Airtanker Simulation Model

This report’s goal is to thoroughly explain the Airtanker simulation model I have written. This report will go through how the whole model works in detail. This report will directly go over the code and explain what every function is doing, how they were implemented, the thought process behind these decisions, some of the assumptions made, and in some cases, ways that the functions could be improved in the future to add further complexity or usability to the model. It will also help users to understand how to use the model correctly.

Understanding the Code

This simulation model was mostly written in python with the use of the simpy library. However, some of the formulas for calculating things related to coordinates on the earth, and things related to ellipses were instead written in C and then wrapped using SWIG so they could be used in the python program. This was because writing these numerical computation heavy functions in C greatly improved the performance of the program. No guarantees are made on the accuracy of the code’s implementation, assumptions made or results, so it should only be used at the users own risk.

Imports

The beginning of the code includes a bunch of import statements. The future division one is to prevent division of two integers resulting in an integer rather than a floating point number. The math library is needed for pi as well as trig functions and the square root function. The random library is used to make this simulation produce “different results every time”. It is used to introduce variability in fire ignition times, service times and other things just like what would be seen in the real world. The simpy library is used to handle all of the event processing, and resource management parts of the simulation. The time library is only used to see how long the simulation took to run. The fbpline2 library is a python implementation of the Canadian FBP system for line ignition fires. This library is used to generate the rate of spreads used for the fires based on numerous different fires. Finally the haversine library is the python wrappers of the C code with many different numerical functions defined inside.

The variable EPS is then defined to equal 0.00001, this number was arbitrarily chosen. Since floating point arithmetic is inexact, it is good practice to check if the difference between two numbers is less than some epsilon rather than checking if they are precisely equal, therefore this constant serves that purpose.

FOREST\_DATA\_LOCATION

* This constant defines the name of the input csv file that will contain all of the forest data. It is canadian\_forest-data.txt

AIRTANKER\_DATA\_LOCATION

* This constant defines the location of the csv file named airtanker\_data.txt that lists the models of different airtankers, as well as the stats to use for each of them in the simulation.

READ\_ONLY\_CONSTANTS

* This constant defines the location of the csv file called read\_only\_constants.txt redefining the values for the read-only constants used by this simulation model.

Read-only Variables

The next section of the code defines various read-only variables used by the program. Although these is general should not be changed, the program gives users the option to get the values from an input file instead of using the default values if they wish. If the user wishes to change the values, they should create a csv file name “read\_only\_constants.txt” that defines every constant separate by commas. I will now go over each of the constants:

EARTHS\_RADIUS

* This value changes the value for the earth’s radius used by the program, however the haversine functions that do the bulk of the calculations using the earth’s radius’ value do not depend on this value, so changing this value will not change the earth’s radius value of 6371.0km they use.

USE\_FOREST\_DATA

* This constant determines whether the forest data should all be randomly generated, or will instead come from a file. This model assumes that the information will come from a file and should generally be run this way, but if you have no file, then this can also be changed to generate a random forest.

IMPROVE\_POINTS\_BURN

* When True, this causes the check to see if points are burned by fires to improve in accuracy, however this adds extra time to the program runtime due to additional calculations. This should generally always be True. Since the fires grow elliptically, the ignition point is not the true centre of the ellipse. When this is set to true, the fire first finds this true centre, then calculates if the point is within the fires area, whereas if this is not true, the program just assumes that the fire’s ignition point is the ellipse’s centre. For small wind speeds, this should give approximately the same result.

EAST\_DIRECTION, CCW\_IS\_POS

* These two variables define the coordinate system that the user is using. This allows the program to convert from different user entered systems to the one the system uses. The default is to use the polar coordinate system. This is done by defining what the east directions value is in radians (it is 0 for the polar coordinate system) in your system, as well as whether moving in the counter clockwise direction causes the radian value to increase (as it does for polar coordinates)

BHE

* When a fire is controlled its size is reduced by this amount (a 0.4 value of BHE means the fire will be brought down to 60% of its original size when controlled). Although this constant doesn’t yet affect anything else, in the future it could be changed to affect the amount of time fighting fires actually takes.

DROP\_TIME

* No longer used by this model.

PROB\_DETECTED\_LIGHTNING, PROB\_DETECTED\_HUMAN

* This variable determines the probably that a fire is detected. When determining if a fire is detected, a uniformly distributed random number on the range [0, 1) is generated. If this random number is less than PROB\_DETECTED\_ then the fire is detected, else it is not detected. There can be a different probability for both lightning caused fires and human caused fires.

LENGTH\_DAY

* This value is used for determining the length of a single day. It is set to 24.0 by default (the number of hours in a day). I don’t see any reason this should be changed, but it can be.

DETECT\_TIME\_LIGHTNING , DETECT\_TIME\_HUMAN

* These values are the mean values inputted in an exponential distribution for determining the amount of time after a detected fire ignites until it is detected. These values can be different for both lightning and human fires, but both default to 120.0 minutes.

REPORT\_TIME\_LIGHTNING, REPORT\_TIME\_HUMAN

* These values are the mean values inputted into an exponential distribution for determining the amount of time after a detected fire is detected until it is reported. At this report time, an Airtanker is immediately requested, and immediately sent off if available. These values can be different for both lightning and human caused fires, and they default to a time of 10.0 minutes.

AMP\_H, LAG\_H, AMP\_L, LAG\_L

* These constants are used for determining the fire arrival rate based on the equation:

Lambda(t) = Lambda\_mean – AMP \* cos(2 \* pi \* (t – LAG) / delta)

Lambda(t) is the fires per hour that arrive at time t in the day, lambda\_mean is the mean fire arrival rate per minute, AMP is the amplitude of this equation, LAG is the time delay in minutes, and delta is the length of the period of time in the day when fires can be ignited in minutes

LAKE\_DIST, AT\_AVAIL\_CONST

* When determining if an Airtanker has enough time to service a fire before it is off duty it estimates the length of time to fight the fire based on the time to get to the fire and perform 2 drops assuming the fire is LAKE\_DIST (in km) away. The read-only value AT\_AVAIL\_CONST is added to this extra time. If this extra time would put the Airtanker after the end of its service time it is considered unavailable. (Note that this is a low estimate of the time it would take to serve the fire that is used, and it does not contain any return to base time).

Classes

I will now go over the different classes used in this simulation to keep track of all of the different values for everything. Aside from the global read-only variables mentioned previously, instances of these classes store the values of everything important that happens in the simulation. To start off I will go through the User\_input class that stores all of the main input parameters given by the user of the model. These are the parameters that will often change between runs. This class takes a list of all of the parameters it will use.

number\_runs

* This input determines how many times a run of the simulation is called. When determining statistics at the end, the average over all of the runs as well as the maximum and minimum value from a run will be displayed.

Length\_run

* Input defining the number of minutes a run lasts for. When creating this simulation model it was assumed this would generally equal the length of a day (1440.0 minutes), but it can be changed if needed. Most of this documentation will discuss the outcome of functions assuming this is only equal to one day since that was the original design of the model, so if this value is different, care should be taken to ensure the model will run as you would expect.

Time\_until\_start

* Input defining the time in minutes into the day when fires are able to start igniting (the sunrise)

Time\_until\_dark

* Input the time in minutes into the day after which new fires stop igniting. This value should be great than the time\_until\_start

Lightning\_fires\_day

* Input defining the estimated number of lightning caused fires that occur each day (NOT EACH RUN) of the simulation

Human\_fires\_day

* Input defining the estimated number of human caused fires that occur each day (NOT EACH RUN) of the simulation

Show\_fire\_attributes

* When this input is set to True, the simulation will print every known detail about every fire and Airtanker after each run of the simulation. This Is useful for understanding exactly what happened during each run of the simulation.

Save\_daily\_averages

* If this input is True, this causes the model to save every runs average for all of the tracked statistics, as opposed to just the current mean, max and min. Currently this should always be set to False as the extra data isn’t used for anything, but later on the ability to calculate other statistics like standard deviation, or to graph the statistics over runs may be added which would require the daily averages to be saved.

Month, day, min\_fmc\_month, min\_fmc\_day, FFMC, BUI, wind\_speed, wind\_direction (direction wind is coming from), percent conifer, percent dead fir, grass fuel load, percent cured

* These inputs are all of the necessary inputs for use by the FBP system that is used to determine the head, flank, and rear rate of spread of every fire, as well as the head direction. In the future this should be changed to read the fbp values specific to location (such as ffmc, wind, etc.) from a file that has the values to use for each cell of the forest, however this has not yet been implemented.

Min\_lat, max\_lat, min\_long, max\_long

* These values represent the coordinates of the rectangle that contains the forest used in this model. Currently this is only used to define the range of the uniform distribution from which point latitudes and longitudes are randomly generated. This is also used if no forest file is given as it will define the bounds of the random forest

Num\_rows, num\_columns

* Integers only used when no forest data file is given that define the number of cells in each row and column of the forest. These cells each have their own fueltype.

Base\_info\_file

* This input defines the location of the csv file that defines all of the airtanker base information, which is the bases latitude, longitude, number of airtankers, number of bird dogs.

Next, this class opens the base\_info file and reads through it to save the data from it.

Bases\_lat

* List storing the latitudes of every base. This is also the initial latitude used for all airtankers that start at this base.

Bases\_long

* List storing the longitudes of every base. This is also the initial longitude used for all the airtankers that start at this base.

Base\_num\_airtankers

* This variable is a list that keeps track of the number of airtankers that are initially at each of the bases.

Base\_num\_bird\_dogs

* This variable is a list that keeps track of the number of airtankers that are initially at each of the bases. Note that bird dogs are not yet implemented in this model so this should be set to 0.

Fire\_location\_file

* This input saves the location of the csv file that contains the probability for a fire to grow in each cell of the forest, as well as the fuel type from that cell to use in the FBP calculations for that fire.

Specific\_airtanker\_file

* This input saves the location of the csv file containing the data specific to each individual airtanker at each base. These are different than the model parameters, as these are values that may be different from airtankers of the same model

Num\_points

* This variable lists the number of points to use in the simulation. These points are random spots spread throughout the forest that are kept track of over the runs. After every run, the model checks to see if each point was burned or not.

Points\_lat, points\_long

* These optional inputs are lists storing the latitudes and longitudes of the points used in the simulation. If this input is not given, or is not given for every point, then values are randomly generated based on a uniform distribution using the min and maximum latitude and longitude inputs.

The Class is the Fire class. This is the class used to store everything specific to each individual fire grown in the simulation (Each individual fire is an instance of this class).

Time\_at\_ignition

* The time in minutes into the run that the fire was ignited at. At this point in time the fire is assumed to start as a point and start growing immediately based on the rate of spread values generated from the fbp.

Time\_at\_detection

* If the fire was detected this value will represent the time in minutes into the run that the fire was detected at, otherwise it will equal -1.0

Time\_at \_report

* If the fire was detected, this value will represent the time in minutes into the run that the fire was reported at. This is the time at an airtanker is requested to fight the fire at. If the fire was not detected, then this value will be set to -1.0

Slope

* The slope used in the fbp calculations for the fire. This value is always currently considered undefined, and therefore doesn’t affect anything.

Time

* The fire’s current time. This value can be updated, followed by calling the fire’s growth() method to find the fires size and parameters at different points of time in the simulation run.

Latitude/longitude

* These values represent the coordinates of the point of ignition for the fire.

Real\_lat, real\_long

* These values represent the coordinates of the centre of the elliptical fire. Unless there is no wind in the simulation run, this value should be different than the point of ignition coordinates.

Detected

* If this attribute is True then the fire was detected, else it was not detected. Only fires that were detected will be fought by airtankers, non-detected fires will silently grow in the simulation model for the length of the run.

Size\_at\_detection, perimeter\_at\_detection, head\_length\_detect, flank\_length\_detect, back\_length\_detect

* These attributes save all of the parameters defining the fires current size at the time of its detection. If the fire was not detected then all of these will have a value of -1.0. These values are in kilometres or kilometres squared.

Size\_at\_report, perimeter\_at\_report, head\_length\_report, flank\_length\_report, back\_length\_report

* These attributes save all of the parameters defining the fires current size at the time it was reported. If the fire was not detected then all of these will have a value of -1.0. These values are in kilometres or kilometres squared.

Size, perimeter, head\_length, flank\_length, back\_length

* These attributes save all of the attributes defining the fire at the current simulation time. These values are in kilometres or kilometres squared.

Max\_size, max\_perimeter, max\_head\_length, max\_flank\_length, max\_back\_length

* These attributes save all of the parameters defining the fires size at the point when it was largest during the simulation run. For detected fires, this point is considered to be right when an airtanker arrives to fight the fire (or the end of the simulation run if no airtankers are able to arrive fast enough), and for non-detected fires, this is always considered to be at the end of the simulation run. These are also used when calculating if the points are burned in the simulation. Every point is checked to see if it was within any fires maximum area. These values are in kilometres or kilometres squared.

Head\_ros, flank\_ros, back\_ros, head\_direction

* In kilometres per minute, these values generated from the FBP represent the constant rate of spread of the head flank and back of the fire, as well as the direction of the head. Once determined, these values are constant for that particular fire for the whole length of the simulation run.

Elevation/slope

* Both of these are optional inputs to the FBP system and therefore are included, however currently they are always undefined and therefore don’t affect anything in this model. In the future it could be possible to implement map data to make these values meaningful.

Cause

* Whether the fire is a human fire or a lightning caused fire. Currently in the simulation this only affects the frequency and location of where fires ignite, and not how they are fought.

Value\_at\_risk

* Does not currently affect anything, but a possible future addition to help determine the damage done by the fires, and potentially the total cost incurred by the whole run for different set ups of bases and airtankers to try and optimize the process.

Airtankers\_required

* Not yet implemented, but would represent the number of airtankers that need to be sent to this fire

Airtankers\_still\_required

* Not yet implemented, but would represent the number of airtankers that still need to be sent to the fire

Bird\_dogs\_required

* Bird dogs not yet implemented in any way, but this would determine the number of birddogs that need to be sent to a fire.

Priority

* Not yet implemented, but could be used for a number of things including allowing a fire to skip a queue for an airtanker, or even cause an airtanker to stop fighting a different fire to fight it instead based on some rules.

size\_at\_airtanker\_arrival

* List storing the size of the fire at each airtankers arrival (currently only one airtanker maximum will only go to a fire however so this will always be 0 or 1 value long)

Perimeter\_at\_airtanker\_arrival

* List storing the perimeter of the fire at each airtankers arrival time. (currently only one airtanker maximum will go to a fire however, so this will always be only 0 or 1 values)

Time\_at\_airtanker\_arrival

* Listing storing the times every airtanker arrives at the fire. Currently only one airtanker maximum will go to a fire, so this will only contain 0 or 1 times.

Time\_at\_controlled

* The time in minutes into the simulation that the airtanker is considered controlled at. After this point the fire no longer grows, but also does not get served by airtankers.

Controlled

* If True the fire has already been controlled and is no longer changing in size and doesn’t need any additional airtankers. If false, then either the fire is not yet controlled, or it was not detected and never will be.

Get\_ros(self, inputs)

* This method uses the fbp input variables to feed into the fbp system and from that is able to get the values for this fire’s head, flank and back rate of spreads, and the head direction.

Calc\_value\_rick(self)

* Not yet implemented, but in some way would calculate the value at risk by the fire

Growth(self)

* Grows the fire to its size at the time of its time attribute. It does this by first using the rate of spread (km/min) \* time since ignition. Then it finds the major axis of the ellipse which is half the sum of the head and back. Then the angle is the angle the flank\_distance makes with the line in the head direction passing through the centre of the ellipse. This is used to calculate what the length of the minor axis of the ellipse is. Once the major and minor axis can be found, it is simple to estimate the size and perimeter of the ellipse. If the time value is before the fire has ignited yet, all of the size parameters are set to -1.0

Max\_growth(self, inputs)

* The fire is assumed to grow until the end of the simulation run. This is the time that undetected fires will grow until, as well as fires that never got an airtanker to fight the fire. Therefore on fire creation these are set as the fires max sizes. If the fire ends up being controlled later on in the simulation, these max sizes will be changed to the size of the fire when the airtanker arrived.

Detect(self)

* Grows the fire to the detection time size, then saves all of the parameters defining its size at detection, then repeats this for its size when reported. Just sets the time attribute, calls the growth() method and then saves the new sizes.

Real\_centres\_max(self)

* Calculates the coordinates of the centre of the ellipse and saves them to the real\_lat and real\_long variables. It does this by calculating the new coordinate as if you moved the point of ignition, half the distance that is the difference between the head and back lengths in the direction of the head. It also adjusts for coordinate systems since this formula assums north is 0 radians, and clockwise is positive.

Print\_attributes(self)

* This method is called to print all the details of the fire (it prints every attribute it stores)

Statistics

The statistics class is the class that stores the important statistics from each run of the simulation In its list attributes. There are two methods of saving the statistics: the save\_daily\_averages way saves the average for every single day in a list, this means if you do a lot of runs, it will save a lot of values. The other method is by only saving the current average, the min runs value and the max runs value. That way the list is only ever 3 long.

Average\_max\_size

* This list contains the average of the max\_sizes of all of the fires in km^2 on each run

Average\_detection\_size

* This list contains the average of the size of the fires in km^2 when they were detected for every fire on each run. Fires that weren’t detected are ignored.

Average\_report\_size

* This list contains the average size of the fires when they were reported in km^2 for every fire on each run. Fires that weren’t detected are ignored.

Average\_ignition\_time

* This list contains the average time in minutes between fires for fires to be ignited for each run. The first fire of each day instead is compared with the start\_of\_day time.

Average\_detection\_time

* This list contains the average time in minutes for the fires in each run to be detected after they were ignited. Non-detected fires are ignored.

Average\_report\_time

* This list contains the average time in minutes for the fires in each run to be reported after they were detected. Non-detected fires are ignored.

Num\_fires

* This list saves the total number of fires (both lightning and human) that occurred each run of the simulation.

Detection\_rate

* This list stores the probability that a fire on each given run was detected. It is found by dividing the number of detected fires on that run by the total number of fires.

Controlled\_rate

* This list stores the probability that a fire was controlled on each given run of the simulation. It was found by dividing the total number of controlled fires on a run by the total number of fires that occurred including the number of non-detected fires.

Average\_cost

* Not implemented, but would store the average cost of each day

Average\_travel\_time

* Average time the airtankers spent travel from bases to fires/bases and back for each run

Average\_travel\_distance

* The average distance each airtanker travelled on a given run of the simulation. Note that this includes the travelling behind lakes and fires, whereas the travel time does not.

Average\_wait\_time

* The average wait time of fires in queue for the airtankers. Note that if a fire was in queue for an airtanker and never ended up starting to get served by the airtanker, the wait time doesn’t count.

Average\_fight\_fire\_time

* The average time airtanker spent fighting fires on each run of the simulation

Average\_control\_time

* The average of the time it takes from when a fire requests an airtanker to the time when a fire is considered controlled for each run of the simulation. This ignores not controlled fires (including non-detected fires)

Airtanker

This class keeps track of all of the statistics related to each airtanker as well as keeps track of their performance so far for each run of the simulation.

NOT FINISHED DOCUMENTATION

Model Overview + Use

This section will go over quickly how the model works and how to use it. This model takes a forest data file to randomly generate fires in this forest over the length of a run based on user entered values for the expected number of lightning caused and human caused fires, as well as coefficients for various parameters in the arrival process equations. Lightning fires and human caused fires are each generated by their own individual generation process that generates fires based on a time-variant Poisson arrival process. This process is assumed to start at the user entered start of day time (note that does not mean the first fire starts at this time, only that the first fire cannot start before this time), and no fires can start after the end of day time. The arrival rate of fires varies throughout the day based on the equation arrival\_rate(t) = arrival\_rate\_mean – amp \* cos (2 \* pi \* (time – lag) / delta) where amp and lag are user entered values, time is the current time in the day and delta is the length of time from the start of the day to the end of day (the period where fires can start). This simulation first uses this to generate every fire that will happen during the run of the simulation. When a fire is created, its determined whether or not it is detected based on a uniform probability distribution, and if it is then a time for it to be detected is generated based on an exponential distribution, followed by a time to be reported from the time it was detected also from an exponential distribution. The location of the fire is determined by the forest data file. The probability of it being in each cell is found, then a random number is generated to determine the cell. Once the cell is determined, the cell’s area is used to assume the cell is a square, and the fire is randomly based on a uniform distribution placed somewhere in that cell. This is repeated for every fire that is generated, both lightning and human caused fires follow the same procedure, except some of the numbers used will be different between them.

Now that all of the fires report times are known, it is possible to start trying to control them with airtankers. Fires that were not detected are ignored during this part and grow until the end of the run. Bases locations as well as their number of airtankers are read from a file. Another file then outlines the simulation values to use for these airtankers which includes their service times and their model. The model of the airtanker is then looked up in another file to find some common values airtankers would share such as the flight speed. The model now has each airtanker as an instance of the Airtanker class, and an airtanker resource for each for use in simpy.

The model keeps skipping ahead to the next reported fire in line then requested the closest available airtanker that is within initial attack range. If no airtankers are available then the fire enters a queue for the first available airtanker within initial attack range of the fire. When a fire is selected to be controlled, the airtanker flys to the fire, then a random number based on a uniform distribution to determine the distance to the nearest lakes, and the number of drops is required. The time per drop is found from double the time it takes the airtanker to fly this distance (since it flies to the fire and back), + a drop time constant that can be different for each airtanker. The total time to control the fire is the number of drops required multiplied by the time for each drop. Once a fire is controlled, the airtanker will either return to its home base, go to the nearest base, or go straight to the next fire if it’s in queue, else go to the nearest base, depending on the rule given to the airtanker. Airtankers can only be sent to fight a fire if a number of conditions are met: the fire is within the initial attack range of the airtanker, it is within the airtankers operating times, the airtanker hasn’t gone over its travel or service time limits. If no airtanker is suitable to fight a fire, it will not be controlled. This whole process continues for the entire duration of the length of the run.