Simulator Analysis and Prometheus comparison

Jaime Carrasco, Cristobal Pais

August 29, 2017

Abstract

SimName is a new cells-based fire spread simulator. It includes a wide range of options that gives flexibility to the user when performing simulations such as statistical analysis, graphical outputs, GIS interaction, etc. In addition, a parallel implementation for high performance computers is included, allowing the users to take advantage of this approach when performing large-scale simulations. Its outputs are already integrated with optimization software like AMPL or PYOMO. It has been completely programmed in Python.

1 Introduction

In order to evaluate methods for creating forest harvest plans in the presence of forest fires, fire spread must be simulated. We will begin with some assumptions:

1. Harvest plans are issued and then executed each year.
2. Harvesting takes place before fires.
3. Replanting is soon after harvest.
4. Hourly time resolution is sufficient for modeling the spread of fire.
5. Harvest planning is done for *harvest blocks* and fire spread is modeled using *fire cells* that are much smaller.

Any of these assumptions could be relaxed later. Since the time resolution is the most pervasive and most likely to be changed, we will eschew the use of "hour" and "hourly" and instead refer to *time steps* with the understanding that we mean "fire simulation time steps" and in initial experiments, we mean simply "hours" but we won't use that word.

Assumption 1 implies that the basic time step for computing values and costs is one year.
Assumption 3 is relevant for simulations that run for a large number of years.
Assumption 4 is strong, fire can be modeled within minutes, good as a first approach.

Simulator scheme
In order to represent the problem's dynamic, figure 1 shows the different events in the forest. At the beginning of every year we have a *HPeriod* (Harvest Period) where we take harvest decisions - before fire dynamics - and every stand/block is replanted soon after these actions. After this *HPeriod*, *FPeriods* (Fire Periods) start, where all the ignitions and fire spread happen. Important is to note that for the moment we are working without a particular time step size, avoiding confusions about the length of each period.

Figure 1: Scheme of fire simulator.

Fire will start at some point - remembering that there is a specific fire season during summer - and then the fire spread model will start to simulate the fire dynamics using the weather and other information inside the simulator (probability of an ignition or fire spread). In figure 2, as an example we can see a forest with 9 stands/blocks where a fire ignites in the first *FPeriod* - after the first *HPeriod* - in cell 4 and then it spreads to different harvest units for two more *FPeriods* (note that the harvest decision was to not touch the forest, since all the forest's cells are available). After these periods, a new *HPeriod* (next year) starts.

Figure 2: Simulation example

Where:

- H_{t_h} = Harvest plan for $HPeriod\ t_h$
- R_{t_h} = Reforestation plan during $HPeriod\ t_h$
- $FI_{i,t_f} = 1$ if fire ignites in cell/block i during $FPeriod\ t_f$
- $FS_{j,i,t_f} = 1$ if fire spreads from cell/block j to cell/block i during $FPeriod\ t_f$

Note: *HPeriods* and *FPeriods* do not need to be in the same time scale, we just need them to be consistent. See appendix for details and examples for this idea, applied in a stochastic multi-stage model, including an explanation of *FI* and *FS*.

Then, we can define two main time sets: Harvest Periods (*HPeriods*) indicating the beginning of each year (independent of the time step size) and Fire Dynamic periods (*FPeriods*) where ignitions and fire spreads occur. Important is to note - again - that these sets can use different time scales if they are consistent.

We can define them in a general way:

- Set $Periods := \{t \in \mathbb{Z}^+\}$
- Set $HPeriods := \{t_h \in Periods : t_h = 1 + \alpha(1 + h)$ where $\alpha \in \mathbb{Z}_0^+, h$ is the fire period size$\}$
- Set $FPeriods := \{t_f \in \bigcup_{t_h=1}[t_h + 1, t_h + h]\}$

Example: Using 40 periods as the total planning horizon, h = 9 hours and harvesting after every fire period we have:

- Set $Periods := \{1, ..., 40\}$
- Set $HPeriods := \{1, 1 + 1 * 10 = 11, 1 + 2 * 10 = 21, 31\}$
- Set $FPeriods := [2, 10] \bigcup [12, 20] \bigcup [22, 30] \bigcup [32, 40]$

2 SimName: Stochastic Cell-based simulator

SimName is a stochastic fire ignition & spread cell-based simulator developed in Python. It allows the user to simulate the fire dynamics inside a grid instance that can represent a real forest based on variables such as weather, fuel type of each cell, elevation (terrain components), ignition points, spotting probability, etc.

Its flexibility is one of the biggest advantages to other simulations since it includes a series of user-defined options and almost all parameters and variables can be modified in order to perform tests and experiments, both in real (real data instances) or user generated instances.

In addition, performance has been optimized in order to be able to run large-scale simulations in seconds and it also includes a parallel implementation based on MPI.

2.1 Fire Simulation

All Python code can be executed from command line or from a GUI (work in progress).

The simulation code is divided in 5 main steps:

- **Step 0: Initializing instances, classes, global parameters, sets, etc.**
  Instance data is read from external sources (or can be nested in the code, but not the best option), initializing global parameters and generating all the objects needed for the simulation.

  In parallel implementation, objects are created in parallel using the resources available. Then, if we have N parallel processes, objects generation process will be divided into N job arrays.

- **Step 1: Lightning/Ignition loop to find the first week with fire**
  Based on the scheme shown in section 2.4, simulation starts when a week is selected as a fire "week". Once this week is selected, a cell, based on its characteristics, status and probabilities, ignites and the fire spread model starts.

- **Step 2: Burning cells send messages (or not) to their adjacent cells**
  Based on the weather, their status, characteristics and a probability distribution, burning cells can send messages to their adjacent cells. If no messages are sent, end of the simulation. Otherwise, go to Step 3.

- **Step 3: Messages processing**
  Based on their status, characteristics and a probability distribution function, cells that got messages can get burnt or not. Forest's cells status is updated. Go back to Step 2.

  Critical in parallel implementation where cells will be processed in parallel processes in order to take advantage of the problem structure, saving great amount of time. Cells processing will be divided in a balanced way into the available resources.

- **Step 4: Results**
  Statistics, animations and plots are generated and recorded in separated files. A connection with a GIS allows a visual representation of the fire dynamic for real instances.

  In the case of plots and animations (gif files representing the fire's dynamics), colors represent the status of the cell: green for available cells, brown for harvested cells, orange for burnt cells and different reds for burning cells. Time decay factor is represented by this colors according to the fire's intensity, thus, when a cell has just got burnt, it has the most intense red (alongside with the highest probability of spreading fire to its adjacent cells in the wind's direction) and every period its color will tend to an orange one. See **section 4.2** for more details.

Figure 3: Simulator implementation example.

Algorithm 1 Simulator Pseudo-code

```
procedure FIRE SIMULATOR(Forest_Data, Weather, Demand, Periods, TotalRuns)
    Step 0: Initializing parallel environment
        Determine number of parallel processes
    Step 1: Initializing Instances, classes, global parameters, sets, etc.
        Read forest's data
        if parallel process == True:
            Distribute data among processes
            Initialize Objects and Broadcast them
        else:
            Serial Initialization
    Step 2: Lightning/Ignition loop in order to find the first week with fire
        NoIgnition = False
        while Fire_Period < FirePeriods:
            if strike(Fire_Period) == True:
                Fire_start = Fire_Period
                Cells.Status = Cells.Ignition(Fire_start)
                Update BurningCells_Set(Fire_start)
                break, go to Step 3
            else:
                NoIgnition = True
                Next Year, Step 2
            Fire_Period+ = 1
    Step 3: Sending messages
        EmptyMsgList = True
        while BurningCells_Set(Fire_Period) is not empty:
            for c ∈ BurningCells_Set(Fire_period):
                if Cells[c].sendMsg == True:
                    MsgList = MsgList ∪ Cells[c].msg
                    EmptyMsgList = False
            UpdateCells.Status
            Fire_Period+ = 1
            if EmptyMsgList == False:
                go to Step 4
    Step 4: Receiving messages
        for c in MsgList:
            if Cells[c].gotBurnt == True:
                Cells[c].Status = "Burning"
                BurningCells_Set(Fire_Period) ∪ {Cells[c].ID}
            go to Step 3
    Step 5: Results and Outputs generation
        Generate Excel and txt files
        Output plots and animation (optional)
end procedure
```

In figure 3 we can see an example of the fire dynamics scheme. Fire ignites in cell 4 during the first *FPeriod* - first message sent to that cell - and this stand starts to send "messages" to some neighbor cells (or to all of them, depending on weather conditions). Cell 4 sends a message to cells 1, 5 and 7. These messages are received by those cells and processed by an independent core/thread in parallel by the simulator - using MPI - in order to check if they are going to be burnt or not. In the next *FPeriod*, cell 1 and 7 got burnt because of the weather and their own conditions while cell 5 not. A global update of the forest status is made.

In the second FPeriod cells 1,4 and 7 are sending messages to cells 2, 5 and 8 - as a first approach, then we can add the possibility that more than one cell sends a message to the same neighbor, augmenting the probability of getting burnt for that cell - , then parallel processing starts again.

FPeriods end when no more messages are sent by the cells. Forest's status is updated and a new HPeriod starts. Repeat.

2.2 Current features

Besides the basic simulation steps described above, the current features included in SimName are:

1. **Spotting**
   A physical spotting model (approximation) is implemented, allowing the user to simulate fires that are able to cross/jump above cells that are non-burnable such as water zones.

2. **Logical Grid output: statistical tests**
   Text files containing a grid structure can be obtained as outputs from SimName, representing the evolution and final fire scar in a simple format that allows the usage of statistical tools in order to perform comparisons between some experiments and/or other simulation software. More details in **section 2.3.3**.

3. **3D-visualization**
   A three dimensional visualization of the current instance (currently limited only to real life instances) can be obtained as an executable application by the end of the simulation process. Integrating SimName with a GIS allows the user to visualize a 3D terrain representing the instance under study. See **section 3** for details.

4. **Data files output: ready for optimizing**
   Data files representing each replication can be obtained by the end of the simulation process. These files can be read by optimization software such as PYOMO and AMPL, given the user the opportunity to develop a stochastic optimization model without post-processing issues.

5. **Graphical outputs**
   Evolution grid plots can be obtained at any point of the simulation, being able to depict the full evolution of the fire since the ignition point until burnt-out. Simple .png file are then combined in order to generate an animated .gif file for visualization purposes.

6. **Parallel Implementation (MPI)**
   A natural distributive/parallel simulation approach is included in SimName. In order

to take advantage of this implementation, tasks such as sending/receiving messages are naturally parallelized and distributed along the available resources (cores, nodes, etc.). Windows OS users may need special packages in order to be able to use this feature, for more details visit: Microsoft MPI

2.3 Modules: Classes (updating in progress...)

In the current implementation, the following classes are used by SimName:

1. **CellsFBP**
   Detailed information about each forest's cell, including the send/receive messages functions (fire spread model) and all the math involved when determining a new Fire. This is the most important class of the simulator.
   $Obj = Cells(ID, Area, Coord, Age, FType, Terrain, Vol, Perimeter, Status, Adjacents, Colors)$

   (a) **Parameters**
   Inmutables
   - ID: Id number. (int)
   - $Area$: Area of the forest [ha]. (float)
   - Vol: Volume in $[m^3]$ of cell. (float)
   - $Coord$: Coordinates of cell [lat,long,alt]. (list[float])
   - Age: Age of the forest. (int)
   - $FType$: Types of fuel. (int)
   - $Terrain$: Type of terrain. (int)
   - $Perimeter$: Perimeter of cell. (float)
   - $Adjacents$: Adjacent cells divided/classified by cardinal points. (dictionary)
   - $Colors$: Cells RGB colors based on fuel types. (list)

   Examples
   i. Vol = 1000
   ii. TerrainD = $\{0 : \text{“}Soft\text{”}, 1 : \text{“}Medium\text{”}, 2 : \text{“}Hard\text{”}\}$
   iii. $Adjacents = \{\text{“}N\text{”} :[\cdot], \text{“}NE\text{”} :[\cdot], \text{“}NW\text{”} :[\cdot], \text{“}S\text{”} :[\cdot], \text{“}SE\text{”} :[\cdot], \text{“}SW\text{”} :[\cdot], \text{“}E\text{”} :[\cdot], \text{“}W\text{”} :[$
   where $[\cdot]$ are lists that include all the adjacent cells in the eight possible directions, related to the angle between cell's gravity centers (see figure 4).

   Mutables
   - $Status$: Status of the cell. (int)
   - $FICell$: List with Fire ignition parameter. 1 if fire ignites in cell i in period t. (list[int])
   - $FSCell$: List with Fire spread parameter. Indicates the adjacent cell's ID if fire spreads from cell $j \in Adjacents_i$ to i in period t. (list[int])
   - $HPeriod$: indicates the period where the cell is harvested (if). (int)
   - $Firestarts$: indicates the period where the cell is burnt (if burnt). (int)
   - $GMsgList$: List with received messages for a cell. (list[lists])

Examples

i. Status parameter is associated to a dictionary defined by StatusD = $\{0 : \text{“}Available\text{”}, 1 : \text{“}Burning\text{”}, 2 : \text{“}Burnt\text{”}, 3 : \text{“}Harvested\text{”}\}$.

ii. $FICell$ =[0, 0, 1, 0, 0, 0, ...], then, fire ignites in period 3 for this cell.

iii. $FSCell$ =[[·],[·],[·],[2],[·],...], then, fire comes from cell 2 in period 4.

iv. $GMsgList$ =[[·],[3,6,7],[1],[·],...] indicates that this cell got messages from cells 3, 6 & 7 in period 2 and a message from cell 1 in period 3.

(b) **Functions & Methods**

- $set_Status(Status_int)$:
  Changes the status of the cell: 0 available, 1 burnt, 2 harvested. (void method)
- $get_Status()$:
  Returns cell status. (returns string)
- $send_msg(Period, Weather, AvailSet)$:
  A cell sends messages to its available (not burnt or harvested) adjacent cells based on weather. (returns list[int])
- $ignition(Period, Lambda, Weather)$:
  Determines if a cell ignites, based on the period, cell characteristics and Poisson distribution. (returns boolean)
- $got_Burnt(Period, Weather, NMsg)$:
  Based on the number of messages (with at least 1 msg), weather and cell characteristics, determines if the cell gets burnt. (returns boolean)
- $got_burnt_from(Period, MsgLists)$:
  Computes FS parameter. (void method)
- $print_info()$:
  Prints Cell information (ID, Status, Area, etc.). (void method)

Examples

- $send_msg(Period, Weather, AvailSet)$
  A cell will send messages to its adjacent cells only if some conditions are satisfied:

  i. Wind Speed $\geq$ WST (wind speed lower bound)

  ii. Temperature $\geq$ TMT (Temperature lower bound)

  iii. Dew Point $\geq$ DPT (dew point lower bound)

  iv. Precipitation $\leq$ RNT (precipitation upper bound)

  v. Radiation $\geq$ RDT (radiation lower bound)

  vi. Rate of Spread $\geq$ ROST (rate of spread lower bound)

  If these conditions are satisfied, we need to compute a sending message probability, in other words, a fire spread probability. In order to include the time effect, a decreasing probability function on t can be applied, including the $Firestart$ parameter in its definition as:

  $\mathbb{P}(\text{Send a messages in } (t > s)/\text{Cell burnt in } s) = e^{-\frac{t-s-1}{C}}$ where C is a constant that we can modify.

  After computing this probability, we obtain a random number from an

U(0,1) distribution and we check if a message is sent. If a message is sent, next step is to evaluate which adjacent cells are going to receive that message, analyzing the wind direction. Based on the angle between cells' gravity centers, each cell has an *Adjacents* dictionary where cells are classified in any of the eight cardinal points. These cardinal points are associated to different ranges (see **??**): $N \in [45^o, 135^o]$, $S \in [225^o, 315^o]$, $E \in [0^o, 45^o] \cup [315^o, 360^o]$ & $W \in [135^o, 225^o]$.

Figure 4: Adjacent cells and angles example. Cell 5 sends a message to cell 3 and then cell 3 sends message to cell 2, based on wind direction.

- $got_Burnt(Period, Weather, NMsg)$:
  Similar to the previous function, a cell that got a message (or more) from an adjacent cell will get burnt if some weather conditions are satisfied and also, depending on the number of received messages.

  If more than N_{cell} messages are received (a parameter, depending on cell characteristics, type or other elements), fuel type (FType parameter) of the cell is "burnable" and particular weather conditions are satisfied, a cell gets burnt with a very high probability (between 90 to 100%), otherwise, if a cell is not a "burnable type", it still has a chance of getting burnt, based on some probability function.

  The idea behind this scheme is to add stochasticity to the simulation. If we not include these probabilities, every time a cell gets a message, it will have the same behavior/result no matter the number of messages received or fuel type, something that in practice is not so realistic.

2. **Forest**
   Defines the main parameters of the instance.
   $Obj = Forest(ID, Location, Coord, NCells, Area, Vol, Age, Perimeter, FTypes)$

   (a) **Parameters**
   Inmutables

   - ID: Id number. (int)
   - $NCells$: Number of cells inside the forest. (int)
   - $Area$: Area of the forest [ha]. (float)
   - Vol: Volume in $[m^3]$ of cell. (float)
   - Age: Age of the forest. (int)
   - $Location$: Location of the forest. (string)

- $Coord$: Coordinates of forest's gravity center. (list[float])
- $Perimeter$: Forest perimeter [m]. (float)
- $FTypes$: Types of fuels. (dictionary)
- $CellsDistance$: Distance matrix for cells (gravity centers). (dictionary)

Mutables

- $AvailCells$: List of available (non-burnt) cells. (list[int])
- $BurntCells$: List of burnt cells. (list[int])
- $HarvestCells$: List of harvested cells. (list[int])

(b) **Functions & Methods**

- $set_AvailCells(Period, AvailCells_set)$:
  AvailCells list takes the value from the actual AvailCells set (global set). (returns list[int])
- $set_BurntCells(Period, BurntCells_set)$:
  BurntCells list takes the value from the actual BurntCells set (global set). (returns list[int])
- $print_info()$:
  Prints forest information (ID, Location, etc.). (void method)

3. **Lightning**
   Fire ignition class, determines the week when fire starts, based on a (non) homogeneous Poisson strike.

   $Obj = Lightning(Period)$.

(a) **Parameters**
Inmutables

- λ_0: Initial fire strike rate [Fires/Fire Season], where Fire Season's length is 12 weeks (3 months). (float)

Mutables

- α: Probability factor. (float)

(b) **Functions & Methods**

- $Lambda_Homogeneous(Period)$:
  Returns true if a fire starts in a particular week of the fire season using a Poisson distribution. (returns boolean)
- $Lambda_Non_Homogeneous(Period)$:
  Returns true if a fire starts in a particular week of the fire season, but using a non-homogeneous Poisson distribution. (returns boolean)

Example
Let's say that the ignition probability during the fire season is based on a poisson strike $\lambda(t)$ [fires/fire season]. Index t represents the week when fire ignites. Since we know that a fire has to ignite in order to start the simulation, $\lambda(t)$ can be an increasing function on t, having a higher probability of ignition in a week if no ignitions happened before.

Let:
$\lambda(t) = \lambda_0(1 + \alpha(t-1)) \qquad (t \geq 1)$

Then $\Lambda_{(t_a,t_b)} = \int_{t_a}^{t_b} \lambda_t d_t = \frac{\lambda_0}{2}(2(t_b - t_a) + \alpha(t_b^2 - t_a^2 - 2))$

Since we are dealing with a non-homogeneous Poisson process, we have:
$N(t) :=$ Poisson counting process, number of fires.
Probability of having k ignitions until period t: $\mathbb{P}(N(t) = k) = \frac{(e^{-\Lambda_{(0,t)}{}^t})\Lambda_{(0,t)}^k}{k!}$

Thus, probability of not having an ignition in week 1 is:
$\mathbb{P}(N(1) = 0) = \frac{(e^{-\Lambda_{(0,1)}})\Lambda_{(0,1)}^0}{0!} = e^{-\Lambda_{(0,t)}}$

Finally, probability of not having an ignition in week t, if no ignition happened in previous weeks:
$\mathbb{P}(N(t) = 0/N(t-1) = 0) = \mathbb{P}(N(t) - N(t-1) = 0) = e^{-\Lambda_{(t-1,t)}}$

Thanks to this expression, ignition probabilities are related to the previous events, instead of the classic "loss of memory" property, present in a homogeneous process.

Another classic approach is to model the $\lambda(t)$ function with a peak in the middle of the time horizon and then decrease it until the end of the season:

$$\lambda(t) = \begin{cases} \lambda_0 t, & \text{if } t < \text{Middle of fire season} \\ \lambda_0(\rho_t - t), & \text{otherwise} \end{cases}$$

Where ρ_t is a parameter that can change through time (or not), allowing us to have different probabilities distributions: symmetric, higher probability in extreme periods, etc.

4. **ReadData**
   Reads input files needed for performing the simulations. It interacts with The Canadian Forest Fire Behavior Prediction (FBP) System functions and formulas that are included inside SimName for computing the fire spread dynamics. In addition, it defines the structures applied for manipulating all the input data.

5. **ReadDataPrometheus**
   This class reads all the input files that can be used in Prometheus simulator and translates them for SimName. Thus, users that have already their files in this format can easily use them for simulating fires with SimName.

6. **SpottingFBP**
   Defines the spotting physical model and computes the probabilities of sending spotting messages within the forest.

7. **WeatherFBP**
   Weather objects where all parameters like Wind Speed, Wind Direction, Dew Point, Temperature, etc., are computed and updated from an initial state.

   $Obj = Weather(Initial_Weather)$, where $Initial_Weather$ is a dictionary.

   (a) **Parameters**
   Mutables
   - $WindSpeed$: Wind speed in [km/hr]. (float)
   - $WindDirection$: Wind Direction in grades (angles). (float)

- $Temperature$: Temperature in $[C^o]$. (float)
- $DPoint$: Dew point in $[C^o]$. (float)
- $Rain$: Precipitation in [mm]. (float)
- $Radiation$: Radiation in [J/kg]. (float)

Example
$Initial_Weather = \{$"$Wind_Speed$" : 24, "$Wind_Direction$" : 56, "$Temperature$" : 23, "Dew_Point" : 8, "$Precipitation$" : 0.0 , "$Radiation$" : 12$\}$

(b) **Functions & Methods**

- $set_Weather(WeatherSet)$:
  Sets a new weather from a dictionary. (void method)
- $update_Weather(Period)$:
  Updates the weather based on its current state and forecasts. (void method)
- $get_Weather()$:
  Returns the actual weather. (returns dictionary)
- $print_info()$:
  Prints weather information (wind, t emperature, etc.). (void method)

Depending on how we are going to deal with weather updating, the class allows the use of the following approaches:

- **Random Weather**
  Each component of the weather is affected by a % depending on a random variable generated from an uniform distribution U(a,b), where a and b are different for each weather component (e.g. $a_{WindSpeed} \neq a_{Temperature}$). Let's call them $\alpha, \beta, \gamma, \delta, \eta$ and θ; associated to wind speed, wind direction, temperature, dew point, rain and radiation, respectively.
  Next, every component of the weather is updated based on current values and the random variables (note that these random values can be negative if $a < 0$):
  - $Wind_Speed_t = Wind_Speed_{t-1}(1+\alpha)$
  - $Wind_Direction_t = Wind_Direction_{t-1}(1+\beta)$
  - $Temperature = Temperature_{t-1}(1+\gamma)$
  - $Dew_point_t = Dew_Point_{t-1}(1+\delta)$
  - $Rain_t = Rain_{t-1}(1+\eta)$
  - $Radiation_t = Radiation_{t-1}(1+\theta)$

  In a second step, a probability of raining p_r is initialized (by some method) and also, a conditional rain probability q_r is computed like $\mathbb{P}$(Rain in period t/Rain in period $t-1$). Since we are focused in the fire season, the main idea is to keep in mind that it is not so likely to have rain in consecutive periods. Thus, $p_r \geq q_r$.

- **Constant wind**
  To be updated...
- **All-directions wind**
  To be updated...
- **Custom weather stream**
  To be updated...

8. **Plot**
   Generates the plots associated with the fire evolution. It translates the forest data into a grid and assigns colors to each cell based on their fuel types. Then, if called, it plots every step during the simulation process indicating which cells are burning, already burnt, harvested, available, and which ones are sending messages to others depending on the weather conditions.

9. **Output**
   Outputs class: contains the logic for generating all the possible information from the simulator such as final forest grids in text format representing the fire scar, replications translated to scenarios.dat data files and excel files ready to perform some statistical analysis.

10. **Parallel**
    Parallel implementation uses MPI (Message Passing Interface) in Python, ready to be used in HPC (High Performance Computers) and in a distributed computing approach in any desktop computer with multiple cores. We exploit parallel programming when: creating objects (initializing the experimental instance), processing them (send and receive message loop) and when writing final results.

11. **Heuristic**
    Contains all the harvesting heuristics that can be applied during the simulation. It also adds the possibility of using a custom harvesting plan for evaluation purposes, allowing the decision maker to test the performance of the proposed plan for a particular year or the entire planning horizon.

2.3.1 Global Parameters

Some global parameters are needed in order to develop the simulation, including sets, lists and dictionaries that are updated on every step.

- $AvailCells_set$: Set that contains all the available cells of the forest. (set)
- $BurntCells_set$: Set of all the burnt cells of the forest. (set)
- $HarvestedCells_set$: Set with all the harvested cells of the forest. (set)
- $GlobalPeriod$: Current stage (week) of the simulation. (int)
- $FirePeriods[Seasons]$: Current fire period of the season. (int)
- $Seasons$: Current fire season in simulation. (int)
- $Max_Seasons$: Number of fire seasons to simulate. (int)
- $Global_Message_List$: Message list, with all the messages senders and receivers (cells) per period. (list[int])

2.4 Quick tutorial

The usage of the simulator is very straightforward for users with python background. Current implementation must be called (invoked) using the command line (Unix or Windows) and some packages are needed in order to exploit all the functionalities.

2.4.1 Getting started

The following Python packages are needed in order to run the simulator. Most of them are included in scientific python distributions like Anaconda (recommended) or they can be separately installed via pip –install PACKAGE_NAME in the command line.

1. numpy
2. mpi4py
3. openpyxl
4. ctypes
5. matplotlib
6. pandas
7. pprint

In a future implementation, the simulator will be distributed as a python package installing all the necessary third party packages (dependencies).

The simulator also uses a shared library object (.so) in order to call some functions used by the Canadian Fire Behavior Prediction (FBP) System. Thus, the following code should be executed before running the simulator:

gcc -c -fpic FBPfunc5_NoDebug.c

gcc -shared -o FBPfunc5NODEBUG.so FBPfunc5_NoDebug.o

*__Note__: Windows users should install a compiler like MinGW in order to compile the .so object using the previous code lines.

2.4.2 Basics

After downloading and decompressing the .rar file, the basic elements/files of SimName are (see 4):

1. **Modules**: Each one associated with a class of the main program. They should be in the same folder as the main code.
2. **FBP files**: FBP functions in both C and Python format. These files are needed in order to compile the shared library object.
3. **Main**: SimV1, the main code of the simulator.

Figure 4: Basic SimName files

2.4.3 Simulator Files

SimName needs a specific structure of input files in order to be able to perform the simulatios. Once a simulation is performed, different output files can be obtained depending on the arguments passed through the command line.

1. **Main Inputs**
   SimName requires three main input files in order to start the simulations:

   - **Data.dat**: This csv file contains all the relevant information for each cell, including fueltype, elevation, coordinates, wind speed and direction on that cell, among other parameters.

Figure 5: Quick tutorial: Data (.dat) input file

   - **fbp_lookup_table.csv**: This csv file contains all the fuel types information and codification. It follows the same format used in Prometheus including the RGB associated to each type in order to plot the forest grid using characteristics colors depending on the cells' fuel type.
   - **Forest.asc**: Following Prometheus format, this file contains a grid representing the real forest' cells using the codification specified in the fbp_lookup_table.csv file. It also includes information regarding the number of columns/rows of the forest and the coordinates associated to the left bottom corner of it.

```
grid_value, export_value, descriptive_name, fuel_type, r, g, b, h, s, l
1,1,Spruce-Lichen Woodland,C-1,209,255,115,57,255,185
2,2,Boreal Spruce,C-2,34,102,51,95,128,68
3,3,Mature Jack or Lodgepole Pine,C-3,131,199,149,96,96,165
4,4,Immature Jack or Lodgepole Pine,C-4,112,168,0,57,255,84
5,5,Red and White Pine,C-5,223,184,230,206,122,207
6,6,Conifer Plantation,C-6,172,102,237,192,201,170
7,7,Ponderosa Pine - Douglas-Fir,C-7,112,12,242,188,231,127
11,11,Leafless Aspen,D-1,196,189,151,35,70,174
12,12,Green Aspen (with BUI Thresholding),D-2,137,112,68,27,86,103
13,13,Aspen,D-1/D-2,196,189,151,35,70,174
21,21,Jack or Lodgepole Pine Slash,S-1,251,190,185,3,227,218
22,22,White Spruce - Balsam Slash,S-2,247,104,161,-272,229,176
23,23,Coastal Cedar - Hemlock - Douglas-Fir Slash,S-3,174,1,126,-285,252,88
31,31,Matted Grass,O-1a,255,255,190,42,255,223
32,32,Standing Grass,O-1b,230,230,0,42,255,115
40,40,Boreal Mixedwood - Leafless,M-1,255,211,127,28,255,191
50,50,Boreal Mixedwood - Green,M-2,255,170,0,28,255,128
```

Figure 6: Quick tutorial: FBP lookup table content (Prometheus csv format)

```
ncols        357
nrows        223
xllcorner    457900
yllcorner    5716800
cellsize     100
NODATA_value  -9999
101 101 101 101 101 101 101 101 101 101 101 101 101 31 101 101 101 1 1 1 2 2 2 2 101 101 101 101 101 101 101
31 31 101 101 31 31 101 101 101 101 31 31 31 31 101 101 101 1 101 2 101 101 101 101 101 101 101 101 101
31 31 31 31 101 101 31 31 31 31 31 31 31 31 31 101 101 101 101 101 101 101 101 101 101 31 31 31 101 101 1
31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 101 101 101 101 101 101 101 101 31 31 31 101 101 31 3
31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 101 101 101 101 101 101 31 31 101 101 31 31 2
31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 101 101 101 31 101 31 101 101 31 31 2 2 3
31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 101 101 101 31 31 101 31 2 2 2 3 3 3 2 2
1 1 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 101 101 101 31 31 101 101 31 31 2 3 2 3 2
31 1 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 101 101 101 31 101 101 101 31 101 1 31 2 31 3 2
31 31 1 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 101 101 101 101 31 101 1 2 2 3 2 2 2 3 2
1 31 1 31 31 1 31 31 31 31 31 31 31 31 31 31 31 31 31 31 1 31 1 2 31 31 101 101 31 2 2 2 2 2 2 3 3 2 3 3 2 2
1 31 31 1 1 31 31 31 31 31 31 31 31 31 31 31 31 31 31 101 2 2 2 31 2 2 31 101 31 1 101 2 2 2 2 2 2 3 3 2 2 2 2
```

Figure 7: Quick tutorial: Forest grid input file (Prometheus asc format)

2. **Optional Inputs**
   Users can give three optional files as inputs before starting the simulation in order to modify some specific aspects of the replications:

   - **IgnitionPoints.csv**: Useful for simulating real fires or testing purposes, this file contains information related which cells will be selected as ignition points for each year of the planning horizon. Repeating the same cell every year will lead to a one period simulation. Note that is the selected cell is not burnable, SimName will detect this situation and a message will be printed in the command line.
     If not given by the user, ignition points are randomly selected every period.

```
Year,Ncell
1,66850
2,66850
3,66850
4,66850
```

Figure 8: Quick tutorial: Ignition points (csv) input file

   - **Thresholds.csv**: Users can include this input file in order to manipulate the different thresholds and core parameters of the simulator.

     The following parameters can be modified using this file:

     (a) **TEMP**: Temperature threshold for sending a fire message between cells. Thus, larger thresholds will lead to small fires and viceversaS [Celsius].
     (b) **RH**: Real humidity threshold.
     (c) **WS**: Wind speed threshold [m/s].

(d) **PRECIP**: Precipitation threshold [mm.].

(e) **ROS**: Rate of spread (ROS) threshold. As in the case of TEMP, larger values will tend to generate smaller fires while smaller values will lead to bigger fire scars.

(f) **PROB**: Probability threshold of sending a message to "diagonal" directions when the wind direction is going to one of the main cardinal points (North, South, East or West). In other words, it is a spread tolerance (along with an internal angular tolerance).

(g) **SPOT**: Modifies the value of the spotting tolerance parameter. Larger values decrease the spotting probability function.

Users can modify all or a portion (none included) of them before running a simulation.

Figure 9: Quick tutorial: Custom thresholds csv input file.

- **Weather.csv**: Hourly weather records from relevant weather stations. If not included, the user can use some pre-defined options like random,constant or all-directions-wind weather.

Figure 10: Quick tutorial: Hourly weather csv input file (Prometheus format)

Hence, a typical instance folder will look like this:

Figure 11: Quick tutorial: Folder contents

3. **Outputs**
   Depending on the arguments given to the simulator (**see next section**), different folder/files can be obtained after runing SimName:

   - **ForestGrids**
     When using the –output-grid argument a new folder (ForestGrids) will be generated in the current working directory. This folder contains a series of csv files that contains a grid with 1s (burnt) and 0s (not burnt) representing the final state of the forest associated with each independent replication.

Figure 12: Quick tutorial: Forest grids folder after simulation

These files are useful for statistical comparison and visualization purposes without needing an intense usage of hardware resources (like with the plot option),

making each replication very fast while recording useful information.

As seen in figure 13, the user can appreciate the final fire scar corresponding to each replication, including information regarding the number of rows and columns of the forest as well as its coordinates.

In addition, these files can be read by an extra module included with the simulator files in order to both plot the forest with the corresponding color palette and/or generate the three-dimensional map (see section 3)

Figure 13: Quick tutorial: Example of forest grid file where 1s represents burnt, 0s not burnt.

- **Plots**
  When the argument –plot is given to the simulator, SimName will output a series of plots depicting the evolution of the fire since the ignition occurs. Including this option will generate a new folder called Plots containing all the plots for each independent replication.

  If it is the first time running an instance, a new file called ForestInitial.png (see 14 and 15) will be created. This file will be used as a background image when plotting the active cells, allowing the simulator to be very efficient in the RAM usage levels. After the first running, this step is skipped if the file is still located in the working directory, otherwise, it will create it again.

  Using it alone will lead to a series of plots only showing the burnt/burning cells (saving memory usage) while using it with –combine option will give the user full plots including all cells and background (based on previously created ForestInitial file).

Figure 14: Quick tutorial: Initial forest plot

Figure 15: Quick tutorial: Initial forest plot ForestInitial.png (generated by SimName)

Thus, by the end of a replication the user will have a set of pictures (see figure 16) depicting the evolution of the fire scar while indicating which cells are spreading the fire (sending messages) to others. Note that each cell color is determined by the fbp_lookup_table.csv (rgb fields) file and the coordinates represent the number of rows and columns inside the forest. The area/resolution of each cell is $100^2[m^2]$ by default, however, it can be modified by the user inside the Forest.asc file.

Figure 16: Quick tutorial: Evolution plots example when using –plot and –combine arguments

- **Scenarios**
  In order to connect the simulation outputs with optimization software (like AMPL, PYOMO and solvers), the argument –scenarios-out can be given to the command line when executing SimName. After each replication, a Scenario.dat file will be created with all the relevant information using the format defined in the **previous documentation** file. Thus, each .dat file includes all the information regarding the ignition points (FI parameter), fire spread dynamic (FS parameter) and sets definition ($HPeriods, FPeriods$).

  Thanks to this output, each replication result is ready to be used as a scenario when developing and solving a multi-stage stochastic problems in packages like PYOMO, saving a lot of time for the user.

```
Thresholds.csv   Weather.csv   ForestGrid3.csv   Scenario1.dat
1  ##########################################################################################
2  #
3  #                                   Sets
4  #
5  ##########################################################################################
6  set HPeriods:= 1 7 9 14;
7
8  set FPeriods:= 2 3 4 5 6 8 10 11 12 13 15 16;
9
10 ##########################################################################################
11 #
12 #                                   Parameters
13 #
14 ##########################################################################################
15 param FI:=
16 62950 15 1
17 62935 10 1
18 62940 2 1
19 ;
20
21 param FS:=
22 62940 62583 3 1
23 61869 61512 6 1
24 62226 61869 5 1
25 62583 62226 4 1
26 62935 62578 11 1
27 62578 62221 12 1
28 62221 61864 13 1
29 62950 62593 16 1
30 ;
31
```

Figure 17: Quick tutorial: ScenarioX.dat file format

2.4.4 CMD line arguments

In order to exploit all the features of SimName, the user should combine a series of arguments that can be given to the simulator as inputs via command line. The current arguments are the following:

1. **-h, –help**: shows a help message (summary of all possible arguments) and exit.
2. **–input-data-file** INPUT_FILE_NAME: Path and name of the csv file that contains the data (not needed if input instance folder is given). (String)
3. **–input-forest** INPUT_FOREST: Path and name of the csv file that contains the forest grid data (not needed if input instance folder is given). (String)
4. **–input-fpblookup** INPUT_FBPLOOKUP: Path and name of the csv file that contains the fuel (not needed if input instance folder is given). (String)
5. **–input-instance-folder** INPUT_FOLDER: Path to the folder contains all the files for the simulation (by default the best option for using the simulator). (String)
6. **–weather** INPUT_WEATHER: Path and name of the csv file that contains the weather (hourly) data. (String)
7. **–spotting**: Activates spotting occurrence. (Boolean)
8. **–plot**: Plots are created after the simulation. (Boolean)
9. **–verbose**: Output all the simulation log. (Boolean)
10. **–ignition-point** INPUT_IGNITIONS: Path and name of the csv file that contains the ignition points data. (String)
11. **–ignitions**: Activates the predefined ignition points when using –input-instance-folder argument. (Boolean)
12. **–nsims** INPUT_NSIMS: Total number of simulations (replications). (Int)
13. **–heuristic** INPUT_HEUR: Heuristic used. (Int)
14. **–statistics**: Excel file with statistics is created at the end of the simulation (inside working folder). (Boolean)

15. –**combine**: Combine fire evolution diagrams with the forest background (usage with plot argument). (Boolean)

16. –**save-memory**: Activates memory monitoring version (useful for very large instances). Reduces the amount of RAM used during the simulations. (Boolean)

17. –**scenarios-out**: Activates scenario output files. One file per scenario is created inside a new folder in the current working directory. Ready for AMPL/PYOMO .dat file. (Boolean)

18. –**weather-file**: Activates weather stream file when using –input-instance-folder argument. (Boolean)

19. –**no-output**: Activates no-output mode, useful for a series of experiments (batch files/jobs). (Boolean)

20. –**output-grid**: Activates forest grid's output mode. This argument creates a csv file with 1s and 0s representing the final fire scar for each replication. (Boolean)

21. –**custom-thresholds**: Reads custom thresholds file (if any). (Boolean)

Figure 18: Quick tutorial: Command line usage example

2.4.5 Running SimName: Examples

Depending on the version of Python installed, users must invoke the corresponding version of SimName. Currently, SimName can be used in both Python 2.7 and 3.x versions.

In order to run one experiment/instance the user should invoke python and then call the main program SimV1.py. After that, users can add as many arguments they want, taking into account the type of input needed.

Notice that the user can set the argument –no-output (boolean) in order to not get any output from the active console (just the number of the current replication). On the other hand, –verbose option will output all the information log of each replication.

Some examples:

- Run 100 independent replications in the "save memory" mode with custom thresholds, using a folder with all the necessary input files (Thresholds.csv, Weather.csv, Data.csv, Forest.asc) without showing anything on the console and generating Forest grids (one per replication).

Figure 19: Quick tutorial: Command line example (1)

- Run 100 independent replications with spotting, using a folder with all the necessary input files (Thresholds.csv, Weather.csv, Data.csv, Forest.asc) showing current replication number on the console and generating Forest grids (one per replication) and scenarios.

Figure 20: Quick tutorial: Command line example (2)

- Run 50 independent replications with spotting, using a folder with all the necessary input files (Thresholds.csv, Weather.csv, Data.csv, Forest.asc) showing current replication number on the console and generating Forest grids (one per replication), plots (combining the fire scar with the forest's background) and scenarios.

Figure 21: Quick tutorial: Command line example (3)

The power of using the command line invocation relies on the possibility of sending a set/batch of experiments with only one file. Therefore, batch file (Windows OS) and/or bash files (Unix) can be generated with the arguments needed by the user such that multiple experiments are performed.

2.4.6 Output Examples

1. **Dogrib instance with constant east (0) wind.**
   In this example, we run the Dogrib forest instance (as in Prometheus) with a constant east wind instead of the original north-east direction for simulating another possible fire spread dynamic on the same forest.

   As we can see in figure 24 the ignition point is the same as the original fire but since the wind direction is different (and constant) in this case we end up with a different fire scar, that actually reaches the eastern border of the instance under study.

   Therefore, it is possible to understand the power of SimName when performing different experiments and tests when some parameters are modified (more details in **section 4**).

Figure 22: Fire evolution for Dogrib instance when using constant east wind ($\theta = 0$) in SimName.

2. **15 Cells with barrier, random wind**
   This instance can be used for testing the spotting function of SimName since the fire will not spread across the vertical "barrier" in the second column of cells. In figure 23 we can see how the fire spreads only on the right side of the barrier since no spotting argument was given.

Figure 23: Fire evolution for a 15 cells instance with vertical barrier using random wind in SimName.

3. **400 Cells Homogeneous, west and constant wind** In this case, we have a homogeneous (one fuel type) instance consisting of 400 cells with west and constant wind. This means that west (plus north and south) spread directions are allowed at each period, for testing purposes.

Figure 24: Fire evolution for a 400 cells (homogeneous) instance using random wind in SimName.

Other examples can be seen in our website or repository.

3 3D output: GIS based output (work in progress)

In order to obtain a more realistic visualization of the results of the simulation, an integration of the simulator with a Geographical Information System (GIS) has been done.

Using satellite pictures of the relevant areas under study, an algorithm obtains the texture and heightmaps of the zones in order to generate three-dimensional meshes based on the elevation of the different components of the terrain (everything is on a real scale). Then it adds the terrain textures (based on satellite images, like Google Earth), 3D-trees associated with the fuel type and it is possible to actually see the fire evolution in a 3D animation (not in these pictures).

Figure 25: Heightmap to 3D-mesh process

Figure 26: Textured 3D-terrain based on a heightmap

3.1 Output

Two main outputs are generated when using the GIS based script:

1. **3D-Visualization APP**
   After simulating a fire, a visual application is created in an external server (internet connection needed). This consists of a 3D application that allows the user to view and interact with the three-dimensional forest generated from the satellite pictures and height maps.

   Thanks to this application, the user would be able to "visit" the original forest and compare it with the final state reached after the fire period.

   In addition, the user can move and visit each part of the forest "flying" and/or "walking" over the 3D-terrain. The application also takes into account the current time - internet connection needed - and simulates the "real life" climate and light (day, night, raining, etc.)

2. **Video**
   Based on the evolution of the fire scar and cells' information of each replication, a 3D-animated video is generated, showing the evolution of the fire from the initial state of the forest until the end of the fire period.

3.2 Examples

The basic plot obtained by the simulator when dealing with a complete instance is based on the information contained inside the Forest.asc file where the forest grid information is codified depending on the cells' fuel types. Same input (see 27) is generated when working with Prometheus.

Figure 27: Dogrib 2D forest example, as seen in Prometheus

These maps -as we already indicated in previous sections- will be used as the general backgrounds for generating the simulator graphics. In addition, thanks to this implementation we do not ask the simulator to recompute the plot for those cells who were not actually active during the replication (burning state) saving a relevant amount of memory usage.

However, these “pixelated” cells’ based maps are not completely accurate to the real forest distribution since a series of approximations must be done in order to fit it into the grid disposition while satisfying cells’ resolution limitations, and they also lack of details that can be relevant to further analyze some harvesting and/or firewall policies.

Thus, an integrated workflow with a GIS is performed by the end of the simulation. Based on a web-server’s script, a 3D-textured mesh is generated (see 28) from the associated heightmap and satellital pictures obtained from different services like gooogle maps/earth.

Figure 28: Dogrib 3D model example: Aerial view

A 3D-forest visualization app is generated using the data from the GIS. Then, the user can visit and visualize any point of the virtual terrain (real scale). Examples of this output can be seen in figures 28 to 34.

Figure 29: Dogrib 3D model example: Mountains view

As can be seen in the pictures, the visualization app takes into account the position, texture and volume of each component of the terrain in order to generate the 3D-mesh, texturizing it with data collected from satellital services. In addition, light/shadows and weather elements are calculated using online information, giving a “real and live” view of the forest state before and after the simulation.

Figure 30: Dogrib 3D model example: Landscape view

Figure 31: Dogrib 3D model example: General view of the instance.

Figure 32: Dogrib 3D model example: Sunset effect.

Figure 33: Dogrib 3D model example: Transition to night.

Figure 34: Dogrib 3D model example: Night ambientation.

Hence, a completely interactive visualization app is generated by the simulation when using the GIS-based output module.

4 Prometheus: Comparison with deterministic simulator

Blablabla....

4.1 Methodology

Blablabla....

4.2 Instances

Blablabla....

4.3 Results

Blablabla....

5 Future/Current Work

1. Gather and test more real life instances with different weather conditions and compare them with existing simulator outputs in order to validate SimName's performance. A series of experiments must be done in order to determine the best value of relevant parameters (like thresholds).

2. Fire evolution comparison with Prometheus. The current comparisons take into account the last state of the forest (final fire scar), while progressive scars (fire evolution) are not being compared. Thus, we would be able to determine if both models present similar fire spread evolutions.

3. Spotting model: implement a formal spotting model.

4. Optimization of SimName functions. A general optimization of the most relevant functions and methods has been performed. Secondary functions will be optimized, taking advantage of parallel/concurrent computing approaches.

5. An optimized version for HPC (high performance computers) will be finished. It takes advantage of the cluster structure and implements a series of rules for queuing the different jobs among clusters/nodes.

6. Full integration of the simulator with the GIS module for the 3D-mesh generation. Current beta implementation needs a series of software packages to be installed on the local machine in order to generate the visualization app and three dimensional forest video. Next version will run on an external server, just needing an internet connection.

7. Formal article (paper): start writting it based on the documents and results we have.

6 Bibliography

BibTeX.