimized parameters. The curve labeled 13 parameters represents optimizing all the parameters in the Table 3 and that labeled 10 parameters represents the convergence of the optimization process when fixing the three less sensitivity parameters obtained in Section 5.

The curve clearly shows the tendency for faster optimization convergence when fixing the parameters with low-sensitivity index. This behavior is due to the reduction of search space when optimizing less parameters. At the same time the possible error of the fixed value does not have a great influence on the result because the fixed parameters have a low sensitivity index. We can conclude that reducing the search-space is a good policy for speeding up optimization convergence.

8 Conclusion

One of the most common sources of fire-spread prediction deviation from real fire propagation is imprecision in input-simulator parameters. This problem can be approached by applying an evolutionary optimization such as the Genetic Algorithm so as to calibrate input-simulator parameters. Bearing in mind that fire spread prediction must be carried out under real time constraints in order to have useful predictions, techniques to accelerate the whole optimization process might be applied.

Since any optimization strategy is a time-demanding task, we have proposed a global-sensitivity analysis to accelerate optimization convergence by detecting which are the parameters that are better to spend effort on tuning them. This technique reduces the search space screened by fixing the less sensitive parameters to an estimated value and by focusing optimization on the most sensitive parameters. This techniques was carried out on a Linux cluster com-

posed of 16 PC's. We used a master/worker programming paradigm, where the master and worker processes communicate with each other using MPI. The results show that, combining both sensitivity analysis and cluster computing, the optimization convergence improvement obtained is quite significant.

References

1. B. Abdalhaq, A. Corts, T. Margalef, and E. Luque, Evolutionary optimization techniques on computational grids, in: LNCS 2329 *International Conference on Computational Science 2002* (2002) pp. 513–522.
2. M.A. Finney, FARSITE: Fire area simulator model development and evaluation. USDA For. Serv. Res. Pap. RMRS-RP-4 (1998).
3. W.D. Gropp and E. Lusk, User's guide for mpich, A Portable Implementation of MPI Mathematics and Computer Science Division, Argonne National Laboratory (1996).
4. A.M.G. Lopes, D.X. Viegas, and M.G. Cruz, FireStation—An integrated system for the simulation of fire spread and wind flow over complex topography spreading, in: III *International Conference on Forest Fire Research* (1998) Vol. B.40, pp. 741–754. [6] Martine-Milln and Saura (1998).
5. J. Martinez-Mill and S. Saura, CARDIN 3.2: Forest fires spread and fighting simulation system, in: III *International Conference on Forest Fire Research* (1998).
6. J. Jorba, T. Margalef, E. Luque, J. Campos da Silva Andre, and D.X. Viegas, Parallel approach to the simulation of forest fire propagation, in: *Proc. 13 Internationales Symposium, Informatik fur den Umweltschutz der Gesellschaft Fur Informatik (GI)*. Magdeburg (1999) pp. 69–81.
7. J.C.S. Andr and D.X. Viegas, A strategy to model the average fireline movement of a light-to-medium intensity surface forest fire, in: *Proc. of the 2nd International Conference on Forest fire Research* (1994) Coimbra, Portugal, pp. 221–242.
8. K. Beven and A. Binley, The future of distributed models: Models calibration and uncertainty prediction. *Hydrological Process* 6 (1992) 279–298.
9. K. Chetehouna, A. DeGiovanni, J. Margerit, and O. Sro-Guillaume, Technical Report, INFLAME Project (October 1999).
10. Z. Michalewicz and D. Fogel, *How to Solve it: Modern Heuristics* (Springer-Verlag Berlin Heidelberg, 2000).
11. K.A. De Jong, Genetic Algorithms are NOT function optimisers, in: L.D. Whitley (ed), *Foundation of genetic Algorithms 2* (Morgan Kaufmann, 1993).
12. T. Baeck, U. Hammel, and H. Schwefel, Evolutionary computation: Comments on the history and current state, *IEEE Transactions on Evolutionary Computation* 1(1) (April 1997) 3–17.
13. J. Holland, *Adaptation in Natural and Artificial systems* (University of Michigan Press, 1975).
14. J.D. Lohn and S.P. Colombano, Automated analog circuit synthesis using a linear representation, in: *Second International Conference on Evolvable Systems: From Biology to Hardware* (Springer-Verlag, September 1998) pp. 23–25.
15. J. Xiao, Z. Michalewicz, L. Zhang, and K. Trojanowski, Adaptive evolutionary planner/navigator for mobile robots, *IEEE Transactions on Evolutionary Computation* 1(1) (April 1997) 18–28.
16. A. Satelli, K. Chan, and M. Scott (Eds.), *Sensitivity Analysis*. (John Wiley & Sons publishers, Probability and Statistics series, 2000).
17. J.C.S. Andr, A theory on the propagation of surface fire fronts, PhD Dissertation (in portugues), Universidade de Coimbra, Portugal (1996).
18. R.C. Rothermel, A mathematical model for predicting fire spread in wildland fuels, USDA FS, Ogden TU, Res. Pap. INT-115 (1972).
19. R. Salvador, P. Piol, S. Tarantola, and E. And Pla, Global sensitivity analysis and scale effects of a fire propagation model used over mediterranean shrub lands. Elsevier, *Ecological Modelling* 136 (2001) 175–189.
20. M.H. Wadsworth, *Handbook of Statistical Methods for Engineers and Scientists* (McGraw.Hill, Inc, 1990).
21. P.L. Andrews, BEHAVE: fire behavior prediction and modeling systems Burn subsystem, part 1. General Technical Report INT-194. Ogden, UT, US Department of Agriculture, Forest Service, Intermountain Research Station (1986) p. 130.
22. W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical recipes in C The art of Scientific Computing*, 2nd edn (Cambridge University press, 1992).



Baker Abdalhaq received the BSc. Computer Science from Princess Sumaya University College, Royal Jordanian Society, Amman Jordan in 1993. In 2001 and 2004, he got the MSc and PhD in Computer Science from Universitat Autònoma de Barcelona (UAB), respectively. His main research interest is focused on parallel fire simulation and, in particular, how to take advantage of the computational power provided for massively distributed systems to enhance wildland fire prediction.



Ana Cortés received both her first degree and her PhD in Computer Science from the Universitat Autònoma de Barcelona (UAB), Spain, in 1990 and 2000, respectively. She is currently assistant professor of Computer Science at the UAB, where she is a member of the Computer Architecture and Operating Systems Group at the Computer Science Department. Her current research interests concern software support for parallel and distributed computing including algorithms and software tools for the load-balancing of parallel programs. She has also been working on enhancing wildland fire prediction by exploiting parallel/distributed systems.



Tomàs Margalef got a BS degree in physics in 1988 from Universitat Autònoma de Barcelona (UAB). In 1990 he obtained the MSc in Computer Science and in 1993 the PhD in Computer Science from UAB. Since 1988 he has been working in several aspects related to parallel and distributed computing. Currently, his research interests focuses on development of high performance applications, automatic performance analysis and dynamic performance tuning. Since 1997 he has been working on exploiting parallel/distributed processing to accelerate and improve the prediction of forest fire propagation. He is an ACM member.



Germán Bianchini received the BSc. Computer Science from Universidad Nacional Del Comahue, Argentina, in 2002. In 2004 and 2006, he got the MSc and PhD in Computer Science from Universitat Autònoma de Barcelona (UAB), respectively. His main research interest is focused on parallel fire simulation and, in particular, how to take advantage of the computational power provided for massively distributed systems to enhance wildland fire prediction.



Emilio Luque received the Licenciante in physics and PhD degrees from the University Complutense of Madrid (UCM) in 1968 and 1973 respectively. Between 1973 and 1976 he was an associate professor at the UCM. Since 1976 he is a professor of “Computer Architecture and Technology” at the University Autònoma de Barcelona (UAB), where he is leading the Computer Architecture and Operating System (CAOS) Group at the Computer Science Department. Professor Luque has been the Computer Science Department chairman for more than 10 years. He has been invited lecturer/researcher in Universities of USA, Argentina, Brazil, Poland, Ireland, Cuba, Italy, Germany and PR of China. He has published more than 35 papers in technical journals and more than 100 papers at international conferences and his current/major research areas are: computer architecture, interconnection networks, task scheduling in parallel systems, parallel and distributed simulation environments, environment and programming tools for automatic performance tuning in parallel systems, cluster and Grid computing, parallel computing for environmental applications (forest fire simulation, forest monitoring) and distributed video on demand (VoD) systems.